



2005

**Proceedings of the
4th International Modelica Conference**

March 7-8, 2005
Hamburg University of Technology
Hamburg-Harburg, Germany

Gerhard Schmitz (editor)

Volume 1

organized by
The Modelica Association and
the Department of Thermodynamics, Hamburg University of Technology

All papers of this conference can be downloaded from
<http://www.Modelica.org/Conference2005/papers/>

Proceedings of Modelica' 2005

Hamburg University of Technology,
Hamburg-Harburg, Germany, March 2005

Editor:

Prof. Dr.-Ing. G. Schmitz

Published by:

The Modelica Association (<http://www.Modelica.org>) and
the Department of Thermodynamics,
Hamburg University of Technology (<http://www.tt.tu-harburg.de>)

Printed by:

COPY-DRUCK

Gesellschaft für Digital- und Offsetdruck mbH

Preface

The first Modelica Conference took place October 2000 in Lund, Sweden. Since then, Modelica has been more and more established as a preferred modelling language for complex multi-domain systems. This is indicated by the high number of registrations from industry and science for the 4th International Modelica Conference which is held between March 7th and 8th 2005 at Hamburg University of Technology (TUHH). But it is also indicated by the number of excellent papers submitted to the program committee which made the task of selecting papers for oral and poster presentation very difficult and, last but not least, by the exhibition during the conference at which around 10 companies will present themselves. The proceedings contain the papers of the 60 oral presentations and 9 poster presentations given at the conference. The ability of Modelica as a multi-domain simulation language is demonstrated impressively by the various fields that are covered, e.g. digital electronic devices, hybrid electric power trains, waste water processes or thermodynamic applications.

With the special features of the Modelica language, e.g. object-oriented modelling and the ability to reuse and exchange models, Modelica has become – among other things – a further step towards of an integrated engineering design process. In some fields Modelica is being used as a standard platform for model exchange between suppliers and OEM's, for example in case of vehicle air conditioning systems.

A key issue for the success of Modelica is the continuous development of the Modelica language by the Modelica Association under strict observance of backward compatibility to previous versions. The broad base of private and institutional members of the Modelica Association as a non-profit organization ensures language stability and security in software investments.

The Modelica Conference 2005 was organized by the Modelica Association and by the Department of Thermodynamics of Hamburg University of Technology (TUHH), Germany. Together with the entire team of the local organizing committee I would like to wish all participants an excellent and fruitful conference.

Hamburg-Harburg, March 1, 2005

Gerhard Schmitz

Program Committee

- Prof. Gerhard Schmitz, Hamburg University of Technology, Germany (Program chair).
- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Germany.
- Dr. Francesco Casella, Politecnico di Milano, Italy.
- Dr. Hilding Elmqvist, Dynasim AB, Sweden.
- Prof. Peter Fritzson, University of Linköping, Sweden
- Prof. Martin Otter, DLR, Germany
- Dr. Michael Tiller, Ford Motor Company, USA
- Dr. Hubertus Tummescheit, Scynamics HB, Sweden

Local Organizers

- Gerhard Schmitz
- Katrin Prölb
- Wilson Casas
- Henning Knigge
- Jens Vasel
- Stefan Wischhusen
- TuTech Innovation GmbH

Contents

Volume 1

Session 1a

Mechanical Systems	11
I. Kossenکو , <i>Moscow State University of Service, Russia</i> : Implementation of Unilateral Multibody Dynamics on Modelica	13
F. Schiavo, G. Ferretti, L. Viganò , <i>Politecnico di Milano, Italy</i> : Object-Oriented Modelling and Simulation of Flexible Multibody Thin Beams in Modelica with the Finite Element Method	25
T. Pulecchi, M. Lovera , <i>Politecnico di Milano, Italy</i> : Object-oriented modelling of the dynamics of a satellite equipped with Single Gimbal Control Moment Gyros	35
H. Elmqvist, M. Otter, J. Díaz López , <i>Dynasim AB, Sweden; DLR Oberpfaffenhofen, Germany</i> : Collision Handling for the Modelica MultiBody Library	45

Session 1b

Chemical Systems and Thermodynamic Systems I	55
F. Cellier, À. Nebot , <i>ETH Zürich, Switzerland; Universitat Politecnica de Catalunya, Spain</i> : The Modelica Bond-Graph Library	57
J. Ungethüm , <i>German Aerospace Center, Stuttgart, Germany</i> : Fuel Cell System Modeling for Real-time Simulation	67
M. Rubio, A. Urquía, L. González, D. Guinea, S. Dormido , <i>UNED and CSIC, Madrid, Spain</i> : FuelCellLib - A Modelica Library for Modeling of Fuel Cells	75
E. Larsdotter Nilsson, P. Fritzson , <i>Linköping University, Sweden</i> : A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems	85

Session 1c

Methods I	95
F. Casella , <i>Politecnico di Milano, Italy</i> : Exploiting Weak Dynamic Interactions in Modelica	97
H. Olsson, H. Tummescheit, H. Elmqvist , <i>Dynasim AB; Modelon AB, Sweden</i> : Using Automatic Differentiation for Partial Derivatives of Functions in Modelica	105
L. Saldamli, B. Bachmann, P. Fritzson, H. Wiesmann , <i>Linköping University, Sweden; FH Bielefeld, Germany; ABB, Switzerland</i> : A Framework for Describing and Solving PDE Models in Modelica	113
P. Aronsson, P. Fritzson , <i>Linköping University, Sweden</i> : A Task Merging Technique for Parallelization of Modelica Models	123

Session 2

Poster session	129
D. Aiordachioaie, M. Munteanu, E. Ceanga, <i>University of Galati, Romania:</i> Some Results on Neutral Modelling of the Steel Continuous Casting Process	131
C. Clauß, E. Erler, <i>Fraunhofer Institute, Dresden, Berufliches Schulzentrum, Freital, Germany:</i> Switched Capacitor Simulation with Modelica	141
K. Berg, K. Nyström, <i>Linköping University, Sweden:</i> Hydrological modeling in Modelica	149
P. Harman, <i>Ricardo UK Ltd.:</i> Visualisation of Model Transformation Algorithms for a Modelica Translator	155
C. Martin, A. Urquía, S. Dormido, <i>UNED Madrid, Spain:</i> Modeling of Interactive Virtual Laboratories with Modelica	159
K. Nyström, P. Aronsson, P. Fritzson, <i>Linköping University, Sweden:</i> Parallelization in Modelica	169
S.E. Pohl, J. Ungethüm, <i>DLR Stuttgart, Germany:</i> A Simulation Management Environment for Dymola	173
A. Siemers, I. Nakhimovski, D. Fritzson, <i>Linköping University, Sweden:</i> Meta-modelling of Mechanical Systems with Transmission Line Joints in Modelica	177

Session 3a

Automotive Simulation I	183
J. Eborn, H. Tummescheit, K. Prölb, <i>Modelon AB, Sweden; TUHH, Germany:</i> Air-Conditioning - a Modelica Library for Dynamic Simulation of AC Systems	185
D. Limperich, M. Braun, G. Schmitz, K. Prölb, <i>DaimlerChrysler AG; TUHH, Germany:</i> System Simulation of Automotive Refrigeration Cycles	193
M. Hommel, <i>Volkswagen AG, Germany:</i> First Results in Cluster Simulation of Alternative Automotive Drive Trains	201

Session 3b

Thermodynamic Systems II	211
C. Kral, A. Haumer, M. Plainer, <i>Arsenal Research, Vienna, Austria:</i> Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library	213
S. Micheletti, S. Perotto, F. Schiavo, <i>Politecnico di Milano, Italy:</i> Modelling Heat Exchangers by the Finite Element Method with Grid Adaption in Modelica	219
W. Steinmann, <i>DLR Stuttgart, Germany:</i> Calculation of Thermophysical Properties in the Modelica Library TechThermo	229
M. Tiller, <i>Ford Motor Company, USA:</i> Development of a Simplified Transmission Hydraulics Library based on Modelica.Fluid	237

Session 3c

Methods II	245
B. Johansson, P. Krus, <i>Linköping University, Sweden:</i> Probabilistic Analysis and Design Optimization of Modelica Models	247
H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, M. Otter, <i>Dynasim AB, Sweden; DLR, Oberpfaffenhofen, Germany:</i> Optimization for Design and Parameter Estimation	255
M. Thümmel, G. Looye, M. Kurze, M. Otter, J. Bals, <i>DLR Oberpfaffenhofen, Germany:</i> Nonlinear Inverse Models for Control	267

P. Bunus , <i>Linköping University, Sweden</i> : An Empirical Study on Debugging Equation-Based Simulation Models	281
--	-----

Volume 2

Session 4a

Automotive Simulation II	299
L. Morawietz, S. Risse, H. Zellbeck, H. Reuss, T. Christ , <i>TU Dresden, University of Stuttgart, BMW Group, Germany</i> : Modeling an automotive power train and electrical power supply for HiL applications using Modelica	301
E. Surewaard, M. Thele , <i>Ford Forschungszentrum Aachen, RWTH Aachen University, Germany</i> : Modelica in Automotive Simulations - Powernet Voltage Control during Engine Idle	309
T. Bünte, A. Sahin, N. Bajcinca , <i>DLR Oberpfaffenhofen, University of Siegen, Germany</i> : Inversion of Vehicle Steering Dynamics with Modelica/Dymola	319

Session 4b

Thermodynamic Systems III	329
W. Steinmann, J. Buschle , <i>DLR Stuttgart, Germany</i> : Analysis of thermal storage systems using Modelica	331
S. Wischhusen, G. Schmitz , <i>TUHH, Germany</i> : Exergy-analysis of a direct-evaporating cooling plant with heat reclaim	339
T. Ziehn, G. Reichl, E. Arnold , <i>TU Ilmenau; Fraunhofer Institute Ilmenau, Germany</i> : Application of the Modelica library WasteWater for optimisation purposes	351

Session 4c

Tools I	357
G. Ferretti, M. Gritti, G. Magnani, G. Rizzi, P. Rocco , <i>Politecnico di Milano, Italy</i> : Real-Time Simulation of Modelica Models under Linux / RTAI	359
M. Najafi, S. Furic, R. Nikoukhah , <i>Imagine; INRIA-Rocquencourt, France</i> : SCICOS: a general purpose modeling and simulation environment	367
R. Dorling , <i>Advanced Dynamic Systems, Peterborough, U.K.</i> : Model Validation and the Modelica Language	375

Session 5a

Engines	383
J. Batteh, M. Tiller, A. Goodman , <i>Ford Motor Company, USA</i> : Monte Carlo Simulations for Evaluating Engine NVH Robustness	385
S.E. Pohl, M. Gräf , <i>DLR Stuttgart, Germany</i> : Dynamic Simulation of a Free-Piston Linear Alternator in Modelica	393

Session 5b

Thermodynamic Systems IV	401
T. Hirsch, W. Steinmann, M. Eck , <i>DLR Stuttgart, Germany</i> : Simulation of transient two-phase flow in parabolic trough collectors using Modelica	403
L.J. Yebra, M. Berenguel, S. Dormido, M. Romero , <i>CIEMAT-PSA; Universidad de Almeria; UNED, Spain</i> : Modelling and Simulation of Central Receiver Solar Thermal Power Plants	413

A. Cammi, F. Casella, M. Ricotti, F. Schiavo, <i>Politecnico di Milano, Italy</i>: Object-Oriented Modeling, Simulation and Control of the IRIS Nuclear Power Plant with Modelica	423
Session 5c	
Tools II	433
A. Pop, P. Fritzson, <i>Linköping University, Sweden</i>: A Portable Debugger for Algorithmic Modelica Code	435
O. Johansson, A. Pop, P. Fritzson, <i>Linköping University, Sweden</i>: ModelicaDB - A Tool for Searching, Analysing, Crossreferencing and Checking of Modelica Libraries	445
Session 6a	
Automotive Simulation III	455
C. Schweiger, M. Dempsey, M. Otter, <i>DLR Oberpfaffenhofen, Germany; Claytex Services Ltd, UK</i>: The PowerTrain Library: New Concepts and New Fields of Application	457
P. Bengtsson, H. Jansson, N. Pettersson, T. Sandberg, <i>Scania CV AB, Sweden</i>: Development of a Modelica Heavy Vehicle Modeling Library	467
M. Tiller, <i>Ford Motor Company, USA</i>:: Evaluation of Motor and Battery Requirements for Hybrid-Electric Powertrains during Cranking	477
Session 6b	
Thermodynamic Systems V	485
W. Casas, K. Prölß, G. Schmitz, <i>TUHH, Germany</i>: Modeling of Desiccant Assisted Air Conditioning Systems	487
B. Oehler, <i>Airbus Deutschland GmbH, Germany</i>: Modeling and Simulation of Global Thermal and Fluid Effects in an Aircraft Fuselage	497
Session 6c	
Modelica Language	507
J. Mauss, <i>DaimlerChrysler AG, Berlin, Germany</i>: Modelica Instance Creation	509
P. Fritzson, A. Pop, P. Aronsson, <i>Linköping University, Sweden</i>: Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica	519
C. Nytsch-Geusen et. al., <i>Fraunhofer Institutes, Germany</i>: MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics	527
Session 7a	
Electrical Systems	537
C. Clauß, U. Donath, A. Schneider, E. Weber, <i>Fraunhofer Institute for Integrated Circuits, University of Applied Sciences Mittweida, Germany</i>: Standard Package Modelica.Electrical. Digital	539
C. Kral, A. Haumer, <i>Arsenal Research, Austria</i>: Modelica libraries for dc machines, three phase and polyphase machines	549
Th. Bödrich, Th. Roschke, <i>Dresden University of Technology; Saia-Burgess Dresden GmbH, Germany</i>: A Magnetic Library for Modelica	559

Session 7b	
Real-Time and Reactive Systems	567
M. Otter, K.-E. Årzén, I. Dressler , <i>DLR Oberpfaffenhofen, Germany; Lund Institute of Technology, Sweden</i> : StateGraph-A Modelica Library for Hierarchical State Machines	569
J. Bäckman, M. Edvall , <i>Metso Paper AB, Sweden</i> : Using Modelica and Control Systems for Real-time Simulations in the Pulp & Paper industry	579
Session 7c	
Modelica Interfaces	585
J. Köhler, A. Banerjee , <i>ZF Friedrichshafen AG, Germany</i> : Usage of Modelica for transmission simulation in ZF	587
M. Tiller , <i>Ford Motor Company, USA</i> : Implementation of a Generic Data Retrieval API for Modelica	593
H. Olsson , <i>Dynasim AB, Sweden</i> : External Interface to Modelica in Dymola	603

Index of Authors

Årzén, Karl-Erik	569	Elmqvist, Hilding	45, 105, 255
Aiordachioaie, Dorel	131	Erler, Elisabeth	141
Arnold, Eckhard	351	Ernst, Thilo	527
Aronsson, Peter	123, 169, 519	Ferretti, Gianni	25, 359
Bäckman, Johan	579	Fritzson, Dag	177
Bödrich, Thomas	559	Fritzson, Peter 85, 113, 123, 169, 435, 445, 519	
Bünthe, Tilman	319	Furic, S.	367
Bachmann, Bernhard	113	González, Leandro	75
Bajcina, Naim	319	Goodman, Adam	385
Bals, Johann	267	Gräf, Markus	393
Banerjee, Alexander	587	Gritti, Marco	359
Batteh, John J.	385	Guinea, Domingo	75
Bengtsson, Per	467	Harman, Peter	155
Berenguel, M.	413	Haumer, Anton	213, 549
Berg, Karin	149	Hirsch, Tobias	403
Brück, Dag	255	Holm, Andreas	527
Braun, Marco	193	Hommel, Mathias	201
Bunus, Peter	281	Jansson, Henrik	467
Buschle, Jochen	331	Johansson, Björn	247
Cammi, Antonio	423	Johansson, Olof	445
Casas, Wilson	487	Joos, Dieter	255
Casella, Francesco	97, 423	Köhler, Jochen	587
Ceanga, Emil	131	Kossenko, Ivan	13
Cellier, François E.	57	Kral, Christian	213, 549
Christ, Thomas	301	Krus, Petter	247
Clauß, Christoph	141, 539	Kurze, Matthias	267
Díaz López, José	45	Larsdotter Nilsson, Emma	85
Dempsey, Mike	457	Leopold, Jürgen	527
Doll, Ulrich	527	Limperich, Dirk	193
Donath, Ulrich	539	Looye, Gertjan	267
Dorling, Richard	375	Lovera, Marco	35
Dormido, Sebastian	75, 159, 413	Magnani, Gianantonio	359
Dressler, Isolde	569	Martin, Carla	159
Eborn, Jonas	185	Mattes, Alexander	527
Eck, Markus	403		
Edvall, Mattias	579		

Mattsson, Sven Erik	255	Surewaard, Erik	309
Mauss, Jakob	509	Thümmel, Michael	267
Micheletti, Stefano	219	Thele, Marc	309
Morawietz, Lutz	301	Tiller, Michael M.	237, 385, 477, 593
Munteanu, Mihai	131	Tummescheit, Hubertus	105, 185
Najafi, Masoud	367	Ungethüm, Jörg	67, 173
Nakhimovski, Iakov	177	Urquía, Alfonso	75, 159
Nebot, Àngela	57	Vetter, Matthias	527
Nikoukhah, R.	367	Viganò, Luca	25
Nordwig, André	527	Weber, Enrico	539
Nouidui, Thierry	527	Wiesmann, Hansjürg	113
Nyström, Kaj	149, 169	Wischhusen, Stefan	339
Nytsch-Geusen, Christoph	527	Wittwer, Christof	527
Oehler, Bettina	497	Yebrá, Luis Jose	413
Olsson, Hans	105, 255, 603	Zellbeck, Hans	301
Otter, Martin	45, 255, 267, 457, 569	Ziehn, Tilo	351
Perotto, Simona	219		
Pettersson, Niklas	467		
Plainer, Markus	213		
Pohl, Sven-Erik	173, 393		
Pop, Adrian	435, 445, 519		
Pröbß, Katrin	185, 193, 487		
Pulecchi, Tiziano	35		
Reichl, Gerald	351		
Reuss, Hans-Christian	301		
Ricotti, Marco E.	423		
Risse, Silvio	301		
Rizzi, Gianpaolo	359		
Rocco, Paolo	359		
Romero, M.	413		
Roschke, Thomas	559		
Rubio, Miguel A.	75		
Sahin, Akin	319		
Saldamli, Levon	113		
Sandberg, Tony	467		
Schiavo, Francesco	25, 219, 423		
Schmidt, Gerhard	527		
Schmitz, Gerhard	193, 339, 487		
Schneider, André	539		
Schneider, Peter	527		
Schwarz, Peter	527		
Schweiger, Christian	255, 457		
Siemers, Alexander	177		
Steinmann, Wolf-Dieter	229, 331, 403		

Session 1a

Mechanical Systems

Implementation of Unilateral Multibody Dynamics on Modelica

Ivan I. Kossenko

Moscow State University of Service, Department of Engineering Mechanics
Glavnaya str., 99, Cherkizovo-1, Moscow reg., 141221, Russia

Abstract

The problems of computer modeling and simulation of dynamics for multibody systems consisting of rigid bodies with unilateral constraints (MBSUC) are considered in the scope of the obstacles to overcome ones related to the variation of structure for equations of motion. The approach to modeling the MBSUC dynamics based on Modelica language is described.

The approach allowing to avoid the growth of the model structural complexity is described. This approach actively uses the algorithmic features of Modelica and its Dymola compiler. On this way the large number of objects corresponding to different closed systems of DAEs (states of hybrid automata) is replaced by only one object. For this object constraint components vary their states dynamically during the simulation process.

Another problem of the similar level of complexity relates to the accuracy of simulation is solved here with the set of special regularization procedures. These procedures concern particularly transitions of the unilateral constraint: from disconnected state to contact, from rolling to slipping.

Other methods to improve the quality of the MBSUC dynamics simulation are also under consideration.

Keywords: unilateral dynamics; multibody systems; simulation; dry friction; impacts; regularization; acausal modeling

1 Introduction

Mechanical system subjected to unilateral constraints exhibits behavior considerably more complicated than the system subjected to the bilateral ones. One can find in such a case new dynamical properties connected with irregular character of appropriate systems of DAEs. Let us develop the approach proposed in [1]. There the Modelica library of classes oriented to simulation the sparse dynamics of multibody systems has been developed. We can consider now this library as a

set of the new generation models allowing description of unilateral constraints.

Let us suppose that some of constraints are unilateral. For definiteness and simplicity we state the following assumptions: (a) unilateral constraint is implemented as a contact of outer surfaces bounding two rigid bodies; (b) surfaces supposed being regular i. e. the normal vector is always properly defined; (c) the contacting surfaces interact within the model of Coulomb friction for continuous motions as well as for impacts.

For simplicity we investigate the MBSUC comprising only two bodies, A , and B . Moreover, we suppose that the body A is a fixed horizontal surface, and the heavy convex body B is bounded by ellipsoidal surface. These assumptions are not obstacles for generality of the developed MBSUC models.

2 Basic Ideas

According to the approach applied in [1] let us represent the constraint as an object providing information communications between the objects of bodies A and B . Such communications are implemented indirectly using the kinematic and wrench connectors. Information communications are “filtrated” through the mechanism of constraint equations encapsulated in the object C , see Figure 2.1.

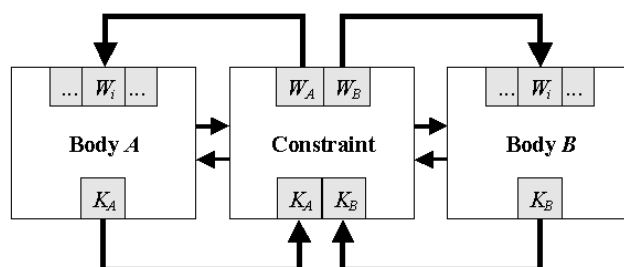


Figure 2.1: Architecture of Unilateral Constraint

Besides the bidirected connections applied in [1] let us add to the model the set of directed connections. Assume that these connections are able to transmit

the impact signals arising in objects of unilateral constraints all over the MBSUC, namely throughout its connected components. These signals play role of strobing ones for recalculation of velocities in the MBSUC.

The nature of unilateral constraint allows us to describe it with the fundamental state variable. This variable takes one of three values: “Flight”, “Sliding”, “Rolling” at any time instant. The sense of the enumerated values is transparent. The state “Flight” means that the constraint is not stretched at the considered instant, i. e. the bodies aren’t in touch and freely fly one relative to another. As state variable has one of values “Sliding” or “Rolling” then bodies supposed to be in a contact. The difference is that the first state permits the relative slipping of the bodies but the second one doesn’t.

Example 2.1 Consider the set of n balls in a billiard pool. The system comprises $n + 1$ rigid bodies: n balls and the surface of the pool table. Vertical surfaces around the table are neglected for simplicity. All bodies enumerated can encounter mutually, slip, and roll. The correct description of this MBSUC involves the specification of $m = n(n + 1)/2$ unilateral constraints. Since each constraint can be in one of three states, then the whole MBSUC comprises $3^m = 3^{\frac{n(n+1)}{2}}$ states. For the pool with three balls we obtain the total value of $3^6 = 729$ states.

2.1 Constraint Geometry

Let us use here the same as in [1] the dynamics of a rigid body translational–rotational motion. However the representation of mechanical constraint model undergoes here essential changes. We use the so called complementarity rules [2] as a base for the unified description of the unilateral constraint. By virtue of complementarity rules any constraint is always defined by the three scalar equations. In order to derive these equations let us consider the local geometry of the problem, see Figure 2.2.

The base body of MBSUC supposed to be connected with the absolute frame $O_0x_0y_0z_0$ (AF) fixed in the inertial space, $O_\alpha x_\alpha y_\alpha z_\alpha$ is the frame BF_α fixed in the body $\alpha \in \{A, B\}$. The outer surfaces Σ_α are defined by the equations

$$f_\alpha(\mathbf{r}_\alpha) = 0 \quad (\alpha = A, B).$$

with respect to appropriate BF_α whose axes are coincident to the principal central axes of inertia. In AF

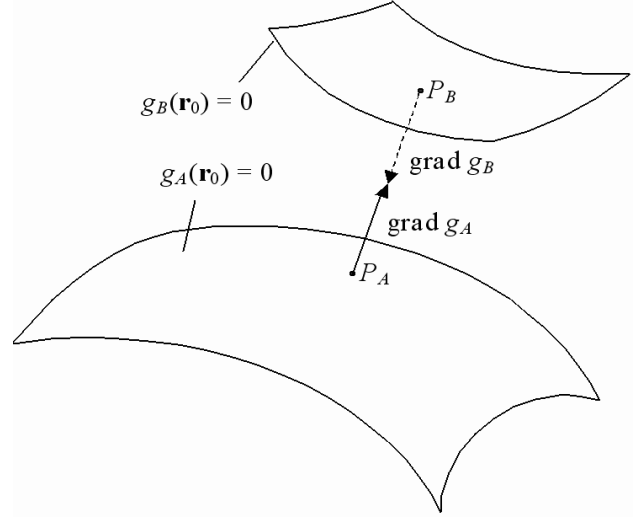


Figure 2.2: Area of Constraint

these the equations read

$$g_\alpha(\mathbf{r}_0) = f_\alpha [T_\alpha^* (\mathbf{r}_0 - \mathbf{r}_{O_\alpha})] = 0 \quad (\alpha = A, B).$$

Here $\mathbf{r}_{O_A} = O_0O_A$, $\mathbf{r}_{O_B} = O_0O_B$, T_A, T_B are the orthogonal matrices determining orientation of the BF_A and BF_B with respect to the AF . An asterisk denotes the matrix transposition. The functions $g_A(\mathbf{r}_0)$, $g_B(\mathbf{r}_0)$ depend upon the time indirectly through the variables $\mathbf{r}_A, \mathbf{r}_B, T_A, T_B$.

The constraint object of our model has to compute at each current instant the points $P_A \in A$ and $P_B \in B$ realizing the minimal distance between the bodies. These points depend on relative orientation of the bodies. By virtue of above assumptions such points are to be evaluated in a unique way. Denote by $\mathbf{r}_{P_A}, \mathbf{r}_{P_B}$ the radii vectors of these points with respect to AF . The simple geometric reasons imply the following system of algebraic equations

$$\begin{aligned} \text{grad } g_A(\mathbf{r}_{P_A}) &= \lambda \cdot \text{grad } g_B(\mathbf{r}_{P_B}), \\ \mathbf{r}_{P_A} - \mathbf{r}_{P_B} &= \mu \cdot \text{grad } g_B(\mathbf{r}_{P_B}), \\ g_A(\mathbf{r}_{P_A}) &= 0, \\ g_B(\mathbf{r}_{P_B}) &= 0. \end{aligned} \quad (2.1)$$

The gradients of the functions g_A and g_B read

$$\text{grad } g_\alpha(\mathbf{r}_{P_\alpha}) = T_\alpha \text{grad } f_\alpha [T_\alpha^* (\mathbf{r}_{P_\alpha} - \mathbf{r}_{O_\alpha})], \quad (2.2)$$

where $\alpha = A, B$. The system (2.1) consists of eight scalar equations with respect to eight scalar variables: $x_{P_A}, y_{P_A}, z_{P_A}, x_{P_B}, y_{P_B}, z_{P_B}, \lambda, \mu$, where λ, μ are auxiliary variables. The equations (2.1) are in use either without or with a presence of the contact of bodies A, B . In the latter case the equation $\mu = 0$ is in use instead of one of the surfaces equations.

According to computational experience it is more reliable and convenient to use the equations of constraints (2.1) in a differential form. Such an approach is used frequently also for analyzing of properties of mechanical systems.

Normal vector

$$\mathbf{n}_A = \text{grad } g_A / |\text{grad } g_A| \quad (2.3)$$

will play an important role in the further course. Normal for an outer surface of the body A is chosen here for definiteness. One can use the vector \mathbf{n}_B as well.

2.2 Complementarity Rules

Let us perform a unified description of the unilateral constraint using kinematic and/or force equations. Denote by \mathbf{F}_A the force acting on the body A from the body B . And by \mathbf{F}_B denote the force acting on the body B from one of A vice versa. Each force cited acts at the point P_α , $\alpha = A, B$. In addition, let us introduce auxiliary notations

$$F_{An} = (\mathbf{F}_A, \mathbf{n}_A), \quad \mathbf{F}_{A\tau} = \mathbf{F}_A - F_{An}\mathbf{n}_A,$$

$$\mathbf{v}_r = \mathbf{v}_{P_A} - \mathbf{v}_{P_B}, \quad v_{rn} = (\mathbf{v}_r, \mathbf{n}_A), \quad \mathbf{v}_{r\tau} = \mathbf{v}_r - v_{rn}\mathbf{n}_A. \quad (2.4)$$

If the bodies are not in touch and the constraint is in the state “Flight” then the force of reaction is equal to zero. Thus we have three scalar equations. To unify the system of constraint equations and to take into account arbitrary directions of the normal \mathbf{n}_A let us introduce auxiliary scalar variable κ such that

$$F_{An} = 0, \quad \mathbf{F}_{A\tau} - \kappa\mathbf{n}_A = \mathbf{0}.$$

Then the system of four equation with four unknown variables $F_{Ax}, F_{Ay}, F_{Az}, \kappa$ is obtained.

If the bodies are in touch then the condition $F_{An} = 0$ is substituted by the kinematic one $v_{An} = 0$. States “Sliding” and “Rolling” differ from each other by conditions in a tangent plane. Implementation of the Coulomb friction model is supposed for the simplicity. Then the equation of the force balance in the tangent space reads

$$\mathbf{F}_{A\tau} - d \cdot F_{An}\mathbf{v}_{r\tau} / |\mathbf{v}_{r\tau}| - \kappa\mathbf{n}_A = \mathbf{0}, \quad (2.5)$$

where d is the coefficient of friction.

For rolling the tangent velocity is:

$$\mathbf{v}_{A\tau} - \kappa\mathbf{n}_A = \mathbf{0}.$$

2.3 Regularization of the Coulomb Friction

In the case of sliding the model equation (2.5) “works” properly if the relative velocity isn’t very small. However the problem of regularization for the equation of constraint (2.5) arises at the instance of transition from “Rolling” to “Sliding”. It turns out that one can apply here the known approximation for Coulomb’s friction using regularized expression for the tangent force

$$\mathbf{F}_{A\tau} - \kappa\mathbf{n}_A = d \begin{cases} F_{An}\mathbf{v}_{r\tau} / |\mathbf{v}_{r\tau}| & \text{as } |\mathbf{v}_{r\tau}| > \delta, \\ F_{An}\mathbf{v}_{r\tau} / \delta & \text{as } |\mathbf{v}_{r\tau}| \leq \delta, \end{cases}$$

where one supposes that $\delta \ll 1$.

It is known [3] that in this case the solution of the regularized problem remains close to the solution of the original one at the asymptotically large time intervals. Implementation and further simulation show that this closeness holds with the very high degree of accuracy. Such an approach resolves completely the problem of modeling for accurate transitions between states of “Sliding” and “Rolling”.

2.4 Simulation of Impacts

Let us suppose that the unilateral constraint is allowed to undergo an impact in any possible states. In state “Flight” the impact arises at the instant of bodies contact if normal component of the relative velocity v_{rn} for encountering points is not very close to zero. It is the case of the so called direct impact. However in MBSUC consisting of several bodies impact pulses can propagate through the connected components of the system and force it to disconnect of any constraints. This leads to the switch of the whole MBSUC to an another state. Such a case we can consider as an indirect impact.

The constraint model proposed allows the possibility both direct and indirect impacts. Let us consider the equations of the impact theory encapsulated in the objects of the constraint structure, see Figure 2.1. All these algebraic equations are carried out for all the time of simulation. From time to time impact events arising inside the differential part of the whole model strobe “reading” of impact increments for the velocities from the impact algebraic subsystem and instantaneous change of velocities inside the dynamical subsystem.

Thus the equations

$$m\Delta\mathbf{v} = \mathbf{S}, \quad I\Delta\boldsymbol{\omega} = \mathbf{T}, \quad (2.6)$$

are encapsulated in objects A and B of the “Rigid Body” class. Here $\Delta\mathbf{v}$, $\Delta\boldsymbol{\omega}$ are the increments of the

center of mass velocity and angular velocity of the body, \mathbf{S} , \mathbf{T} are correspondingly the total impulse and angular impulse acting on the rigid body belonging to MBSUC. Note that the first equation of the system (2.6) is written in AF . The second one is written, as usually, in appropriate BF_α .

Constraint object, C in Figure 2.1, encapsulates the simplest impact model with dry friction and the Newtonian model for the normal impact

$$\begin{aligned}
\Delta \mathbf{v}_{P_\alpha} &= \Delta \mathbf{v}_{O_\alpha} + [\Delta \boldsymbol{\omega}_\alpha, \mathbf{r}_{P_\alpha} - \mathbf{r}_{O_\alpha}], \\
\Delta v_{P_\alpha n} &= (\Delta \mathbf{v}_{P_\alpha}, \mathbf{n}_A), \\
\Delta v_{rn} &= -(1+k)v_{rn}, \\
\Delta v_{P_B n} &= \Delta v_{P_A n} - \Delta v_{rn}, \\
\Delta \mathbf{v}_{P_\alpha \tau} &= \Delta \mathbf{v}_{P_\alpha} - \Delta v_{P_\alpha n} \mathbf{n}_A, \\
S_{An} &= (\mathbf{S}_A, \mathbf{n}_A), \\
\mathbf{S}_{A\tau} &= \mathbf{S}_A - S_{An} \mathbf{n}_A,
\end{aligned} \tag{2.7}$$

where the restitution coefficient is equal to k .

To make the model of impact with friction more realistic we apply the simplified formula for the impact impulse. It is similar to the regularized formula for the tangent force in the case of slipping with dry friction. Let us note that there exist more realistic models of impact with the Coulomb friction [4] (see [5] for comprehensive survey). However they are much more complicated. These models are suited for the single impact of two bodies only. But we are interested in a general case of MBSUC consisting of several bodies not only of two ones.

2.5 Regularization of Transition between the States of Flight and of the Contact

The most important property of the model developed consists of the possibility of exact calculation of impact instants and the instants of the change of state. This property plays a crucial role for the quality of the model. The landing on the constraint is possible in particular if restitution coefficient satisfies the condition $k < 1$. In this case time intervals between impacts tend to zero as well as the amplitudes of jumps after successive impacts. Thus for exact determination of the landing instant there exists a technological restriction: limit of smallness for the value of the integrator time step.

The change of independent variable, which regularizes the time, gives the resolution of the problem. Indeed, let us consider approximate model of dynamics in a vicinity of the landing instant. In this case we can restrict ourselves to analysis of the relative motion for points P_A and P_B in normal direction. Let us assume

that the normal relative acceleration $a_{rn} = dv_{rn}/dt$ approximately is a constant. Then the relative normal motion of the points A and B is similar to the bouncing ball in field of constant acceleration a_{rn} in the vicinity of the landing instant.

Thus the height of jumps obeys the known formula $h = 0.5v_{rn}^2/a_{rn}$. Hence the instant of transition to contact is defined by the condition when h becomes less than the given value of the tolerance for the constraint feasibility.

The time between two impacts can be also approximately computed with the known formula $T = 2|v_{rn}/a_{rn}|$. This value tends to zero with each new impact leading to the loss of an accuracy of simulation.

Way out of a situation is the transfer to new independent variable τ such that the duration between successive impacts would stay of order one. The simplest solution of this problem is the map $t \mapsto \tau$ according to the scalar differential equation $dt/d\tau = T$. Such an approach is found to be sufficiently reliable. Moreover it is easy to control the accuracy of the model.

3 Implementation

When constructing the model of MBSUC the main task is to develop the Modelica code allowing to switch different constraint states inside the same object, see Figure 2.1. It was found the problem can be resolved using so called acausal [6] approach to build the system of DAEs for the resulting model. Alternatively if one uses the causal approach then the structural complexity of a model code can increase avalanchely. To make sure of this it is sufficient to remind our example about three balls in a billiard pool. If each state of the mechanical system corresponding to the closed system of DAEs is instantiated as an object inside the container of the hybrid automata model then very soon developer will encounter with the problem of large complexity even for a number of balls small enough. Conversely within the acausal approach there exists a possibility to construct the model of MBSUC at the same complexity level as for mechanical system subjected to bilateral constraints only. In this case all variety of MBSUC states is provided by internal capabilities of the constraint objects and, as usually it is implemented with help of an analytical precompiler.

3.1 Connectors

To connect objects we use the classes of kinematic and wrench ports as before [1]. In addition, new connec-

tors are able to transport data of the velocities increments and the impact impulses. Codes of the corresponding derived classes read

```
connector KinematicPortImpacts
  extends KinematicPort;
  SI.Velocity Deltav[3];
  SI.AngularVelocity Deltaomega[3];
end KinematicPortImpacts;

connector WrenchPortImpacts
  extends WrenchPort;
  SI.Impulse ImpactForce[3];
  SI.AngularImpulse ImpactTorque[3];
end WrenchPortImpacts;
```

To transmit impact signals throughout the MBSUC one uses standard signal input and output ports:

```
Interfaces.BooleanInPort,
Interfaces.BooleanOutPort.
```

from the library `Modelica.Blocks`.

3.2 Bodies

This category classes were modified to take into account the possibility of impacts in MBSUC. The base class `RigidBody` considered in [1] has been slightly rearranged and now reads as

```
partial model RigidBody
  replaceable KinematicPort OutPort;
  ...
  Real Active(start=1);
equation
  der(Active) = 0;
  der(r) = Active*v;
  der(v) = Active*a;
  der(q) = Active*0.5*QMult(q,
    {0, omega[1], omega[2], omega[3]});
  der(omega) = Active*epsilon;
  ...
end RigidBody;
```

Dots here mean those parts of the `RigidBody` class from the previous version which haven't been reconstructed. In addition, one can see easily that the time of dynamics can be "stopped" here at all. This can be done with auxiliary variable `Active` putting its value equal to zero. In this case the model will be transformed from dynamical to the static one, which is defined by algebraic equations only.

Declaration `replaceable` is aimed to provide the possibility of choice between modes of simulation with or without impacts.

To implement impact calculations one uses the following class

```
partial model RigidBodyImpacts
  extends RigidBody(redeclare
    KinematicPortImpacts OutPort);
  SI.Velocity Deltav[3];
  SI.AngularVelocity Deltaomega[3];
  SI.Impulse ImpactForce[3];
  SI.AngularImpulse ImpactTorque[3];
  Boolean Impact;
  SI.Force F1[3];
  SI.Torque M1[3];
  WrenchPortImpacts InPort1;
  BooleanInPort InImpactSignal1;
  BooleanOutPort OutImpactSignal1;
equation
  F = InPort1.F + F1;
  M = InPort1.M + cross(InPort1.P - r,
    InPort1.F) + M1;
  OutImpactSignal1.signal[1] = false;
  Impact = false or
    InImpactSignal1.signal[1];
  OutPort.Deltav = Deltav;
  OutPort.Deltaomega = T*Deltaomega;
  OutPort.Deltav = Deltav;
  OutPort.Deltaomega = T*Deltaomega;
  m*Deltav = InPort1.ImpactForce +
    ImpactForcel;
  I*Deltaomega = transpose(T)*
    (InPort1.ImpactTorque + cross(
      InPort1.P - r, InPort1.ImpactForce)
    + ImpactTorquel);
end RigidBodyImpacts;
```

Since this class is introduced to process impacts then it possesses at least one wrench port supposed for at least one unilateral constraint, which is a potential source of impacts. The class, cited rather its object can be instantiated in the models being developed according to any causality principle. In case of the acausal approach such class is to be completed by the following model

```
model RigidBodyImpactsAcausal
  extends RigidBodyImpacts;
equation
  when Impact then
    reinit(v, v + Deltav);
    reinit(omega, omega + Deltaomega);
  end when;
end RigidBodyImpactsAcausal;
```

providing a self-governing possibility of the object to recalculate velocities at impact. In case of the causal approach such the calculation should be instantiated outside the object of the MBSUC state. Note that in implementations in derived classes for the bodies of MBSUC we can instantiate any number of wrench port objects necessary for constraints of the MBSUC model.

3.3 Constraints

On the same way as for `RigidBody` class the base model `Constraint` has been slightly rearranged and now has the following modified form

```
partial model Constraint
  parameter Integer ConstraintNo = 1;
  replaceable KinematicPort InPortA;
  replaceable WrenchPort OutPortA;
  replaceable KinematicPort InPortB;
  replaceable WrenchPort OutPortB;
equation
  ...
end Constraint;
```

Then one can construct easily the constraint base model taking into account impacts of bodies in the form

```
partial model ConstraintImpacts
  extends Constraint(
    redeclare KinematicPortImpacts
      InPortA,
    redeclare WrenchPortImpacts
      OutPortA,
    redeclare KinematicPortImpacts
      InPortB,
    redeclare WrenchPortImpacts
      OutPortB);
equation
  OutPortA.ImpactForce +
  OutPortB.ImpactForce = zeros(3);
  OutPortA.ImpactTorque +
  OutPortB.ImpactTorque = zeros(3);
end ConstraintImpacts;
```

Now it is time to construct a base model for the unilateral constraint satisfying our assumptions stated earlier and processing impact events correctly

```
model UnilateralConstraintAcausal
  extends ConstraintImpacts;
  parameter Real k;
  parameter Real f;
  parameter SI.Velocity delta;
  UnilateralConstraintState State;
  Boolean Impact;
  Boolean NormalImpact;
  Boolean NormalImpactIndicator;
  Boolean ImpactMask;
  Real[3] normA;
  SI.Impulse ImpactForcen;
  SI.Impulse[3] ImpactForcet;
  SI.Impulse kappa;
  SI.Acceleration[3] arA;
  SI.Acceleration[3] arB;
  SI.Acceleration[3] rela;
```

```
SI.Acceleration relan;
Real Active(start=1);
...
algorithm
  when relan > 0 and not ImpactMask
  then
    ImpactMask := true;
  end when;
  when State == 0 and pre(State) <> 0
  then
    ImpactMask := false;
  end when;
equation
  ...
  NormalImpactIndicator = if mu < 0
    and State == 0 and ImpactMask
  then true else false;
  NormalImpact = edge(
    NormalImpactIndicator);
  Impact = if noEvent(NormalImpact)
  then true else false;
  Active*arA = der(vrA);
  Active*arB = der(vrB);
  rela = arA - arB;
  relan = rela*normA;
  ImpactForcen = OutPortA.ImpactForce*
    normA;
  ImpactForcet = OutPortA.ImpactForce -
    ImpactForcen*normA;
  if noEvent(Impact) then
    if relvtsqrt <= delta then
      zeros(3) = ImpactForcet +
        f*abs(ImpactForcen)*
        relvt/delta - kappa*normA;
    else
      zeros(3) = ImpactForcet +
        f*abs(ImpactForcen)*
        relvt/relvtsqrt - kappa*normA;
    end if;
  else
    zeros(3) = DeltavrAt + vrAt -
      vrBt - DeltavrBt - kappa*normA;
  end if;
  der(Active) = 0;
end UnilateralConstraintAcausal;
```

State of the constraint is tracked here by the variable `State`. If `State = 0` then the constraint is disconnected. For `State = 1` the constraint is in the state “Sliding”. And for `State = 2` corresponding state is “Rolling”. In a current version of the MBSUC model we suppose that at each instant of time it is possible to occur not more than one impact.

Modelica code presented above has the following features:

1. Impact signal is generated if and only if: the con-

straint be in the state “Flight”, $State = 0$; outer surfaces of the bodies arrive to the contact, $\mu < 0$; and a special impact mask is open. This latter becomes closed for the only case of the smooth launching from the constraint. The variable μ is defined according to the differential version of the system (2.1) such that for $\mu > 0$ the constraint is disconnected, and the contact begins as $\mu = 0$.

2. Kinematic formulae and expressions for the impact impulses are implemented.
3. The variable `Active` is applied here to scale the independent variable as it has been done for the `RigidBody` model.
4. The following parameters of problem are applied: k is the coefficient of restitution at impact, f is the friction coefficient, δ is the regularizing parameter for dry friction.

Dots represent the blocks of an equations implementing the functions: (a) impact signal transmission through the constraint, now under the further development; (b) computation of an intermediate variables according to formulae (2.2, 2.3, 2.4, 2.7)

A key role in the whole model plays the following class

model

`UnilateralConstraintAcausalAddOnRegular`

extends `UnilateralConstraintAcausal`;

parameter `SI.Length`
`ClearanceTolerance`;

parameter `SI.DampingCoefficient`
`ConstraintAttraction=1`;

`SI.Force nu`;

`SI.Force Forcen`;

`SI.Force[3] Forcet`;

`SI.Acceleration Drelvn`;

`Real StateIndicator`;

`SI.Length Clearance(start=1)`;

equation

```
der(relvn) = Active*(Drelvn + (if
  StateIndicator > 0.5 then -
  ConstraintAttraction*relvn else
  0));
```

```
Forcen = OutPortA.F*normA;
```

```
Forcet = OutPortA.F - Forcen*normA;
```

```
if StateIndicator <= 0.5 then
```

```
  State = 0;
```

```
  Forcen = 0;
```

```
  Forcet - nu*normA = zeros(3);
```

```
  if mu > 0 and relan < 0 then
```

```
    StateIndicator = 0;
```

```
  else
```

```
    if Clearance < ClearanceTolerance
```

```
  then
```

```
    if relan < 0 then
```

```
      // Case of launch
```

```
      StateIndicator = 0;
```

```
    else
```

```
      // Case of landing
```

```
      if relvtsqrt > delta then
```

```
        StateIndicator = 1;
```

```
      else
```

```
        StateIndicator = 2;
```

```
      end if;
```

```
    end if;
```

```
  else
```

```
    StateIndicator = 0;
```

```
  end if;
```

```
end if;
```

```
else
```

```
Drelvn = 0;
```

```
if relvtsqrt <= delta then
```

```
  State = 2;
```

```
  StateIndicator = if Forcen > 0
```

```
    then 0 else 2;
```

```
  Forcet - f*Forcen*relvt/delta -
  nu*normA = zeros(3);
```

```
else
```

```
  State = 1;
```

```
  StateIndicator = if Forcen > 0
```

```
    then 0 else 1;
```

```
  Forcet - f*Forcen*relvt/relvtsqrt
  - nu*normA = zeros(3);
```

```
end if;
```

```
end if;
```

```
der(Clearance) = 0;
```

```
when Impact then
```

```
  reinit(Clearance, 0.5*abs(relvn*
  relvn/Drelvn));
```

```
end when;
```

```
when StateIndicator > 0.5 then
```

```
  reinit(Clearance, 1);
```

```
end when;
```

```
OutPortA.M = zeros(3);
```

```
OutPortA.ImpactTorque = zeros(3);
```

```
end
```

```
UnilateralConstraintAcausalAddOnRegular;
```

which implements the real switching of the constraint states.

The hybrid automata states are controlled by two variables: `StateIndicator` and `State`. The first one is included into the algebraic loops and has the `Real` type. Hence in some sense the states themselves correspond to fuzzy values and are identified by the inequalities. It is clear that such a situation is connected with the compiler restrictions. The variable `State` doesn't belong to the algebraic loops. It has the `Integer` type and doesn't influence on the switching between

the states.

In frames of the model under consideration in order to estimate the maximal clearance between the bodies during the time from one impact to the next one, the variable `Clearance`, is introduced to detect the instant of the transition to the state of the bodies contact. The complementarity rules are also implemented here.

Remark 3.1 *We have to perform the regularization of the independent variable for the case of landing on the constraint using variables `Active` of the bodies and the constraints objects outside these objects but inside the corresponding container of the whole MBSUC model. In this case we have a possibility for the correct control over the regularization process because the change of the independent variable should be total throughout the MBSUC.*

At last, the models implementing the system of constraint equations complete a chain of inheritance for the constraint classes. Namely two classes

`SurfacesOfConstraintAcausalDifferential`,
`EllipsoidAndHorizontalPlaneDifferential`.

have been constructed. First one doesn't depend upon specific type of the outer surfaces. The second model implements a specific case of the ellipsoidal surface and the plane as a surfaces of constraint.

4 Examples

Experimental computations and verification of the models developed were carried out using a well known example from classical dynamics: motion of heavy body on/over the horizontal surface. Visual image of the MBSUC model is presented in Figure 4.1.

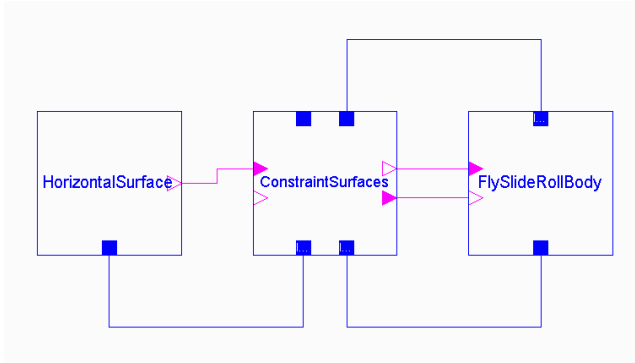


Figure 4.1: Visual Model of MBSUC

The object `HorizontalSurface` on the left hand side of the figure represents model of the base body describing a horizontal plane fixed in AF . The object

of the heavy rigid body is shown on the right hand side of the Figure 4.1. And the model of total MBSUC for our example has the following Modelica code

```

model MBSAcausalDifferential
  ...
  parameter Period TimeScale=1;
  Period deltat(start=1);
  Time t(start=0);
equation
  ...
  der(deltat) = 0;
  der(t) = deltat/TimeScale;
  when ConstraintSurfaces.Impact then
    reinit(deltat, min(1, 2*abs(
      ConstraintSurfaces.relvn/
      ConstraintSurfaces.relan)));
    reinit(FlySlideRollBody.Active,
      min(1, 2*abs(ConstraintSurfaces.
        relvn/ConstraintSurfaces.relan)));
    reinit(ConstraintSurfaces.Active,
      min(1, 2*abs(ConstraintSurfaces.
        relvn/ConstraintSurfaces.relan)));
  end when;
  when ConstraintSurfaces.
    StateIndicator > 0.5 then
    reinit(deltat, 1);
    reinit(FlySlideRollBody.Active, 1);
    reinit(ConstraintSurfaces.Active,
      1);
  end when;
end MBSAcausalDifferential;

```

To estimate an accuracy of the model developed we performed a comparison of the results with ones for the exact model of the hybrid automata built using causal approach with the three instantiated objects each corresponding to one state of the mechanical system and having a structural complexity of the whole MBSUC, see Figure 4.1.

The rigid body already considered in one of the examples of the paper [1] starts its motion from a position suspended over the surface with the initial data

$$\begin{aligned}
 \mathbf{r}(0) &= (0, 5, 0)^T, & \mathbf{v}(0) &= (0.05, 0, 0)^T, \\
 \mathbf{q}(0) &= (1, 0, 0, 0)^T, & \boldsymbol{\omega}(0) &= (0, -10, 2)^T.
 \end{aligned} \quad (4.8)$$

Motion is simulated on time segment $[t_0, t_1] = [0, 150]$ and consists of the several stages of flight alternating by stages of sliding. Note that sliding followed by rolling as energy decreases. During several decades of seconds one can observe easily so called stick-slip phenomena transferring finally to the pure rolling.

The results of simulation are presented in the Figures 4.2, 4.3, 4.4. The final part of the projection of the trajectory for the point P_B of the ellipsoid corresponding to the stick-slip phase is shown in Figure 4.2.

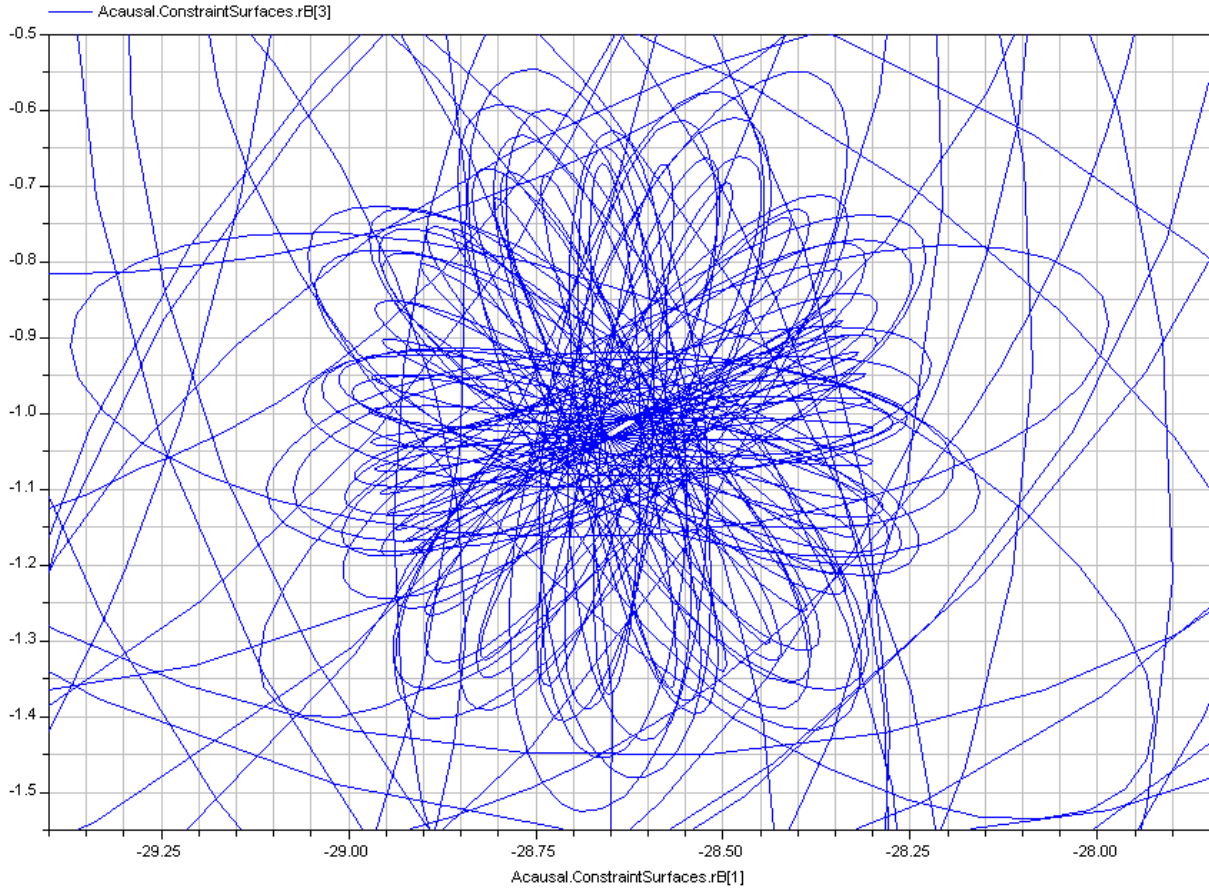


Figure 4.2: Stick-Slip Oscillations, Contact Point Trajectory

An accuracy of the model is of our special interest. The accuracy of computation for instants of impacts and of transitions between the MBSUC states is a causal point for the models of systems with impacts. In our example the instant of the first transfer to rolling at the beginning of stick-slip oscillations has a relative error of the order 10^{-4} . Such an error was accumulated after several thousands of impacts and several transitions between states “Flight” and “Sliding”. More accurate regularization of the independent variable allows to achieve further reduction of the error. Of course it needs considerable computational time in addition. For comparison of physical time variables depending on the regularizing time for the models compared see Figure 4.3. As one can see, physical times almost coincide for the acausal and causal models. In addition, it would be interesting to observe the initial interval of the simulation corresponding to the several stages of a decrementing bouncing of the body, see Figure 4.4. Here we can see the screenshot of the body while it performs one of jumps. The image of the first transfer from the flight mode to the mode of sliding is presented here in details. One can see in this inserted fragment the regularizing independent “time” counted

along x -axis. y -axis represents the variable μ .

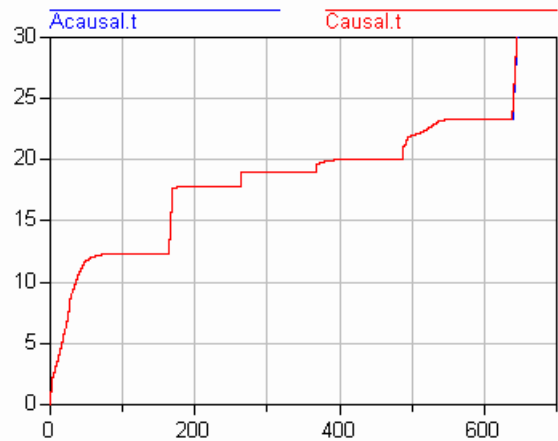


Figure 4.3: Physical Times Depending upon Regularizing Time

In the case of motion with a contact the switching between sliding and rolling is observed. For this case the simulation was performed with the following initial data

$$\begin{aligned}
 \mathbf{r}(0) &= (0, 1, 0)^T, & \mathbf{v}(0) &= (0.05, 0, 0)^T, \\
 \mathbf{q}(0) &= (1, 0, 0, 0)^T, & \boldsymbol{\omega}(0) &= (0, -2, 2)^T.
 \end{aligned}$$

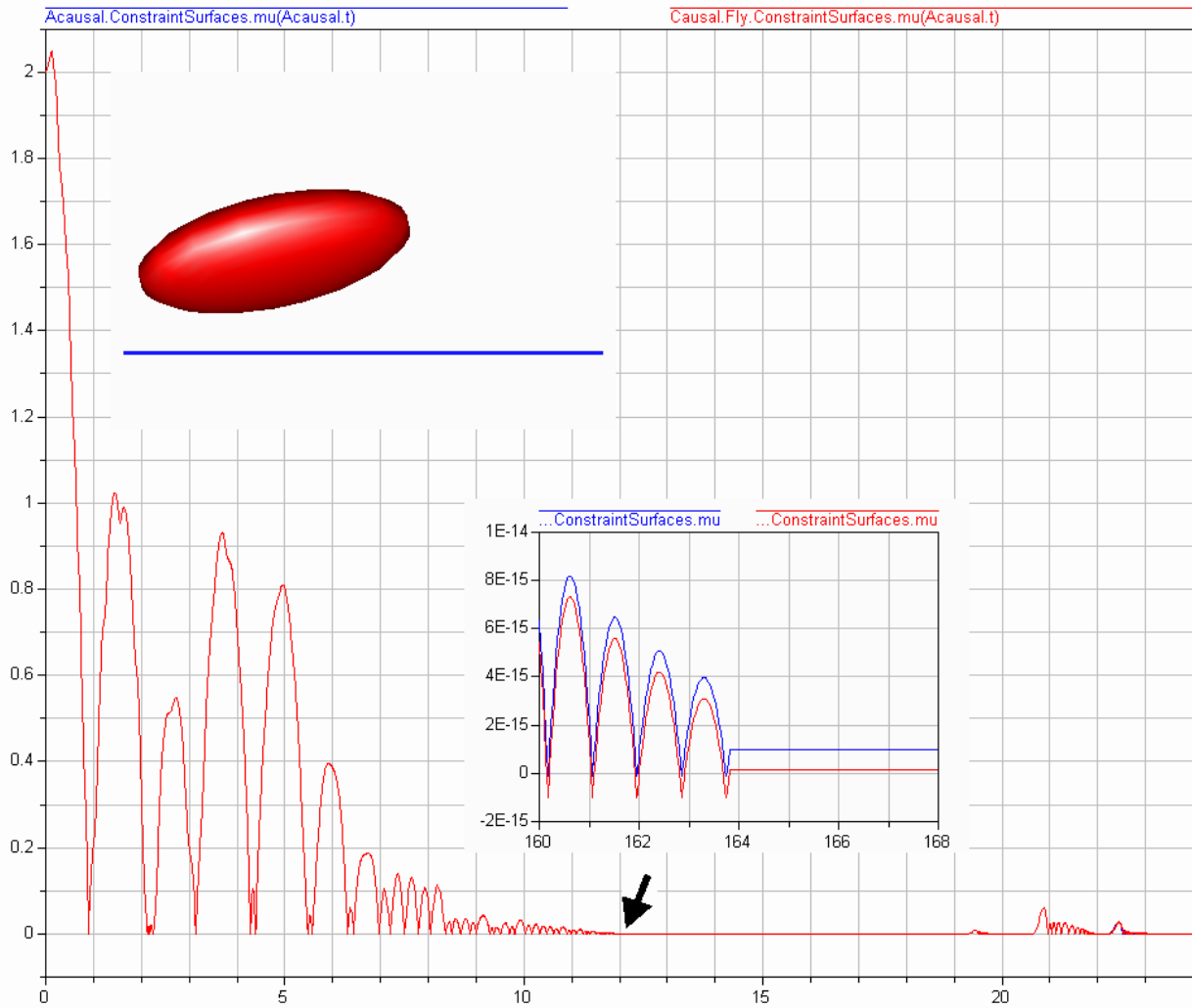


Figure 4.4: Stages of Bouncing

During the time of $t_1 - t_0 = 150$ units and after several hundreds of stick–slip oscillations the relative error accumulated for state switch instants was equal to 10^{-11} . Thus the absence of impacts during the simulation improves the quality of the model more than in million times.

For the sliding/rolling mode the absolute error of determination of the contact point does not exceed $3 \cdot 10^{-5}$. It was observed that the error grows almost linearly. The error in determination of the position of the point P_B in the normal direction is equal to $2 \cdot 10^{-15}$, while for rolling the error of determination of the tangent component of velocity of this point does not exceed 10^{-7} .

Let us consider now the motion of the homogeneous body bounded by an ellipsoidal surface on the horizontal plane [7]. The coefficient of the Coulomb friction supposed to be equal to $d = 0.01$. Let us try to repeat numerically the following experiment described qualitatively by A. P. Markeev. *The body touches the hor-*

izontal surface by its shortest semi-axis at the initial instant. Let us put it in rapid rotation. Then the body tends to the position in which it touches the plane by its longest semi-axis.

In our example the semiaxes of the body are close one to another: $a_1 = 1.2$, $b_1 = 1$, $c_1 = 1.3$. Axes of outer surface ellipsoid coincide with ones of central principal ellipsoid. Choosing the initial data as in (4.8) with one exception: $\mathbf{r}(0) = (0, 1, 0)^T$ one obtains the result cited above: the ellipsoid masscenter “rises” progressively from the height of minimal semi-diameter to one of maximal semi-diameter, see Figure 4.5. The angular velocity almost holds its direction with respect to the AF , see Figure 4.6, blue (lower) curve. At the initial instant this vector is directed along the minimal semi-axis, red (middle) curve is its projection on the corresponding axis of the body; while on the final stage the angular velocity is directed along the maximal ellipsoid semi-axis, green (upper) curve.

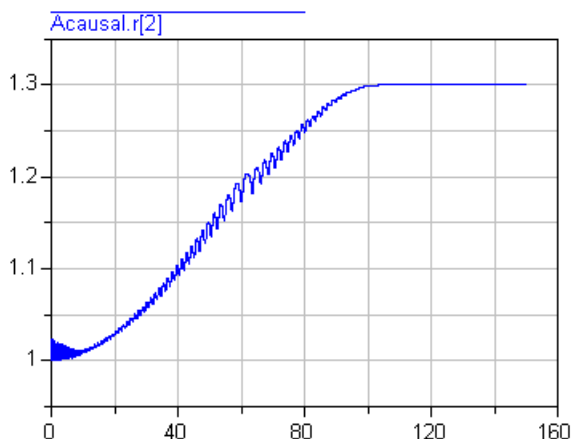


Figure 4.5: Center of Mass Altitude

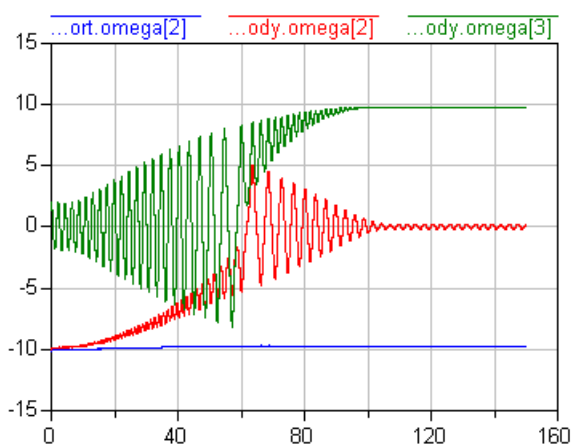


Figure 4.6: Projections of Angular Velocity

5 Conclusions

Summarizing the results obtained while developing the class library for the dynamics of the MBSUC let us enumerate several relevant problems and their solutions.

Problem 1: How one can implement the geometry of the unilateral constraint? **Solution:** Use the system of algebraic equations like (2.1).

Problem 2: How one can ensure the reliability of the implementation of the constraint? **Solution:** Use the differential form of the equations (2.1).

Problem 3: How one can implement impacts in MBSUC in the acausal manner? **Solution:** Use the independent algebraic subsystem of equations distributed throughout the MBSUC model.

Problem 4: How one can implement the dichotomy flight/contact? **Solution:** Use the complementarity rule for the normal force of reaction and the derivative of the normal relative velocity at the contact point.

Problem 5: How one can implement the dichotomy slipping/rolling? **Solution:** Use the regularized tangent force for the Coulomb friction.

Problem 6: How one can implement the exact “landing” on the constraint? **Solution:** Use the regularizing independent variable for the total model.

Problem 7: How one can implement switching between states of the constraint in the acausal manner? **Solution:** Use the **if** clause in combination with the state variable of `Real` type. This variable is included to corresponding algebraic loop. As a result the structural complexity of the total model doesn't increase.

6 Acknowledgement

The paper was prepared with partial support of Russian Foundation for Basic Research, grants 02-01-00196, SS-2000.2003.1.

References

- [1] Kossenko, I. I., and Stavrovskaja, M. S., How One Can Simulate Dynamics of Rolling Bodies Via Dymola: Approach to Model Multibody System Dynamics Using Modelica // Proceedings of the 3rd International Modelica Conference, Linköpings universitet, Linköping, Sweden, November 3–4, 2003, pp. 299–309.
- [2] Pfeiffer, F., Unilateral Multibody Dynamics // *Meccanica*, 1999, Vol. 34, No. 6, pp. 437–451.
- [3] Novozhilov, I. V., Fractional Analysis : Methods of Motion Decomposition. — Boston: Birkhauser, 1997.
- [4] Routh, E. J., A Treatise on the Dynamics of a System of Rigid Bodies. — London: Vol. 1, 1897.
- [5] Ivanov, A. P., Dynamics of Systems with Mechanical Impacts. — Moscow: 1997. ISBN 5-7781-0031-0.
- [6] Dymola. Dynamic Modeling Laboratory. User's Manual. Version 5.1b — Lund: Dynasim AB, Research Park Ideon, 2003.
- [7] Markeev, A. P., On the Motion of an Ellipsoid on a Rough Surface with Slippage. // *Journal of Applied Mathematics and Mechanics*, Vol. 47, Iss. 2, 1983, pp. 260–268.

Object-Oriented Modelling and Simulation of Flexible Multibody Thin Beams in Modelica with the Finite Element Method

Gianni Ferretti, Francesco Schiavo*, Luca Viganò
Politecnico di Milano,
Dipartimento Di Elettronica Ed Informazione (DEI)
Via Ponzio 34/5, 20133 Milano, Italy
e-mail: {ferretti, schiavo, viganò}@elet.polimi.it

Abstract

In this paper the development, simulation and validation of *Modelica* models for flexible thin beams is presented.

The models are based on the application of the finite element method. Exploiting the object-oriented features of the language, mixed-mode models (finite element-finite volume) are developed as well.

All the models use the standard connectors defined within the *Modelica* multibody library, guaranteeing thus full compatibility with the library components.

The details of the mathematical modelling are fully analyzed, showing the development of the equations of motion.

The models feature also a graphical interface, with visualization of the simulation outcomes within the same 3D environment used in the multibody library, allowing the user to have an immediate visual feedback.

Finally, the models are analyzed and validated by mean of selected simulation experiments, with reference both to theoretical predictions and to results commonly accepted within the scientific literature.

1 Introduction

Many engineering applications require the development of simulation models for flexible multibody systems (e.g., robot manipulators, helicopter rotors, aircraft wings, space structures, machining tools, car suspensions, etc.) both dynamically accurate and computationally affordable.

The task of developing models for generic-shaped, fully deformable bodies is usually demanded to specialized simulation codes and tools, due to the complexity of the task. Such models are usually adequate

for structural analysis and design tasks, while being far too complex for affordable dynamics simulation and analysis.

On the other hand, particular classes of deformable bodies, such as flexible beams, can be represented with less complex models which are still able to represent all the dynamically relevant deformation effects.

Flexible beams are continuous non linear dynamical systems characterized by an infinite number of degrees of freedom. Obviously, dealing directly with infinite dimensional models is impractical both for dynamic analysis and simulation purposes. Hence it is necessary to introduce methods to describe flexibility with a discrete number of parameters.

Three different approaches have been traditionally used to derive approximated finite dimensional models: lumped parameters, assumed modes and finite element method [3],[5].

The lumped parameter approach is the simplest one. In this method each flexible beam is divided into a finite number of rigid beams, introducing pseudojoints, and the flexibility is represented by springs that restrict the motion of each pseudojoint. This method is however rarely used because of the difficulty in determining the spring constants of the pseudojoints and then of achieving a suitable accuracy up to the desired approximation frequency.

The assumed modes model formulation has been widely used in the literature [6]. It describes beam flexibility using truncated modal series, based on spatial mode eigenfunctions and time varying vibrational modes. One of the best features offered by such a method is the fine control on the accuracy up to the desired approximation frequency. Although conceptually simple, this description requires to find out the best selection for spatial modal shapes and the boundary conditions, which is not at all a trivial task. In addition to that, the selection of the appropriate eigenfunc-

*corresponding author

tions and the resulting vibrational modes could depend on the boundary conditions for the specific case at hand, ruling thus out the possibility of a modular approach for the model development.

In the finite element method approach [9], the flexible beam is divided into several elements, with a local description of the deformation field by the use of element-wise basis functions. Although such approach could be computationally more demanding than the modal one (it is usually necessary to use a larger number of elements than of modal eigenfunctions to obtain the same approximation), it allows a formulation which is independent of the actual boundary condition [7]. The finite element method is then a viable choice for the representation of flexible beams within a modular environment.

As far as the theory of elasticity to be used is concerned, it must be pointed out that beam deflection, with respect to the rigid configuration, is generally assumed to be small, which allows to adopt linear theory. In this case the Euler-Bernoulli theory [8] can be used to describe beam flexibility, neglecting the effects of shear deformation and assuming uniform cross-sectional properties along the beam. In this paper, we consider linear elasticity theory for the modelling of flexible *thin* beams. On the other side, Timoshenko theory [8] should be used for models where such effects need to be taken into account (e.g., for short beams).

The paper is organized as follows: in Section 2 the problem of the representation of a generic deformable body in a multibody system is introduced; in Section 3 the development of the equations of motions is shown, with reference both to the finite element method case and to the mixed-method one; in Section 4 the *Modelica* implementation is analyzed; Section 5 contains selected simulation results; finally, in Section 6 the main results are summarized and future developments are introduced.

2 Deformable Body Degrees of Freedom

Consider a generic multibody system (Fig. 1). The position, in body coordinates, of a point on a specific deformable body has the following expression:

$$\bar{u} = \bar{u}_0 + \bar{u}_f, \quad (1)$$

where \bar{u}_0 is the “undeformed” (i.e., rigid) position vector and \bar{u}_f is the deformation contribution to position (i.e., the deformation field).

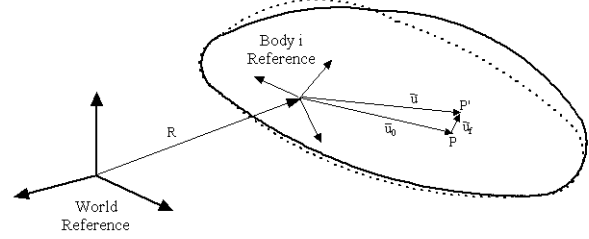


Figure 1: Flexible body reference systems

The formal and mathematically sound description of the generic deformation of a body requires the deformation field to belong to an infinite dimensional functional space, requiring, in turn, an infinite number of deformation degrees of freedom.

In this paper, the deformation field is described by an approximation of the functional basis space it belongs to, supposing such space has a finite dimension, say M , so that the vector u_f can be expressed by the following finite dimensional product:

$$\bar{u}_f = S q_f, \quad (2)$$

where S is the $[3 \times M]$ shape functions matrix (i.e., a matrix of functions defined over the body domain and used as a basis to describe the deformation field of the body itself) and q_f is the M -dimensional vector of deformation degrees of freedom.

The position of a point on a deformable body can then be expressed in world reference as follows:

$$r = R + A\bar{u} = R + A(\bar{u}_0 + S q_f) = R + A\bar{u}_0 + A S q_f, \quad (3)$$

where R is the vector identifying the origin of the body local reference system and A is the rotation matrix for the body reference system.

The representation of a generic deformable body in world reference requires then $6 + M$ d.o.f. (i.e., 6 corresponding to rigid displacements and rotations and M to deformation fields):

$$q = [q_r \ q_f]^T = [R \ \theta \ q_f]^T, \quad (4)$$

where θ represents the undeformed body orientation angles and q_r is a vector containing the 6 rigid degrees of freedom.

3 Motion Equations

The equations of motion for a generic flexible body in a multibody system can be developed applying the principle of virtual work [3]. It should be pointed out that the same results could be obtained using the classical Lagrangian approach (as in, e.g., [5]), though

such approach is quite knotty and difficult to use in practice, due to the complexity of the required analytical differentiation of the kinetic energy expression.

The principle of virtual work states that the virtual work of the inertial forces δW_i must counterbalance the sum of the virtual work of the *continuum* elastic forces δW_s and of the external ones δW_e :

$$\delta W_i = \delta W_s + \delta W_e. \quad (5)$$

Note that, in case $\delta W_i = 0$, the problem reduces to the well-known problem of structural statics [9].

The terms of equation (5) are defined as follows:

$$\delta W_i = \int_V \rho \delta r^T \ddot{r} dV, \quad (6)$$

$$\delta W_s = - \int_V \delta \varepsilon^T \sigma dV, \quad (7)$$

$$\delta W_e = \int_V \delta r^T F_e dV + \int_{\Omega} \delta r^T f_e d\Omega, \quad (8)$$

where V is the body volume, ρ is the body density, δr is an infinitesimal virtual displacement, \ddot{r} is the body acceleration (in world reference), $\delta \varepsilon$ is a vector of virtual infinitesimal internal strains, σ is the internal stresses vector, F_e is the vector of external volume forces, Ω is the body surface and f_e is the vector of external surface forces.

The quantities δr and \ddot{r} can be computed using equation (3):

$$\begin{aligned} \delta r &= \delta R + \delta A \bar{u} + A \delta \bar{u} = \delta R + \theta_d \times A \bar{u} + A S \delta q_f, \\ \ddot{r} &= \ddot{R} + \omega \times \omega \times u + \alpha \times u + 2\omega \times A \dot{\bar{u}} + A \ddot{\bar{u}}, \end{aligned} \quad (9)$$

where α and ω are the body angular acceleration and velocity (in world reference), respectively, and $\theta_d = \omega dt$ represents a virtual-infinitesimal rotation.

The expressions in (9) can be substituted in 5, leading to

$$\delta W_i = \int_V \rho \delta r^T \ddot{r} dV = \delta R^T Q_i^R + \theta_d^T Q_i^\theta + \delta q_f^T Q_i^f. \quad (10)$$

The terms Q_i^R, Q_i^θ and Q_i^f can be calculated using the

following definitions:

$$m_{RR} = \int_V \rho dV, \quad (11)$$

$$m_{R\theta} = \int_V \rho A (\bar{u} \times)^T A^T dV, \quad (12)$$

$$m_{Rf} = \int_V \rho A S dV, \quad (13)$$

$$m_{\theta\theta} = - \int_V \rho A \bar{u} \times \bar{u} \times A^T dV, \quad (14)$$

$$m_{\theta f} = \int_V \rho A \bar{u} \times S dV, \quad (15)$$

$$m_{ff} = \int_V \rho S^T S dV, \quad (16)$$

$$\bar{S} = \int_V \rho S dV = A^T m_{Rf}, \quad (17)$$

$$\bar{S}_t = \int_V \rho \bar{u} dV, \quad (18)$$

$$\tilde{\bar{S}}_t = \int_V \rho (\bar{u} \times) dV = A m_{R\theta} A^T, \quad (19)$$

$$\bar{I}_{\theta\theta} = \int_V \rho (\bar{u} \times)^T (\bar{u} \times) dV = A^T m_{\theta\theta} A, \quad (20)$$

$$\bar{I}_{\theta f} = \int_V \rho (\bar{u} \times) S dV = A^T m_{\theta f}. \quad (21)$$

The vector Q_i^R can then be obtained as follows:

$$\begin{aligned} Q_i^R &= \int_V \rho \ddot{R} dV + \int_V \rho \omega \times (\omega \times u) dV + \int_V \rho (\alpha \times u) dV \\ &+ \int_V \rho 2\omega \times (A \dot{\bar{u}}) dV + \int_V \rho A \ddot{\bar{u}} dV = \\ &= m_{RR} \ddot{R} + A \bar{\omega} \times \bar{\omega} \times A^T \int_V \rho A \bar{u} dV + \\ &+ A \bar{\alpha} \times A^T \int_V \rho A \bar{u} dV + 2A \bar{\omega} \times A^T \int_V \rho A S dV \dot{q}_f \\ &+ \int_V \rho A S dV \ddot{q}_f = \\ &= m_{RR} \ddot{R} + A \tilde{\bar{S}}_t^T \bar{\alpha} + A \bar{S} \dot{q}_f + A (\bar{\omega} \times \bar{\omega} \times \bar{S}_t + 2\bar{\omega} \times \bar{S} \dot{q}_f) = \\ &= m_{RR} \ddot{R} + m_{R\theta} \alpha + m_{Rf} \ddot{q}_f - A Q_i^R, \end{aligned} \quad (22)$$

being $Q_i^R = -\bar{\omega} \times \bar{\omega} \times \bar{S}_t - 2\bar{\omega} \times \bar{S} \dot{q}_f$ the quadratic velocity vector (due to Coriolis and centrifugal forces) associated to translational degrees of freedom.

The second term of the generalized inertial forces can

be expressed as

$$\begin{aligned}
 Q_i^\theta &= A \int_V \rho (\bar{u} \times) dV A^T \ddot{R} - \omega \times \int_V \rho u \times u \times dV \omega \\
 &\quad - \int_V \rho u \times u \times dV \alpha + 2 \int_V \rho u \times \omega \times (AS\dot{q}_f) dV \\
 &\quad + \int_V \rho u \times (AS\dot{q}_f) dV = \\
 &= A \tilde{S}_t A^T \ddot{R} + A \bar{\omega} \times \int_V -\rho \bar{u} \times \bar{u} \times dV \bar{\omega} \\
 &\quad - A \int_V \rho \bar{u} \times \bar{u} \times dV \bar{\alpha} - 2A \int_V \rho \bar{u} \times (S\dot{q}_f) \times \bar{\omega} dV \\
 &\quad + A \int_V \rho \bar{u} \times S dV \ddot{q}_f = \\
 &= A \left(\tilde{S}_t A^T \ddot{R} + \bar{\omega} \times \bar{I}_{\theta\theta} \bar{\omega} + \bar{I}_{\theta\theta} \bar{\alpha} + \dot{\bar{I}}_{\theta\theta} \bar{\omega} + \right. \\
 &\quad \left. + \bar{\omega} \times \bar{I}_{\theta f} \dot{q}_f + \bar{I}_{\theta f} \ddot{q}_f \right) = \\
 &= m_{RR}^T \ddot{R} + m_{\theta\theta} \alpha + m_{\theta f} \ddot{q}_f - A Q_v^\theta,
 \end{aligned} \tag{23}$$

where the quadratic velocity vector associated to the rotational degrees of freedom is $Q_v^\theta = -\bar{\omega} \times \bar{I}_{\theta\theta} \bar{\omega} - \dot{\bar{I}}_{\theta\theta} \bar{\omega} - \bar{\omega} \times \bar{I}_{\theta f} \dot{q}_f$.

The Q_i^f term, which is related to the deformation d.o.f. q_f , can be expanded as follows:

$$\begin{aligned}
 Q_i^f &= \int_V \rho S^T A^T \ddot{R} dV + \int_V \rho S^T A^T \omega \times (\omega \times u) dV \\
 &\quad + \int_V \rho S^T A^T (\alpha \times u) dV + \int_V \rho S^T A^T 2\omega \times (A\bar{u}) dV \\
 &\quad + \int_V \rho S^T \ddot{u} dV = \bar{S}^T A^T \ddot{R} + \int_V \rho S^T \bar{\alpha} \times \bar{u} dV + \\
 &\quad + \int_V \rho S^T (\bar{\omega} \times \bar{\omega} \times \bar{u} + 2\bar{\omega} \times S\dot{q}_f) dV \\
 &\quad + \int_V \rho S^T S dV \ddot{q}_f = \\
 &= \bar{S}^T A^T \ddot{R} + \bar{I}_{\theta f}^T \bar{\alpha} + m_{ff} \ddot{q}_f + \\
 &\quad + \int_V \rho S^T (\bar{\omega}^2 \bar{u} + 2\bar{\omega} S\dot{q}_f) dV = \\
 &= m_{Rf}^T \ddot{R} + m_{\theta f}^T \alpha + m_{ff} \ddot{q}_f - Q_v^f,
 \end{aligned} \tag{24}$$

being $Q_v^f = -\int_V \rho S^T (\bar{\omega}^2 \bar{u} + 2\bar{\omega} S\dot{q}_f) dV$.

The virtual work of the internal elastic forces, under the hypothesis of elastic constitutive law for the material, can be expressed as:

$$\delta W_s = - \int_V \delta \epsilon^T \sigma dV = -\delta q_f^T K_{ff} q_f, \tag{25}$$

where K_{ff} represents the structural stiffness matrix. The form of such matrix depends on the specific material constitutive law and on the body shape.

The virtual work of external forces reads as follows:

$$\delta W_e = \delta R^T Q_e^R + \theta_a^T Q_e^\theta + \delta q_f^T Q_e^f, \tag{26}$$

where Q_e^R , Q_e^θ and Q_e^f represent, respectively, the generalized components of the active forces associated to translational, rotational and deformation coordinates.

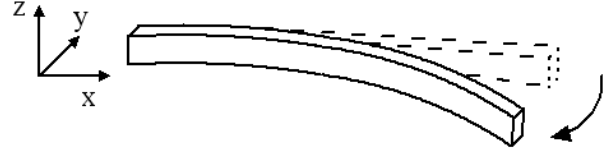


Figure 2: Planar beam deformation

Equation (5) must be satisfied for every virtual displacement so that the following identities must hold:

$$Q_i^R = Q_e^R, \tag{27}$$

$$Q_i^\theta = Q_e^\theta, \tag{28}$$

$$Q_i^f = -K_{ff} q_f + Q_e^f. \tag{29}$$

Equations (27), (28) and (29) are the equations for 3D motion of a generic flexible body characterized by an elastic constitutive law for its material. In the scientific literature, such expressions are generally referred to as the *generalized Newton-Euler* equations (see e.g., [5]). The equations of motion can be easily expressed in body axes, resulting in:

$$\begin{aligned}
 &\begin{bmatrix} m_{RR} & \tilde{S}_t^T & \bar{S} \\ & \bar{I}_{\theta\theta} & \bar{I}_{\theta f} \\ & & m_{ff} \end{bmatrix} \begin{bmatrix} \ddot{R} \\ \bar{\alpha} \\ \ddot{q}_f \end{bmatrix} = \\
 &= \begin{bmatrix} O_3 \\ O_3 \\ -K_{ff} q_f \end{bmatrix} + \begin{bmatrix} Q_v^R \\ Q_v^\theta \\ Q_v^f \end{bmatrix} + \begin{bmatrix} Q_e^R \\ Q_e^\theta \\ Q_e^f \end{bmatrix}.
 \end{aligned} \tag{30}$$

Equations (30) are valid for a general deformable body, though many of the quantities involved (e.g., the matrix K_{ff}) depend on specific body characteristics such as the shape or the material properties.

From now on, the case of a *thin beam* will be considered. In detail, it will be assumed that the body is a 1D elastic *continuum* with constant cross-sectional properties. Furthermore, it will be assumed that the beam constitutive material is homogeneous, isotropic and perfectly elastic (i.e., the elastic internal forces are conservative). Finally, it will be assumed that the deformation field is restricted to lie within the xy plane of the beam local reference system (Fig. 2).

It should be pointed out that such assumptions do not restrict the model validity or generality, since the model remains still representative for a large number of dynamic simulation applications (e.g., almost all the flexible robots commonly studied have flexible links which can be represented by such model [7]).

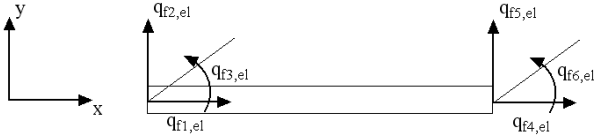


Figure 3: Element coordinate systems

3.1 The element point of view

The finite element method is based upon a discretization of the beam into N elements. A single element can itself be viewed as a thin beam characterized by a planar deformation field. It is then possible to define the local dimensionless *abscissa* as $\xi = x/\ell$, where x is the longitudinal local coordinate and ℓ is the element length.

In [9] it is shown that the partial differential equations associated with the deformation problem at hand, under the hypothesis of elastic constitutive law for the material, require, for a consistent finite element formulation, the use of linear and Hermite cubic polynomials for the approximation of the axial and transversal deformation field, respectively. Thus, for a single element, the generic equations of motion (30) can be expanded as follows:

$$\begin{aligned} \bar{u}_{f,el} &= \begin{bmatrix} \bar{u}_{f1,el} \\ \bar{u}_{f2,el} \\ \bar{u}_{f3,el} \end{bmatrix} = S_{el} q_{f,el}, \\ S_{el} &= \begin{bmatrix} 1-\xi & 0 & 0 & \dots \\ 0 & 1-3\xi^2+2\xi^3 & \ell(\xi-2\xi^2+\xi^3) & \dots \\ 0 & 0 & 0 & \dots \\ \xi & 0 & 0 & \dots \\ \dots & 0 & 3\xi^2-2\xi^3 & \ell(\xi^3-\xi^2) \\ 0 & 0 & 0 & \dots \end{bmatrix} = \begin{bmatrix} S_{el1} \\ S_{el2} \\ S_{el3} \end{bmatrix}, \\ q_{f,el} &= [q_{f1,el} \ q_{f2,el} \ q_{f3,el} \ q_{f4,el} \ q_{f5,el} \ q_{f6,el}]^T, \end{aligned} \quad (31)$$

where the subscript el is used to refer the quantities to a single element.

Fig. 3 depicts the element coordinate systems associated with the deformation degrees of freedom: $q_{f1,el}$ and $q_{f4,el}$ are associated with axial compression, $q_{f2,el}$ and $q_{f5,el}$ with transversal displacement and $q_{f3,el}$ and $q_{f6,el}$ with beam extremities rotation.

Since the third row of the shape matrix S_{el} is composed only by zeros, it could be noted that, despite the fact that the motion equations have been developed for a general 3D case, the deformation field is assumed to lie within the local xy plane.

The planar deformation hypothesis and the assumption of a homogeneous, isotropic and elastic material for the beam, allow to exploit the Euler-Bernoulli theory and to calculate the elastic potential energy U_{el} , neglecting the contribution of shear stresses and considering only the work of the resulting axial force N_{el} and

bending moment M_{el} , as follows [9]:

$$\begin{aligned} U_{el} &= \frac{1}{2} \int_{\ell} \left(\frac{N_{el} N'_{el}}{EA} + \frac{M_{el} M'_{el}}{EJ} \right) dx = \\ &= \frac{1}{2} \int_{\ell} \left(EJ \bar{u}_{f2,el}''^2 + EA \bar{u}_{f1,el}'^2 \right) dx = \frac{1}{2} q_{f,el}^T K_{ff,el} q_{f,el}, \end{aligned} \quad (32)$$

where E is the material Young's modulus, A is the (constant) cross-sectional area and J is the (constant) cross-sectional second moment of area. The analytical expression for the case at hand for the matrix $K_{ff,el}$, usually known as the structural stiffness matrix, is reported in appendix A.

3.2 Finite Element Method Equations Assembly

The equations of motion for the entire beam can be obtained by assembling the equations of motion for beam elements as the one defined in the previous subsection. The body reference system will be the local reference system located at the root of the first element, so that the rigid degrees of freedom, common to all the elements, will be referred to such coordinate system.

Let then m and L be the mass and length of the entire beam, and N the number of elements to be used, so that $\ell = L/N$. Indicating with \widehat{X} the reference system unit vector along the beam axis, the expression of the generic position \bar{u}_j of a point of element j can be expressed as:

$$\bar{u}_j = \bar{u}_{0j} + S_{el} B_j q_f = [\xi_j \ell + (j-1)\ell] \widehat{X} + S_{el} B_j q_f, \quad (33)$$

where \bar{u}_{0j} is the position of the root of the j^{th} element, S_{el} is the shape functions matrix defined by (31), B_j is the so-called *connectivity matrix* and q_f is a vector containing the deformation degrees of freedom for the whole beam.

The matrices B_j have the following form:

$$B_j = [O_{6,3(j-1)} \mid I_6 \mid O_{6,3(N-j)}], \forall j = 1, \dots, N. \quad (34)$$

The connectivity matrices are used to relate the vector q_f , which contains the deformation degrees of freedom for the whole beam, to the corresponding j^{th} element, according to the expression:

$$q_{f,el_j} = B_j q_f. \quad (35)$$

The dynamics of the complete flexible beam can then be described by equation (30), using the following ex-

pressions:

$$\begin{aligned}
 \bar{S} &= \sum_{j=1}^N \frac{m}{L} \int_{V_j} S_{el} B_j dV_j, \\
 \bar{S}_t &= \sum_{j=1}^N \frac{m}{L} \int_{V_j} \bar{u}_j dV_j, \\
 \bar{I}_{\theta\theta} &= \sum_{j=1}^N \frac{m}{L} \int_{V_j} \begin{pmatrix} \bar{u}_{2fj}^2 & -\bar{u}_{2fj}\bar{u}_{1j} & 0 \\ & \bar{u}_1^2 & 0 \\ & & \bar{u}_{1j}^2 + \bar{u}_{2fj}^2 \end{pmatrix} dV_j, \\
 \bar{I}_{\theta f} &= \sum_{j=1}^N \frac{m}{L} \int_{V_j} \begin{pmatrix} O_{(3N,1)} \\ O_{(3N,1)} \\ \bar{u}_{1j} S_{el2} - \bar{u}_{2j} S_{el2} \end{pmatrix} dV_j, \\
 m_{ff} &= \sum_{j=1}^N \frac{m}{L} B_j^T \left(\int_{V_j} S_{el}^T S_{el} dV_j \right) B_j, \\
 K_{ff} &= \sum_{j=1}^N B_j^T K_{ff,el} B_j, \\
 Q_v^f &= - \sum_{j=1}^N \frac{m}{L} \int_{V_j} \left[B_j^T S_{el}^T \left(\tilde{\omega}^2 \bar{u}_j + 2\tilde{\omega} S_{el} B_j \dot{q}_f \right) \right] dV_j.
 \end{aligned} \tag{36}$$

The computation of the above terms can be easily carried out by observing that the integral of a generic quantity \mathcal{F} , varying along the beam, onto the volume of a single element can be computed as follows:

$$\int_{V_j} \rho \mathcal{F} dV_j = \frac{m}{L} \int_0^1 \ell \mathcal{F}(\xi) d\xi = \frac{m}{N} \int_0^1 \mathcal{F}(\xi) d\xi. \tag{37}$$

3.3 Boundary Conditions

The equations of motion for the whole beam must be completed by enforcing suitable boundary conditions for the finite element approximation of the deformation partial differential equations. That means assuming prescribed values for some of the deformation displacements, rotations and velocities (linear or angular) at the body boundaries which are, for the case at hand, the beam root and tip.

The most commonly used boundary conditions for flexible beams are of two kinds, commonly referred to as *clamped-free* and *simply-supported* conditions. In both cases six conditions are given (as it is required from the underlying partial differential equations): the *clamped-free* ones enforce null deformation at the beam root (i.e., q_{f1} , q_{f2} , q_{f3} , \dot{q}_{f1} , \dot{q}_{f2} , \dot{q}_{f3} equal to zero for the first element), while the *simply-supported* ones enforce null axial and transversal displacement at the beam root (i.e., q_{f1} , q_{f2} , \dot{q}_{f1} , \dot{q}_{f2} equal to zero for the first element) and transversal displacement at the beam tip (i.e., q_{f5} and \dot{q}_{f5} equal to zero for the last element).

The choice of which of the two set of conditions has to be used largely depends on the problem at hand.

It should be pointed out that the boundary conditions names are just conventional and are not referred to the objects the beam is connected or linked to (e.g., joints or other bodies), so that enforcing such boundary condition does not limitate in any way the generality and modularity of the model developed so far.

The enforcement of the boundary conditions is traditionally obtained by introducing suitable matrices in equations (30) [6, 9]. On the other hand, it can be observed that such conditions can be enforced by suitable modifications of the connectivity matrices B_1 and B_N , by zeroing some entries. For example, for the *clamped-free* conditions, B_N remains unvaried and B_1 becomes

$$B_1 = \left[\begin{array}{c|c} 0_3 & 0_3 \\ \hline 0_3 & I_3 \end{array} \middle| O_{6,3(N-1)} \right]. \tag{38}$$

3.4 Extended Formulation of the Equation of Motion

In the finite element formulation for the equation of motion for a flexible beam, the reference directions of the internal actions are the same for all the elements. Such representation is acceptable as long as the deformation field is small compared to the beam length, as it is the case, for example, when studying the dynamics of vibrations in machining tools.

On the other hand, when large deformations are involved, the internal actions reference directions should change according to the deformation field. That means that it is necessary to define a local reference system for each element (Fig. 4). This corresponds to the application of the finite volume method to assemble the equations of motion solved over each element (i.e., over each volume). This representation is valid also for large beam deformation, as long as the deformation field is small compared to the volumes length.

Furthermore, it is possible to assemble the equation of motion for a mixed (finite element-finite volume) formulation by dividing every volume into several elements.

It is not necessary to go into the detailed calculations for the finite volume or the mixed formulation since, as it will be shown in section 4, the equations of motion for such extensions can be automatically calculated with the aid of symbolic manipulation algorithms applied to the finite element formulation.

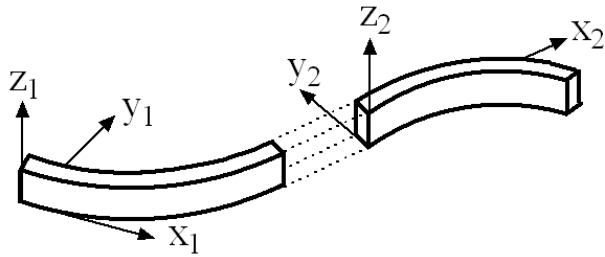


Figure 4: Volume coordinate systems

4 Modelica Implementation

The finite element formulation for the model has been implemented using the *Modelica* language, creating thus a new component, called *FlexBeamFEM* (Fig. 5). The component interfaces are two standard mechanical flanges from the new *MultiBody* library [4]. The connectors choice makes the component fully compatible with the library, so that it is possible to connect directly the flexible beam component with the predefined models such as mechanical constraints (revolute joints, prismatic joints, etc.), parts (3D rigid bodies) and forces elements (springs, dampers, forces, torques).



Figure 5: Component icon

In detail, the flexible beam component uses two mechanical flanges as physical representation of the two ends of the beam while the motion is ruled by equations (30), with addition of a damping term ($-D_{ff}\dot{q}_f$) for the structural dynamics part. The damping term is added to model the dissipative properties of the material.

The terms Q_e^R, Q_e^θ, Q_e^f (i.e., the external actions) are computed on the basis of the forces and torques exchanged at the two connectors with the following code:

```
QeR=matrix(fa+fb_a);
QeTheta=matrix(ta+tb_a+cross([L,0,0]
+S1*B[N, :, :] * qf), fb_a);
Qef=transpose((transpose(matrix(fb_a))*S1*
B[N, :, :] + transpose(matrix(ta))*dS0*B[1, :, :]
+transpose(matrix(tb_a))*dS1*B[N, :, :]));
```

where f_a and f_{b_a} are the forces at the connectors, t_a and t_{b_a} the moments at the connectors, $S1$ the matrix S_{el} evaluated for $\xi = 1$, $B[N, :, :]$ the connectivity matrix B_N and $dS0$ and $dS1$ are matrices used

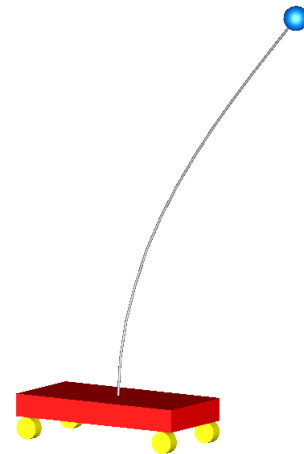


Figure 6: Cart with flexible inverted pendulum

to select the flanges moments acting on the deformation field; forces and moments are referred to the root flange coordinate system.

The model parameters include the beam length and cross sectional area, the material density and Young modulus, the cross sectional inertia, the damping factor and the number of elements.

Particular care has been put into the realization of a 3D interface for the model to visualize the simulation results (Fig. 6), implemented by exploiting the features of the graphical environment of the multibody library. The 3D visualization has revealed itself to be an important feature, giving significative insight and sensible feedback about the dynamical behaviour of the model.

The finite volume model and the mixed one can be easily obtained by connecting several finite element beams composed by one or more elements, respectively. The achievement of such results, which significantly simplify the models implementation, is based on the modular approach adopted in the finite element model development. The assembly of the equations of motion for these cases is demanded to Modelica-based simulation environments, which usually employ advanced symbolic manipulation techniques and index reduction algorithms.

The dynamical properties of the latter models are significantly complex and accurate, featuring a displacement description which is fully non-linear and allowing the simulation of large displacement due to deformation (Fig. 7) at the cost, though, of a significant increase of the computational complexity with respect to the “pure” finite element model.

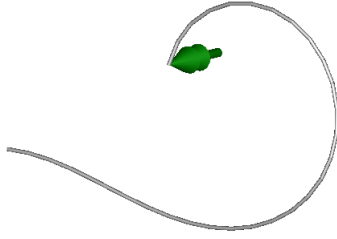


Figure 7: Large deformation of a thin beam

5 Simulations

The different flexible beam models have been validated by several simulation analysis performed within the Dymola simulation environment [1]. The most significant ones are reported in the following subsections.

5.1 Free Vibration

In this simulation the free vibration of a flexible beam is analyzed. The test-case has been set up in order to investigate the models properties with respect to theoretical predictions.

The beam component is connected to the world reference system, so that no rigid motion is allowed; furthermore, no gravity field is considered.

At the initial time instant the beam is standing still with a non-null tip displacement, then it evolves, vibrating, towards steady state.

The vibration frequencies of a flexible beam clamped at the root can be calculated by solving the following partial differential equation:

$$\rho \frac{\partial^2 y(x,t)}{\partial t^2} + EJ \frac{\partial^4 y(x,t)}{\partial x^4} = 0 \quad (39)$$

with the following boundary and initial conditions:

$$\begin{cases} y(0,t), \frac{\partial y}{\partial x}(0,t), \frac{\partial^2 y}{\partial x^2}(0,t), \frac{\partial^3 y}{\partial x^3}(0,t) = 0 \\ y(x,0) = f(x), \frac{\partial y}{\partial t}(x,0) = 0 \end{cases} \quad (40)$$

where x is the axial coordinate, y is the transversal displacement and $f(x)$ is the initial deformation field. In [3] it is shown that the general solution for equation (39) has the following expression:

$$y(x,t) = \sum_{k=1}^{\infty} \varphi_k(x) \alpha_k(t), \quad (41)$$

where $\varphi_k(x)$ are the spatial eigenfunctions and $\alpha_k(t)$ are periodical functions, with natural pulsation (corre-

Mode	Freq.* [Hz]	Freq.† [Hz]	Error [%]
1	2.0854733	2.0854750	8.418e-005
2	13.0694381	13.0698705	3.308e-003
3	36.5948052	36.6041219	2.545e-002
4	71.7112127	71.7795490	9.529e-002
5	118.543772	118.842591	2.521e-001

* Theoretical prediction † Simulation result

Table 1: Theoretical and model natural frequencies

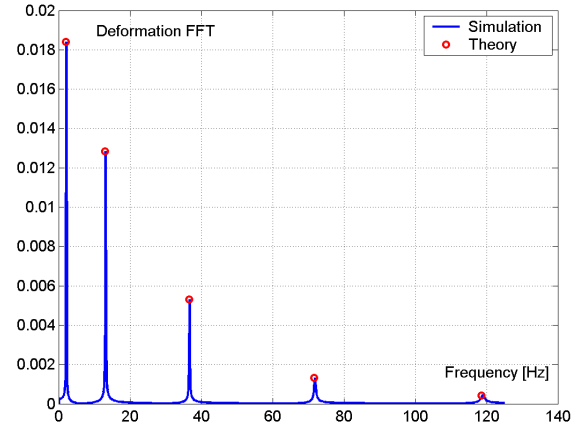


Figure 8: Tip displacement frequency spectrum

sponding to the k^{th} mode of vibration) given by:

$$\omega_k = \beta_k^2 \sqrt{\frac{EJ}{\rho}}, \quad (42)$$

being β_k the k^{th} root of the characteristic equation:

$$\cos(\beta L) \cosh(\beta L) + 1 = 0 \quad (43)$$

The beam, made by aluminium, has square cross section $A = 1 \text{ cm}^2$, length $L = 2 \text{ m}$, density $\rho = 2700 \text{ kg/m}^3$, Young's modulus $E = 7.2 \cdot 10^9 \text{ N/m}^2$ and has been discretized with $N = 10$ elements. The initial tip displacement is 1 cm .

Table (1) contains a comparison between the results for for the first five vibrational modes obtained by simulation and by solving numerically equation (43). The results are in good accordance, as it is shown also in Fig. 8, depicting the tip displacement frequency spectrum.

5.2 Flexible Pendulum

This simulation, reported also in [2], involves the analysis of the vibrations induced by motion in a flexible pendulum swinging under the action of gravity.

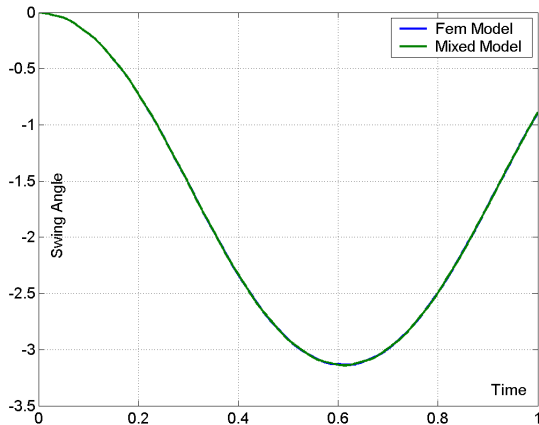


Figure 9: Swing angle

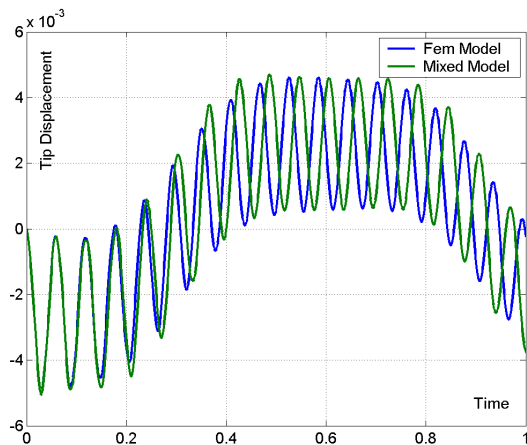


Figure 10: Tip displacement

The pendulum, connected to the world reference system by a revolute joint, has a length $L = 0.4m$, cross sectional area $A = 18cm^2$, density $\rho = 5540kg/m^3$, second moment of area $J = 1.215 \cdot 10^{-8}m^4$ and modulus of elasticity $E = 10^9 N/m^2$. Two different models have been simulated: the first one composed by 10 elements and the second one by 5 volumes with 2 elements each.

In Fig. 9 the swing angle is depicted for both cases. The tip deformation, depicted in Fig. 10, appears to be slightly different for the two models. The results reported in [2] are in accordance with the ones obtained with the mixed model, though.

5.3 Elastic Slider-crank Mechanism

The simulation of an elastic slider-crank mechanism, reported also in [2], has been performed to validate the models for use within closed-loop mechanical chains.

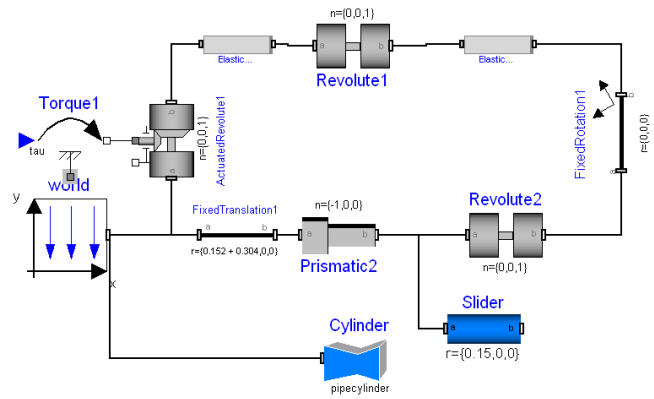


Figure 11: Slider-crank mechanism (Dymola scheme)

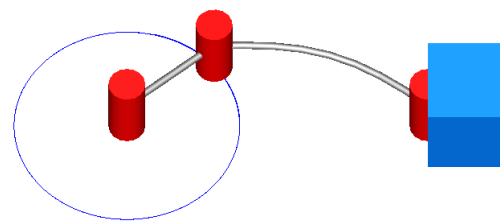


Figure 12: Slider-crank mechanism

The simulation set up involves a slider, a rod and a crankshaft connected by revolute joints (Fig. 11 and 12)

The crank has length $L = 0.152m$, cross sectional area $A = 0.7854cm^2$ and second moment of area $J = 4.909 \cdot 10^{-10}m^4$, density $\rho = 2770kg/m^3$ and modulus of elasticity $E = 10^9 N/m^2$. The connecting rod has the same physical parameters of the crank, apart from the length $L = 0.304m$ and the Young's modulus $E = 5 \cdot 10^7 N/m^2$. The crank and the connecting rod have been discretized with 3 and 8 elements, respectively. Finally, the slider block has been assumed to be a massless rigid body.

During the simulation, the crankshaft is driven by a torque with the following law:

$$\begin{cases} M(t) = [0.01(1 - e^{-t/0.167})]Nm & , t \leq 0.7sec \\ 0 & , t > 0.7sec \end{cases} \quad (44)$$

Fig. 13 and 14 show the slider position and the connecting rod tip transverse displacement, respectively. The results are in perfect accordance with those reported in [2].

6 Conclusion and Future Work

In this paper, a new model for flexible thin beams in *Modelica* is introduced. The model, fully compatible with the *MultiBody* library, is based on the application of the finite element method. Selected simulation results have been presented in order to validate the model properties with respect to scientific literature reference cases.

Future work will include the model extension to handle full 3D deformation and distributed loads. The model will also be employed for the development of applications in the field of robot control and satellite attitude control.

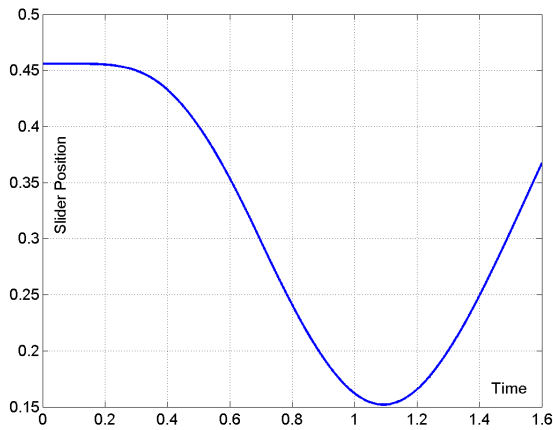


Figure 13: Slider block position

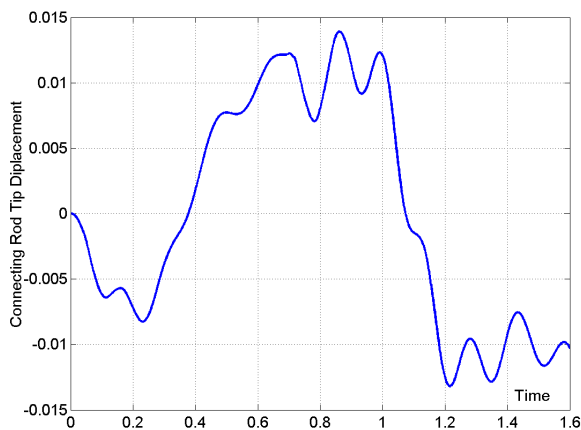


Figure 14: Transverse displacement of the tip of the connecting rod

A Structural Stiffness Matrix

$$K_{ff,el} = \begin{bmatrix} \frac{EA}{\ell} & 0 & 0 & -\frac{EA}{\ell} & 0 & 0 \\ & \frac{12EJ}{3\ell} & \frac{6EJ}{3\ell} & 0 & -\frac{12EJ}{3\ell} & \frac{6EJ}{2\ell} \\ & & \frac{4EJ}{\ell} & 0 & -\frac{6EJ}{2\ell} & \frac{2EJ}{\ell} \\ & & & \frac{EA}{\ell} & 0 & 0 \\ & & & & \frac{12EJ}{3\ell} & -\frac{6EJ}{2\ell} \\ & & & & & \frac{4EJ}{\ell} \end{bmatrix}$$

References

- [1] Dymola. *Dynamic Modelling Laboratory*. Dynasim AB, Lund, Sweden.
- [2] J.L. Escalona, H.A. Hussien, and A.A. Shabana. Application of the absolute nodal co-ordinate formulation to multibody system dynamics. *Journal of Sound and Vibration*, 5(214):833–851, 1998.
- [3] L. Meirovitch. *Analytical Methods in Vibration*. Macmillan Publishing, New York, 1967.
- [4] M. Otter, H. Elmqvist, and S. E. Mattsson. The new modelica multibody library. In *3rd Modelica Conference*, Linköping, Sweden, November 3-4, 2003.
- [5] A. A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, 1998.
- [6] A.A. Shabana. Flexible multibody dynamics: Review of past and recent developments. *Journal of Multibody System Dynamics*, 1(2):189–222, 1997.
- [7] R. Theodore and A. Ghosal. Comparison of the assumed modes and finite element models for flexible multilink manipulators. *International Journal of Robotics Research*, 14(2):91–111, 1995.
- [8] S. Timoshenko, D. Young, and W. Weaver. *Vibration Problems in Engineering*. John Wiley & Sons, New York, 1974.
- [9] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method, 2, Solids and Fluid Mechanics, Dynamics and Non-Linearity*. McGraw Hill, 1991.

Object-oriented modelling of the dynamics of a satellite equipped with Single Gimbal Control Moment Gyros

Tiziano Pulecchi Marco Lovera

Dipartimento di Elettronica e Informazione

Politecnico di Milano

32, Piazza Leonardo da Vinci, 20133 Milano, Italy

Abstract

The development process for spacecraft control systems relies heavily on modelling and simulation tools for spacecraft dynamics. For this reason, there is an increasing need for adequate design tools in order to cope efficiently with tightening budgets for space missions. In this paper, the main issues related to the modelling and simulation of satellite dynamics are briefly summarised, and the results obtained so far in developing Modelica tools for spacecraft simulation are presented and illustrated with a case study for a satellite equipped with Control Moment Gyros as main attitude control actuators.

1 Introduction

The safe and satisfactory operation of a satellite, in terms of its mission objectives, is strongly related to the performance level of its on-board attitude and orbit control systems, which provide the ability to maintain a desired orientation in space (or, e.g., carry out predefined attitude maneuvers) and track a desired, nominal orbit in spite of the presence of external disturbances. In addition, the recent trend towards missions based on constellations or formations of small satellites has led to the formulation of even more complex control problems, related to the relative motion (both in terms of attitude and position) of more vehicles at a time. However, spacecraft designers are also faced with a general reduction of space programmes budget, especially for scientific Low Earth Orbit (LEO) missions, embodied by the spreading of the "faster, better, cheaper" philosophy. This has resulted in an increasing need for efficient design tools in every domain involved in spacecraft design, and particularly in the area of control oriented modelling and simulation. Specific tools have to be developed for the design of both the system architecture and the Attitude and Orbit Control System

(AOCS), bearing in mind the principles of reusability, flexibility and modularity. The main issue in the development of such tools should be to try and work out a unified environment to be used throughout the life cycle of the AOCS software, namely, the mission analysis stage, the preliminary and detailed design and simulation phases, the generation and testing of the on-board code, the development of the AOCS Electrical Ground Support Equipment (EGSE) and the post-launch data analysis activities. A number of commercial tools are available to support one or more of the above mentioned phases in the development of AOCS subsystems, however none of them seems capable of providing complete coverage of the whole development cycle in a sufficiently flexible way.

In particular, the experience gathered in the development of control-oriented spacecraft modelling tools within a "signal oriented" simulation environment (see [2]) showed that a more systematic approach, based on modern acausal object-oriented modelling languages such as Modelica (see [3, 6]), might lead to the development of a spacecraft simulation library the use of which would be made much more efficient by the very nature of the selected modelling approach. Note, in passing, that there is an increasing interest for multidomain problems in the spacecraft control design community (see, e.g., [17]), an area which would benefit from the availability of simulation tools based on the object-oriented approach.

Surprisingly enough, while the use of Modelica for aerospace applications has recently led to the development of a library for flight dynamics (see [14]), very little activity in the spacecraft domain has been reported yet. The development of simulation tools for satellite attitude and orbit dynamics within the object-oriented paradigm has been the subject of previous work (see [10]). Since the development of the model components presented in the cited references, however, a new, more refined version of the Modelica

Multibody library has been released (see [15]) which turns out to be extremely suitable to serve as a basis for the development of the basic model components for the mechanical parts of spacecraft models. In particular, the adoption of the above mentioned library would prove specially beneficial for the simulation of spacecraft equipped with momentum exchange devices, such as, e.g., control moment gyros (CMGs, see, e.g., [8, 22]).

Therefore, the aim of the paper will be to present the current state of the development of spacecraft modelling tools based on the Multibody library, with specific reference to the problem of analysing the (open and closed loop) attitude of satellites equipped with control moment gyros (CMGs) as main attitude control actuators.

The paper is organised as follows: first a brief introduction to the role of mathematical modelling and simulation in the development cycle of spacecraft control system is given, in Section 2; subsequently, an overview of the main model components involved in typical control oriented models will be presented. Finally, the proposed approach to spacecraft modelling will be described in Sections 3-7 and the results obtained in the implementation and application of such an approach to the simulation of spacecraft equipped with CMGs will be presented and discussed in Section 8.

2 Modelling and simulation for AOCS design

As mentioned in the previous Section, the development of the AOCS subsystem for a satellite can be decomposed in the following phases:

1. Feasibility study (conceptual design).
2. Preliminary design.
3. Detailed design.
4. Code generation and testing.

During each of these phases, the designer should be able to rely on appropriate modelling and simulation tools. In particular, modelling tools should be flexible enough to provide the required level of complexity during each of the development phases.

For example, consider the tasks for which a simulation tool would be applied during the feasibility study phase (see Figure 1 and the classical reference [9]):

- Attitude control strategy definition, starting from mission requirements and platform characteristics;
- Evaluation of the external force and torque disturbances acting on the spacecraft, depending on the mission profile. This is normally done using a simple attitude control algorithm to maintain the satellite at nominal conditions;
- Selection and sizing of the actuators in order to counteract disturbances and to maintain the nominal pointing accuracy required by the mission;
- Verification of the possibility to fulfill possible maneuver requirements with the selected actuators.

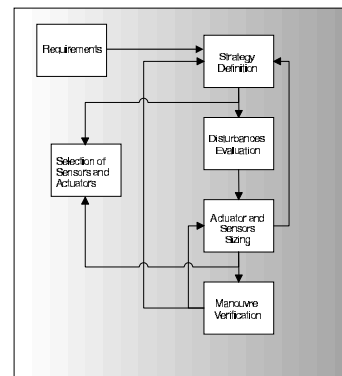


Figure 1: Block diagram of preliminary ACS design process.

The above tasks can be performed without resorting to a dynamic simulator for the spacecraft, however they require accurate modelling capability for orbit dynamics and the availability of reliable models for the space environment. As an example, consider the problem of assessing the external torques acting on the spacecraft. Clearly their characteristics will depend also on the selected control strategy (actuators, sensors and control laws) for the spacecraft, which however is not entirely defined at this stage. Therefore only nominal and/or worst case scenarios need be considered. On the other hand, all the subsequent design phases require the possibility of integrating orbit models and environmental models with a fully dynamic spacecraft simulator, in order to proceed to the refinement of the original design concept. In particular, as the development process goes on, more and more accuracy in the prediction of the achievable performance is required, so that the complexity of the simulation environment is progressively increased.

3 The Modelica Spacecraft modelling library

A library of tools for the modelling and simulation of spacecraft dynamics based on the Modelica language is currently being developed. Modelica turns out to be specially suited for the modelling of spacecraft dynamics under many respects:

- Coordinate frames can be simply included in the model in terms of connectors, describing kinematic transformations from one coordinate system to another.
- Spacecraft dynamics is modelled by defining a Spacecraft class which can be (almost) directly implemented in terms of classical equations for rigid body motion. The data structure to be used in representing all the quantities involved in a specific spacecraft model arises naturally during the modelling process.
- Specific Modelica constructs are available to deal with the modelling of physical fields and environmental quantities. This feature turns out to be extremely useful in modelling the space environment and representing the interaction between the environment and the spacecraft. In particular, with a suitable choice of the environment interfaces, models of increasing complexity for each of the quantities described in Section 5 can be defined. This feature allows for a simple and very convenient implementation of the "scalability" requirement formulated in Section 2.
- Sensors and actuators can also be easily represented in the Modelica paradigm. For example, the generation of magnetic torques is modelled in terms of the interaction with the geomagnetic field, while the momentum exchange between spacecraft and wheels is modelled via a simple mechanical connector allowing one rotational degree of freedom¹.
- Packages of data sheets for each class can be constructed and components easily modified within each spacecraft model, using Modelica's advanced features (see, e.g., [16]).
- Finally, as the components of the library are independent from each other, one can exploit this

¹Mounting errors, which may give rise to interaxis coupling and vibrations, can be easily accounted for.

flexibility in order to build a simulation model of increasing complexity and accuracy according to the needs associated with each phase of the AOCS development process. As an example, one can carry out an analysis of the external disturbance forces and torques acting on the spacecraft in its nominal orbit and attitude, by defining a "simplified" spacecraft ideally attached to its nominal reference attitude.

The original approach to the development of the library contemplated the development of dedicated components also for the mechanical parts. However, the availability of the recently upgraded (see [15]) Multibody Library is leading to some significant changes, since the reuse of the Multibody components would lead to some significant advantages.

The main components of the library are the following:

- A set of basic functions for operations on orbit parameters (transformations between cartesian and orbit elements, see for example [13, 19].
- A similar set of functions for operation on attitude parameters (attitude matrix, quaternions, Euler angles). These have been partially based on the Rigid Body Kinematics Toolbox (see [18]).
- Class definitions for Planet, Orbit, Spacecraft, and the most commonly used actuators and sensors.
- Environmental models of various complexity for gravitational and magnetic field.
- Data sheets for basic model components, such as orbits, actuators and sensors.

4 Dynamics of a spacecraft equipped with momentum exchange devices

For the purpose of the present analysis, the following reference systems are adopted:

- Earth Centered Inertial reference axes (ECI). The origin of these axes is in the Earth's centre. The X-axis is parallel to the line of nodes, that is the intersection between the Earth's equatorial plane and the plane of the ecliptic, and is positive in the Vernal equinox direction (Aries point). The Z-axis is defined as being parallel to the Earth's geographic north-south axis and pointing north. The Y-axis completes the right-handed orthogonal triad.

- Earth Centered Fixed reference axes (ECF).
- Pitch-Roll-Yaw axes. The origin of these axes is in the satellite centre of mass. The X-axis is defined as being parallel to the vector joining the actual satellite centre of gravity to the Earth's centre and positive in the same direction. The Y-axis points in the direction of the orbital velocity vector. The Z-axis is normal to the satellite orbit plane and completes the right-handed orthogonal triad.
- Satellite body axes. The origin of these axes is in the satellite centre of mass; the axes are assumed to coincide with the body's principal inertia axes.

The equations of rotational motion of a rigid spacecraft equipped with momentum-exchange actuators such as CMGs are given by

$$\dot{H} + \omega \times H = T_{ext} \quad (1)$$

where $H = (H_1, H_2, H_3)^T$ is the angular momentum vector of the whole system expressed in the spacecraft body-fixed control axes; $\omega = (\omega_1, \omega_2, \omega_3)^T$ is the spacecraft angular velocity vector; T_{ext} is the global external torque vector applied to the spacecraft, including gravity gradient, solar pressure and aerodynamic torques, expressed in the body-fixed control axes.

The total angular momentum vector consists of the spacecraft main body angular momentum and the angular momentum of the exchange devices, that is

$$H = J\omega + h \quad (2)$$

where J is the overall inertia matrix of the spacecraft and h is the total CMG momentum vector expressed in the body-fixed control axes.

Combining Eqs. (1) and (2), we obtain

$$(J\dot{\omega} + \dot{h}) + \omega \times (J\omega + h) = T_{ext} \quad (3)$$

or, introducing the internal control torque vector generated by CMGs $\tau = -(\dot{h} + \omega \times h)$, we can rewrite Eq. (3) as

$$J\dot{\omega} + \omega \times J\omega = \tau + T_{ext} \quad (4)$$

In addition to the dynamic equations of motion, kinematic equations must be included in the model, which can be easily parameterised in terms of the attitude quaternion

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \quad (5)$$

5 Single Gimbal Control Moment Gyros

A gyroscope, or gyro, is any instrument which uses a rapidly spinning mass to sense and to respond to changes in the inertial orientation of its spin axis. Three basic types of gyroscopes are used on spacecraft: *rate gyros* (RGs) and *rate integrating gyros* (RIGs) are attitude sensors used to measure changes in the spacecraft orientation; *control moment gyros* (CMGs) are used to generate control torques to change and maintain the spacecraft orientation.

A single gimbal control moment gyro (SGCMG) consists of a rotor spinning at a constant rate around an axis that is gimballed to allow changes in its spin direction. The gimbal is rigidly attached to the spacecraft, so that torques generated in response to its input axis rotation apply to the spacecraft itself.

Let $\hat{\theta}_i$ the unit vector along the i -th SGCMG gimbal axis, \hat{h}_i the unit vector along the instantaneous angular momentum, $\hat{j}_i = \hat{\theta}_i \times \hat{h}_i$

Each angular momentum vector depends upon the relevant gimbal angle (for its direction). With respect to the reference frame (spacecraft body axes), the total angular momentum for a system of n SGCMGs is the vector sum of the individual momenta:

$$h(\theta) = \sum_{i=1}^n h_i(\theta_i) = f(h_i, \theta_i, \beta_i) \quad (6)$$

A typical arrangement for SGCMGs is the one in which the CMGs are constrained to gimbal on the faces of a pyramid and the gimbal axes are orthogonal to the pyramid faces. In this case, the overall angular momentum is given by

$$h(\theta) = h_1 \begin{bmatrix} -\cos\beta \sin\theta_1 \\ \cos\theta_1 \\ \sin\beta \sin\theta_1 \end{bmatrix} + h_2 \begin{bmatrix} -\cos\theta_2 \\ -\cos\beta \sin\theta_2 \\ \sin\beta \sin\theta_2 \end{bmatrix} + h_3 \begin{bmatrix} \cos\beta \sin\theta_3 \\ -\cos\theta_3 \\ \sin\beta \sin\theta_3 \end{bmatrix} + h_4 \begin{bmatrix} \cos\theta_4 \\ \cos\beta \sin\theta_4 \\ \sin\beta \sin\theta_4 \end{bmatrix} \quad (7)$$

where β is the skew angle of the pyramid and $h_1 = h_2 = h_3 = h_4$ for the considered cluster of four pyramid-mounted SGCMGs. In particular, when each CMG has the same angular momentum about its spin-rotor axis and the skew angle is chosen as $\beta = 54.73$ deg, the momentum envelope becomes nearly spherical.

The total amount of angular momentum for the system is limited both in value and direction, by individual

SGCMGs momenta. The time derivative of the global CMG angular momentum vector can be obtained as

$$\dot{h}(\theta, \dot{\theta}) = \sum_{i=1}^4 \dot{h}_i(\theta_i, \dot{\theta}_i) = [A(\theta_i)]\dot{\theta} \quad (8)$$

where $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)^T$ is the gimbal angle vector and A is the instantaneous 3×4 Jacobian Matrix.

The gimbal rate command $\dot{\theta}$ is derived in such a way as to supply the required angular momentum for control purposes. A frequently adopted approach is to note that $\dot{\theta}$ can be obtained as

$$\dot{\theta} = A^\dagger \dot{h} \quad (9)$$

where by $A^\dagger = A^T(AA^T)^{-1}$ we denote the Moore-Penrose pseudoinverse of the Jacobian matrix A .

6 Control system

6.1 Attitude control

Since the torque-producing gimbal rates provided by equation (9) can lead to significant problems in the operation of the control law whenever the configuration of the CMGs is such that the Jacobian matrix A is nearly singular, a new strategy must be adopted invoking an approximate solution to equation (8), which must be capable of both minimizing the errors introduced in the output torque supplied and steering the system away from singular states configuration neighborhoods. The errors introduced by this approach in the torque supplied by the SGCMGs cluster can be dealt by the control system as disturbances torques, and appropriately compensated.

In order to determine an inverse solution to equation (8) even when the rank of A is less than 3, the Singularity Robust Inverse solution obtained by solving the following minimization problem must be invoked:

$$\begin{aligned} \min \frac{1}{2} e^T W e \quad (10) \\ e = \begin{bmatrix} \dot{h} - A\dot{\theta} \\ \dot{\theta} \end{bmatrix} \\ w = \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix} \end{aligned}$$

where P and Q are positive definite weighting matrices, that is, $P = P^T > 0$ and $Q = Q^T > 0$.

Minimizing for $\dot{\theta}$, the singularity robust inverse solution can be obtained as

$$\dot{\theta} = A^\# \dot{h} \quad (11)$$

where $A^\# = [A^T P A + Q]^{-1} A^T P$. Note that $[A^T P A + Q] > 0$ and, thus, nonsingular for any set of gimbal angles.

If $Q = 0$, the singularity robust inverse solution has the form of the classical, weighted least squares solution which exists only for a full rank Jacobian matrix A .

6.2 Momentum management

The gimbal angles of a CMG equipped spacecraft may drift to various non-optimal values, due to external disturbances. This can force the spacecraft into a singular state or into a lack of control authority. Typically, a periodic disturbance torque along one spacecraft axis would result in a cyclic variation in the angular velocity of the actuation device directed along that axis, while a constant (secular) disturbance would lead to a linear increase in angular velocity, as the relevant CMG gimbal angle would be accelerated at a constant rate in order to transfer to it the excess of angular momentum due to the external disturbance. This can be sustained up to the physical limit for the rotational speed of the device. In order to prevent this limit from being reached, the so-called desaturation of the actuator must be performed, i.e. an extra set of actuators, generating external torques, must be used to dump momentum from the spacecraft.

Basically, the idea is to use the CMGs cluster to control the spacecraft as required by the Attitude Control System, and to achieve their desaturation by means of a dedicated controller, that keeps the CMGs gimbal rates as small as possible. The goal of the momentum control loop is to maintain the CMGs momentum near zero without interfering with the attitude control loop, that is, the momentum control loop must have a considerably slower response with respect to the attitude control loop.

The external torque to be applied to the spacecraft required to compensate for the gimbal angle offset is taken as

$$T_r = -k h_{cmgs} = -A(\theta)\dot{\theta}_e \quad (12)$$

where

$$\dot{\theta}_e = \frac{\theta - \theta^*}{\Delta t}$$

θ^* is the gimbal angle reference vector and Δt is the time it takes the current gimbal angles to converge to the reference gimbal angles (response time of the momentum control loop).

External torques may be generated by such devices as thrusters or magnetic coils. Since the compensation of the external disturbances is better handled continuously, and given the usual restrictions on waste of

consumable in space applications, magnetic control is usually preferred.

A set of three magnetic coils, aligned with the spacecraft principal inertia axes generate torques according to

$$T_{coils} = m_{coils} \times b \quad (13)$$

where $m_{coils} \in \mathbb{R}^3$ is the vector of magnetic dipoles for the three coils, representing the actual control variables for the coils, and $b(t) \in \mathbb{R}^3$ is the Earth magnetic field described in the body reference frame. The dynamics of the electric coils reduce to a very short electrical transient, and as such can be neglected.

By equating Eqs. (12), (13) and left multiplying by $b(t)$, the magnetic dipole to be applied by the coils and the resulting torque may be derived

$$m_{coils} = -\frac{b \times (A(\theta) \dot{\theta}_e)}{\|b\|^2} \quad (14)$$

$$T_{coils} = -\frac{b \times (A(\theta) \dot{\theta}_e)}{\|b\|^2} \times b \quad (15)$$

7 Model components for CMGs

Taking advantage of the recently released New Modelica Multibody library (see [15]), a set of simulation tools has been developed for satellite attitude and orbit dynamics. Specifically, the following components have been developed:

- **Extended World Model:** a new World model, extending Modelica.MultiBody.World has been defined. It accounts for a more refined description of the Earth's gravitational potential and introduces a model for the geomagnetic field. Such an extension to the basic World model provided in the Multibody library plays a major role in the realistic simulation of the dynamics of a spacecraft as the linear and angular motion of a satellite are significantly influenced by its interaction with the space environment.
- **Extended rigid body model:** similarly, a new rigid body model, extending Modelica.MultiBody.Parts.Body has been defined. The main modification is the possibility of taking into account the interaction between the spacecraft and the geomagnetic field, i.e., to model the effect of magnetic torques applied on the satellite while orbiting around Earth.

- **Rotor:** Model of a Single Gimbal Control Moment Gyro, including as input the gimbal angular rate (fed by the attitude control system) and as output the gimbal angle. Developed using standard Modelica Multibody Library components.
- **Cluster:** Model including the classical set of four pyramid-mounted Single Gimbal Control Moment Gyros arrangement. Developed using standard Modelica Multibody Library components.
- **Attitude Control System:** Block computing the required gimbal rates for the four Single Gimbal Control Moment Gyros used as control torque actuating devices.

For each model component, a short description is given in the following subsections.

7.1 Extended World model

The Extended World Model is an extension of the former Modelica.MultiBody.World model, including among the available selections a more sophisticated model for the Earth's gravity field (described up to the J_2 term of the Earth's gravitational potential) and a model for Earth's magnetic field (modelled up to the quadrupole terms).

As a consequence of Earth's oblateness and not homogeneity, the geomagnetic and gravitational potentials are a non linear function of both the point latitude and longitude, in addition to the distance from the center of the Earth.

The Earth's gravitational potential U_g may be described by the function

$$U_g(r, \theta, \lambda) = -\frac{\mu}{r} \left\{ 1 + \sum_{n=2}^{\infty} \left(\frac{R_e}{r}\right)^n J_n P_n(\cos(\theta)) + \sum_{n=2}^{\infty} \sum_{m=1}^n \left(\frac{R_e}{r}\right)^n P_n^m(\cos(\theta)) (C_n^m \cos(m\lambda) + S_n^m \sin(m\lambda)) \right\}$$

where P_n^m are the Legendre polynomials

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n$$

$$P_n^m(x) = (1 - x^2)^{m/2} \frac{d^m P_n(x)}{dx^m}$$

R_e is the mean equatorial Earth radius, r, θ and λ are the point's spherical coordinates and coefficients J_n, C_n^m, S_n^m are the zonal, sectoral and tesseral coefficients. For the purpose of attitude control simulations a satisfactory approximation can be obtained by neglecting

n	m	g [nT]	h [nT]
1	0	-29682	
1	1	-1789	5318
2	0	-2197	
2	1	3047	-2356
2	2	1685	-425

Table 1: Coefficients of the geomagnetic field model.

the terms after J_2 . The Earth gravitational field components (expressed in spherical coordinates) are then given by

$$g = -\nabla U_g = -\left\{ \frac{\partial U_g}{\partial r}, \frac{1}{r} \frac{\partial U_g}{\partial \theta}, \frac{1}{r \sin(\theta)} \frac{\partial U_g}{\partial \lambda} \right\}. \quad (16)$$

As for the geomagnetic potential U_m , it is described by the function

$$U_m(r, \theta, \lambda) = \frac{R_e}{\mu} \sum_{n=0}^{\infty} \sum_{m=0}^n \left(\frac{R_e}{r} \right)^{n+1} P_{nm}(\cos(\theta)) (g_n^m \cos(m\lambda) + h_n^m \sin(m\lambda))$$

where g_n^m and h_n^m are the Gauss coefficients appropriate to the Schmidt polynomials P_{nm}

$$P_{n,0}(x) = P_n^0(x)$$

$$P_{n,m}(x) = \left(\frac{2(n-m)!}{(n+m)!} \right)^{1/2} P_n^m(x).$$

A first approximation for the geomagnetic potential is obtained by neglecting the terms after the quadrupole. The coefficients for the geomagnetic potential adopted in the simulation environment correspond to the so-called International Geomagnetic Reference Field (IGRF) model for the Earth's magnetic field and are given in Table 1. The components of the geomagnetic field (expressed in spherical coordinates) are then given by

$$B = -\nabla U_m = -\left\{ \frac{\partial U_m}{\partial r}, \frac{1}{r} \frac{\partial U_m}{\partial \theta}, \frac{1}{r \sin(\theta)} \frac{\partial U_m}{\partial \lambda} \right\}. \quad (17)$$

The new function `magneticField`, embedded in the `World` model, receives as input the body position and provides as outputs the corresponding components of the geomagnetic field vector B , expressed in the ECI reference frame. Since the geomagnetic field model is defined with respect to the rotating ECF reference frame, an additional parameter UT_0 , defining the initial value for the Universal Time (UT) and whose default value is set to zero, allows to start the simulation with a desired initial rotation of the ECF reference frame with respect to the ECI reference frame.

7.2 Extended rigid body model

This new component extends the `Modelica.MultiBody.Parts.Body` model to account for the effects of the Earth's magnetic field in terms of torques applied to the satellite. Indeed, if the spacecraft possesses a magnetic dipole m , it experiences an external torque T given by

$$T = m \times B, \quad (18)$$

where B is the geomagnetic field vector in body coordinates. Note, in passing, that the ability of taking into account in the rigid body model the interaction with the geomagnetic field makes it possible to fulfill two different modelling goals: first of all to simulate the effect on the satellite dynamics of a *residual magnetic dipole*, such as due to the internal magnetic field generated by on-board electrical equipment; furthermore, it is possible to model the effect on the angular motion of the spacecraft of *magnetic actuators*, which, as previously mentioned, are frequently used, specially in small satellite missions, for attitude or momentum management.

7.3 Rotor

The `SpacecraftDynamicsLibrary.Rotor` model simulates a Single Gimbal Control Moment Gyro, which has been chosen as the primary actuation source for the satellite attitude manoeuvring and control. The following standard Modelica library components have been employed:

- `MultiBody.Interfaces.Frame_a`, used as the SGCMG-satellite connecting point.
- `MultiBody.Joints.ActuatedRevolute`, used to model the SGCMG rotational degree of freedom. The gimbal speed is driven through a `Mechanics.Rotational.Speed` by the input signal coming from the satellite attitude control system (`SpacecraftDynamicsLibrary.ACS` block).
- `Two Mechanics.Rotational.Speed`
- `Blocks.Interfaces.InPort`, feeding to the SGCMG the desired gimbal rate.
- `Mechanics.Rotational.Sensors.SpeedSensor`
- `MultiBody.Parts.FixedTranslation`
- `MultiBody.Parts.Rotor1D`, used as the physical rotor. Its speed is kept constant by means of

an electrical motor, whose dynamic has not been modelled at this stage, and which is represented by a `Blocks.Sources.Constant`. The rotor rotational inertia has been assigned such that the resulting angular momentum along the rotor axis is $1.76Nms$.

- `Blocks.Continuous.Integrator`
- `Blocks.Sources.Constant`, forcing the rotor angular rate to the chosen 4000 rpm nominal value.
- `Blocks.Interfaces.OutPort`, used to output to the satellite attitude control system (`SpacecraftDynamicsLibrary.ACS` block) the actual gimbal angle.

7.4 Attitude Control System

The `SpacecraftDynamicsLibrary.ACS` block computes the four SGCMGs gimbal rates required to control the satellite attitude.

The computation is performed in two separate steps. First, the required control torques are computed, by means of a suitable state feedback gain (designed via LQR control techniques), where the system states considered in the control algorithm are the quaternion errors and the satellite angular rates errors. Subsequently, the gimbal rates are derived in such a way as to supply the required torques (via variation of the SGCMGs cluster angular momentum). To this purpose, the control moment gyro steering logic proposed by Wie, Bailey and Heiberg [22] was adopted.

The `SpacecraftDynamicsLibrary.ACS` provides the four SGCMGs gimbal rates, and receives as inputs the gimbal angles, the satellite inertial attitude quaternion, the measured gimbal rates and the unit vectors of the local orbital frame.

7.5 Cluster

The `SpacecraftDynamicsLibrary.Cluster` model simulates the classical configuration of four identical pyramid mounted Single Gimbal Control Moment Gyros, with a skew angle $\beta = 54.73^\circ$. It employs four `SpacecraftDynamicsLibrary.Rotor` models.

Receives as input the SGCMG gimbal rates computed by the `SpacecraftDynamicsLibrary.ACS` block.

8 Simulation study

In order to analyze the performance of the spacecraft dynamics library components developed, a specific

mission scenario has been considered, namely:

- Spacecraft in equatorial orbit (0° inclination).
- Orbit altitude of 450 Km (a typical altitude for a small scientific mission).
- The attitude control must keep an Earth pointing attitude, aligned with the Pitch-Roll-Yaw reference frame (orbital frame).
- The spacecraft is provided with a star sensor for attitude determination (i.e., an ideal state feedback situation is considered).
- Attitude control is based on a set of four pyramid mounted Single Gimbal Control Moment Gyros.
- The spacecraft is subject to the effect of a magnetic disturbance torque, due to a residual magnetic dipole for which a value of $1Am^2$ along each body axes has been assumed.

As an illustrative example, a simulation has been carried out in which the initial conditions for the spacecraft are characterised by a high value of the components of the angular rate, such as would occur during the initial acquisition of the desired Earth pointing attitude. Figure 2 shows how three axis attitude control is achieved by means of the four SGCMG actuators. In particular, note that the residual pointing error due to the geomagnetic disturbance torque is hardly visible. The corresponding time history of the geomagnetic field along the considered orbit and of the associated disturbance torque are shown in Figure 3. As expected, since the considered spacecraft is operating in an equatorial orbit, the only significant component of the geomagnetic field is aligned with the orbit normal (Z ECI axis). In spite of this, it is interesting to note that because of equation (18) the spacecraft is subject to a disturbance torque along all three axes.

As was mentioned previously, external disturbance torques force the gimbal angles of the CMGs to slowly drift away from their optimal value. In particular, the periodic component of the magnetic torques force a cyclic variation in the angular rate of the SGCMGs, while the secular component lead to a linear increase in the gimbal angular rates. This can be sustained up to the physical limit for the rotational speed of the device. In order to prevent this limit from being reached, the actuator must be desaturated, i.e. an extra set of actuators, generating external torques, must be used to dump momentum from the spacecraft.

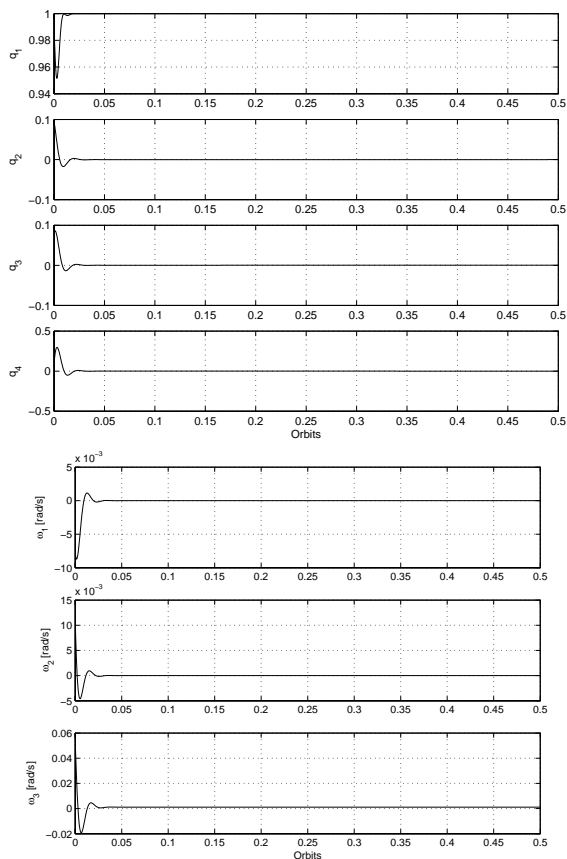


Figure 2: Quaternion and angular rates for the attitude acquisition.

9 Concluding remarks

The main issues related to the modelling and simulation of satellite dynamics have been described, the results obtained so far in developing Modelica tools for spacecraft simulation have been presented and a case study for a satellite equipped with Control Moment Gyros as main attitude control actuators has been illustrated by means of a simulation study.

10 Acknowledgements

This activity is supported by the Italian National Research Project “New Techniques of Identification and Adaptive Control for Industrial Systems”.

References

- [1] Alberti, M., De Rocco, L., Morea, D., and Lovera, M. Environmental external torques estimation of a low Earth orbiting satellite based on

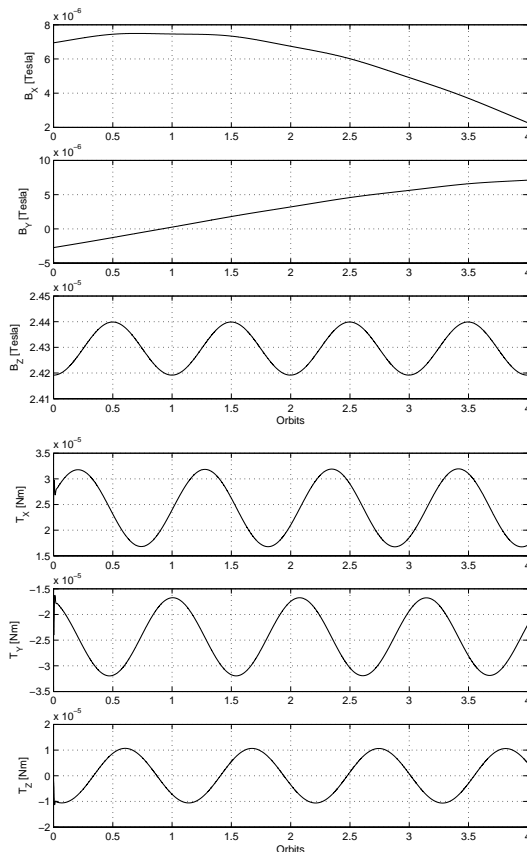


Figure 3: Geomagnetic field and magnetic disturbance torque along the considered orbit.

MITA platform flight data. In: 52nd IAF International Astronautical Congress, Toulouse, France, 2001.

- [2] Annoni, G., De Marchi, E., Diani, F., Lovera, M. and Morea, G.D. Standardising tools for attitude control system design: the MITA platform experience. In: Data Systems in Aerospace (DASIA) 1999, Lisbon, Portugal, 1999.
- [3] Modelica Association, Modelica - a unified object-oriented language for physical systems modelling. Language specification. Technical report, <http://www.modelica.org>, 2002.
- [4] Bate, R.R., Mueller, D.D. and White, J.E., Fundamentals of astrodynamics. Dover Publications Inc., 1971.
- [5] Della Torre, A., Lupi, T., Sabatini, P. and Coppola, P. MITA: the Advanced Technology Italian Minisatellite. In: Data Systems in Aerospace (DASIA) 1999, Lisbon, Portugal, 1999.

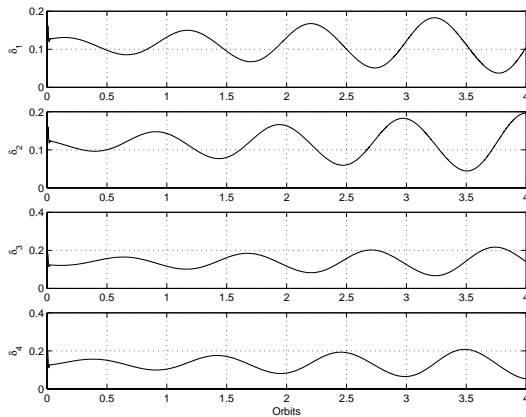


Figure 4: Gimbal angles for the attitude acquisition.

- [6] Fritzon, P. and Bunas, P. Modelica - a general object-oriented language for continuous and discrete-event system modelling and simulation. In: Proceedings of the 35th IEEE Annual Simulation Symposium, San Diego, CA, 2002.
- [7] Hughes, P. Spacecraft attitude dynamics. John Wiley and Sons, 1986.
- [8] Lappas, V. J., Steyn, W. H., and Underwood, C.I. Practical Results on the Development of a Control Moment Gyro based Attitude Control System for Agile Small Satellites. In: Proceedings of the 16th AIAA/USU Small Satellite Conference, Session SSC02-VIII-8, 2002.
- [9] Larsen, W.J. and Wertz, J.R., editors. Space mission analysis and design. Kluwer Academic Publisher, 1992.
- [10] Lovera, M. Object-oriented modelling of spacecraft attitude and orbit dynamics. In: Proceedings of the 54th International Astronautical Congress, Bremen, Germany, 2003.
- [11] Lovera, M. and Astolfi, A. Spacecraft attitude control using magnetic actuators, *Automatica*, 40(8):1405–1414, 2004.
- [12] Martinez-Sanchez, M. and Pollard, J.E. Spacecraft electric propulsion - an overview. *Journal of Propulsion and Power*, 14(5):688–699, 1998.
- [13] Montenbruck, O. and Gill, E. Satellite orbits: models, methods, applications. Springer, 2000.
- [14] Moorman, D. and Looye, G. The Modelica flight dynamics library. In: Proceedings of the 2nd International Modelica Conference, Oberpfaffenhofen, Germany, 2002.
- [15] Otter, M., Elmqvist, H. and Mattsson, S. E. The New Modelica MultiBody Library. In: Proceedings of the 3rd International Modelica Conference, Linköping, Sweden, 2003.
- [16] Otter, M. and Olsson, H. New features in Modelica 2.0. In: Proceedings of the 2nd International Modelica Conference, Oberpfaffenhofen, Germany, 2002.
- [17] Roithmayr, C., Karlgaard, C., Kumar, R. and Bose, D. Integrated Power and Attitude Control with Spacecraft Flywheels and Control Moment Gyroscopes. *Journal of Guidance, Control, and Dynamics*, 27(5):859-873, 2004.
- [18] Schaub, H. and Junkins, J.L. Matlab toolbox for rigid body kinematics. In: AAS/AIAA Space Flight Mechanics Meeting, Breckenridge, CO, 1999.
- [19] Sidi, M. Spacecraft dynamics and control. Cambridge University Press, 1997.
- [20] Silani, E. and Lovera, M. Magnetic spacecraft attitude control: A survey and some new results. *Control Engineering Practice*, 13(3):357-371, 2005.
- [21] Wertz. Spacecraft attitude determination and control. D. Reidel Publishing Company, 1978.
- [22] Wie, B., Bailey, D., and Heiberg, C. Singularity Robust Steering Logic for Redundant Single-Gimbal Control Moment Gyros. *Journal of Guidance, Control, and Dynamics*, 24(5):865-872, 2001.

Collision Handling for the Modelica MultiBody Library

Martin Otter², Hilding Elmqvist¹, José Díaz López¹

¹Dynasim AB, Lund, Sweden, {Elmqvist, Jose.Diaz}@Dynasim.se

²DLR Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany, Martin.Otter@DLR.de

Abstract

The Modelica MultiBody library is extended with collision handling. It is demonstrated how to use this new feature. Different implementations are explained based on parametric surfaces, on surfaces described by algebraic constraints, and on surface descriptions by primitives and triangles using the collision package SOLID 3.5. Furthermore, the response calculation by a resultant contact force and torque is discussed.

1 Introduction

Modeling of contacts between mechanical objects is important in many disciplines such as wheel handling for vehicle dynamics, robot gripping, CAM modeling, etc. The Modelica.Mechanics.MultiBody library [13] is extended with support for collision handling. The user interface and the implementation variants are discussed in the next sections.

Describing collisions between mechanical bodies is still a difficult topic. The solution can be divided into two main steps:

(1) Collision detection of surfaces, determining features such as shortest distances, penetration depths and contact normal vectors. Several software systems are available for this task, e.g., SWIFT [6], ODE [16] or SOLID [2][3]. This is also an important part of CAD and FEM systems. Fast real-time solutions are mainly driven by the game industry due to their particular needs [5][16].

(2) Calculation of the contact response. Several quite different approaches are in use:

(2a) The response is computed in an idealized way using impulses based on an impact law such as Poisson's hypothesis: relating the impulses of the compression and decompression phase of an impact to each other, see, e.g., [15][10][5]. The main advantage is that only few constants are needed to describe the impact law and that the integrator step size is not influenced by the response calculation because it is

performed in an infinitely small time instant. The disadvantages are that such idealized impact laws are only valid for stiff collisions and that the constants of the impact law cannot be computed by material properties of the colliding bodies, i.e., they must be determined by measurements. Furthermore, it is quite involved to compute the new initial conditions after an impact in a robust way, especially if *several* surface *contacts* are present at the same time instant. In the latter case either no or infinitely many solutions may exist using impulse descriptions. For a physically meaningful response there are cases where multiple impacts have to be applied one after each other whereas also cases are present where they must be applied altogether (for a more thorough discussion, see [5], pp. 256 – 264).

(2b) The response is computed by a simple elastic spring/damper element. E.g., the spring force is just proportional to the penetration depth. The advantages of this approach are its simplicity and that it can be used for stiff and soft contacts. This approach works also reasonably well if several contact points are present at the same time instant. The disadvantage is that the integrator step size is reduced significantly in the contact phase in order to catch the rapidly changing contact forces and torques. A necessary and harder task is to determine experimentally the spring and damper constants. Those are in consequence only valid in situations close to the experimental conditions. The main reason is that the contact force is not only proportional to the penetration depth but rather to the contact area and the contact volume.

(2c) The response is computed by taking into account the contact area and the contact volume. This might be performed by a discretization of the contact area or the contact volume, see, e.g., [8][9]. Contrary to (2b), the force and torque computation will be more precise and the material properties, such as the E-modul and the contraction number ν , can be used to calculate the spring constants. Furthermore, the contact torque can be calculated in a reasonable way. This torque is particularly important for gripping operations.

(2d) The response is computed for special situations, e.g., for wheel/road contact [14]. Solutions that are specialized to a particular contact problem are usually more precise and practically applicable as the generic solutions of (2a,b,c).

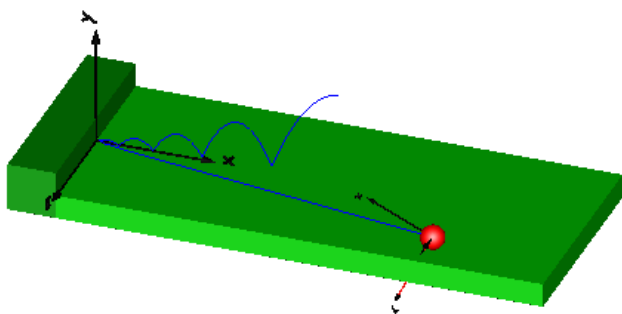
At the moment, it is not possible to implement the solution with impulses (2a) in a *generic* way in Modelica. For special cases it can be implemented with the `reinit(..)` operator. The reason is that an appropriate Modelica language element is missing as well as the needed symbolic algorithms for the most general cases. In the European project “RealSim” basic research was carried out to handle models with *varying index* and with *dirac impulses*. The latter might be implicitly occurring at switching points where the number of states, and therefore also the DAE index, is changing. For certain classes of systems a reasonable solution method was developed [11]. Still, the algorithms are in a research stage.

For this reason, in this article only elastic response actions according to (2b) and (2c) are taken into account. A specialized solution for wheel/road contact is already available in the Modelica VehicleDynamics library [1].

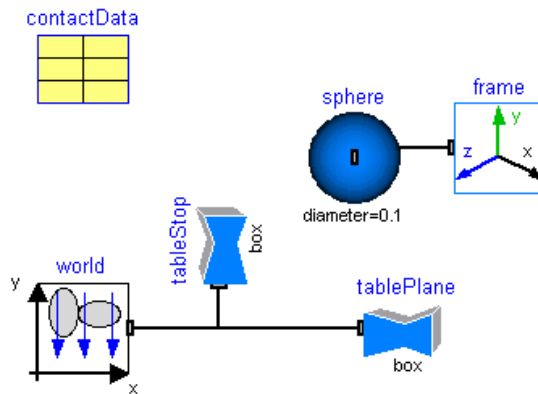
2 Users View

In this section the user’s view of the library is shortly sketched. This view is independent of the implementation variants discussed in subsequent sections.

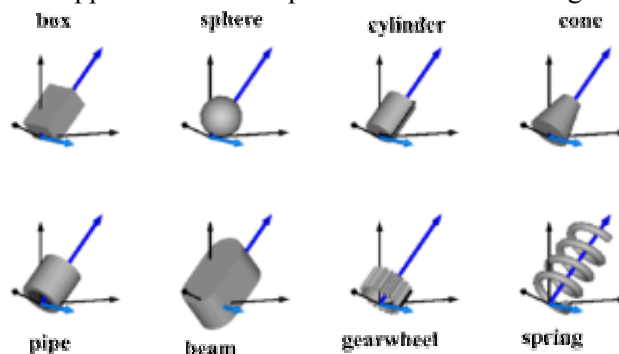
Components are provided to handle collisions between bodies using elastic force/torque laws at contact points. An example is shown in the following figure where a ball is thrown on a table. The ball first bounces on the table, then into the wall and finally rolls on the table.



This example is defined by the Modelica model of the next figure. In brief, there is a modified `MultiBody.World` component with a new subcomponent named “collisionHandling”. This new object performs collision detection and contact response calculations. The table is defined by two boxes of the component “MultiBody.Visualizers.FixedShape”.

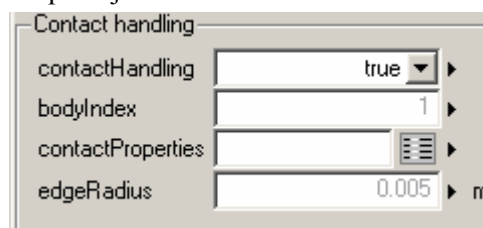


The ball is described by a sphere of the component “FixedShape” together with body properties, such as mass and inertia. The record “contactData” contains material constants that are used in the table and sphere components. No special collision handling objects are needed. Instead, the existing “MultiBody.Visualizers.FixedShape” component has been modified to optionally detect and treat collisions for the supported visual shapes shown in the next figure:



For collision detection, shapes “pipe”, “gearwheel” and “spring” are treated as full cylinders. There are currently limitations for shapes using “*.dxf” files (AutoCAD R12 descriptions): Only sets of triangles are supported and only one contact point between two surfaces is taken into account, although more contact points might be present for non-convex objects. Note, all objects can be scaled in the 3 coordinate axes by providing length, height and width of the shape. E.g., ellipsoids are also supported by defining `shapeType="sphere"` and appropriate length, height, and width scaling.

The following new parameters are present in a FixedShape object:



The collision handling has to be explicitly activated by setting “**contactHandling = true**”. The effect in

this case is that the distance between this object and all other objects that have `contactHandling = true` is continuously computed and monitored. When the distance between two objects becomes zero, an event is triggered and a contact response is applied.

If two `FixedShape` objects are rigidly attached to each other (see, e.g., the two boxes representing the table in the example above), a contact would permanently be present. To avoid this, all objects are reduced in size by a factor of "1 - 1.e-9" for the collision detection. As a consequence, shapes that are fixed together, do not lead to an unnecessary contact response computation.

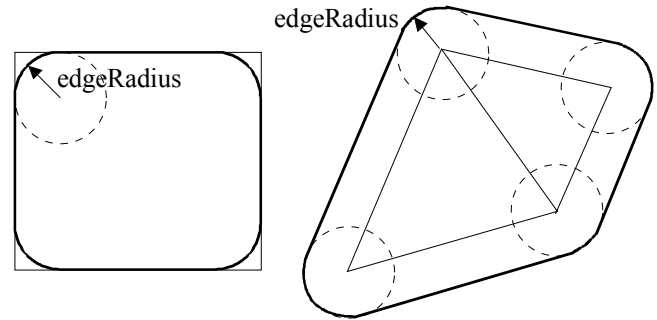
Parameter "**bodyIndex**" is a unique Integer identifier for each "`FixedShape`" object. For instance, if 4 `FixedShape` objects with `contactHandling = true` are present, they must have `bodyIndex = 1, 2, 3, and 4`. Additionally, in the "World" object, parameter "`nContactBodies`" has to be set to the number of `FixedShape` objects with `contactHandling = true`. In the example above, `nContactBodies = 4` is required. There is currently a Modelica language enhancement under development, in order that this user input is no longer required since it can be automatically deduced by a Modelica translator.

The data for the response calculation are provided via parameter record "**contactProperties**", see next figure. It defines material data of the corresponding surface. The type of response calculation used for all collisions is defined in the World object: If parameter `simpleResponse = true`, a linear spring and a linear damper force acts in contact normal direction. Additionally, linear rotational damping proportional to the relative angular velocity is present in contact normal direction, and a sliding friction force acts in opposite direction to the tangential sliding velocity at the contact point. If `simpleResponse = false`, the contact area is discretized and a resultant force and torque is computed by summation of appropriate forces over the contact area. The latter option is currently under development. More details of the force/torque calculations are given in section 4.5.

Parameters			
c	1.0e7	N/m	Spring constant
d	0.0	N.s/m	Damping constant
d_w	0.0	N.m.s/rad	Rotational damping
mu	0.0		Friction coefficient
v_min	1.e-3	m/s	Start of sliding friction force
w_min	1.e-3	rad/s	Start of sliding friction torque

Finally, parameter "**edgeRadius**" defines how much the edges of primitive shapes, such as boxes, cylinders etc., are "rounded" with spheres, see left part of

figure below. For "*.dxf" files, a *layer* of spheres with radius "edgeRadius" is put on the surfaces to get a smooth surface description, too, see right part of figure below. The edge rounding and the "layer of spheres" is used for collision detection and response calculation. It is currently **not** shown in the rendering (animation). It is recommended to use a non-zero `edgeRadius` because the collision detection will be usually faster and more robust. Still, it is possible to set `edgeRadius=0`. The technique of smoothing the surfaces with spheres is from [3][2].



In sublibrary `MultiBody.Parts` the available body components have now also optional collision handling support. Furthermore, new body types have been added, as shown in the next figure:



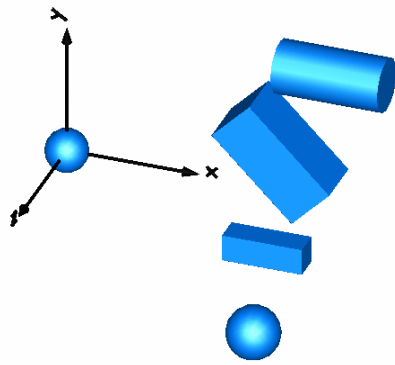
For example, "`BodyEllipsoid`" is a part that defines an ellipsoid by length, width, height and material properties. From this information, the body properties (mass, center of mass, inertia tensor) are computed and the rendering and collision handling is deduced.

3 Applications

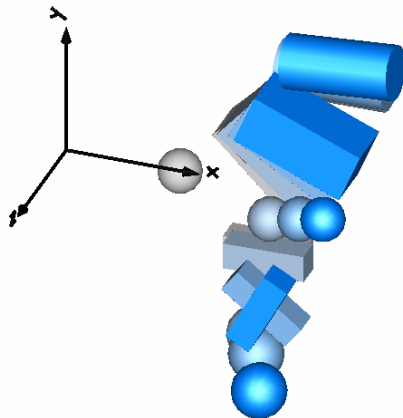
In this section some applications of the library are shown.

3.1 Free Flying Objects

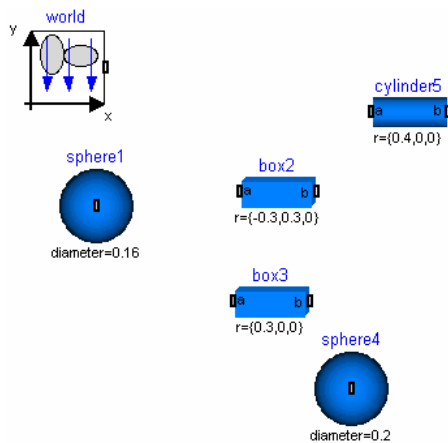
Five different free flying objects are colliding with each other. The start configuration is shown in the next figure. The 4 objects on the right are in rest at the beginning and the sphere at the left side is flying in the direction of the other objects.



Several collisions between all objects occur after a few seconds:

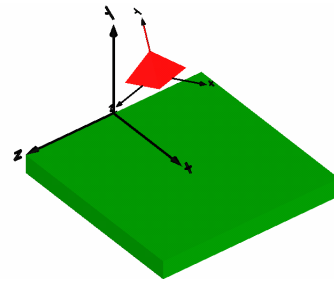


This system is defined with the following Modelica model:

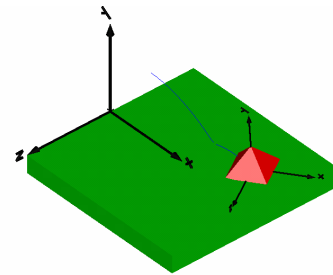


3.2 Collision of triangularized surface with a table

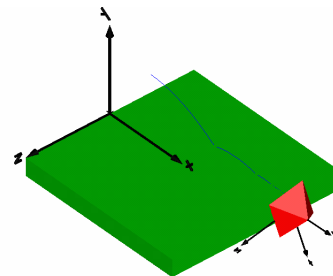
A simple application of AutoCAD files in the library is shown in the following example: An AutoCAD generated pyramid is rotating around his main axis and falls to a table.



After some time we observe how the trajectory of the pyramid evolves. Due to friction, the velocity and angular velocity of the pyramid is permanently reduced.



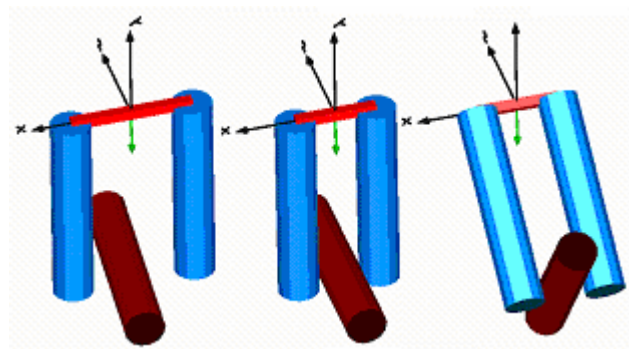
The final position after gliding over the surface is shown in the next figure



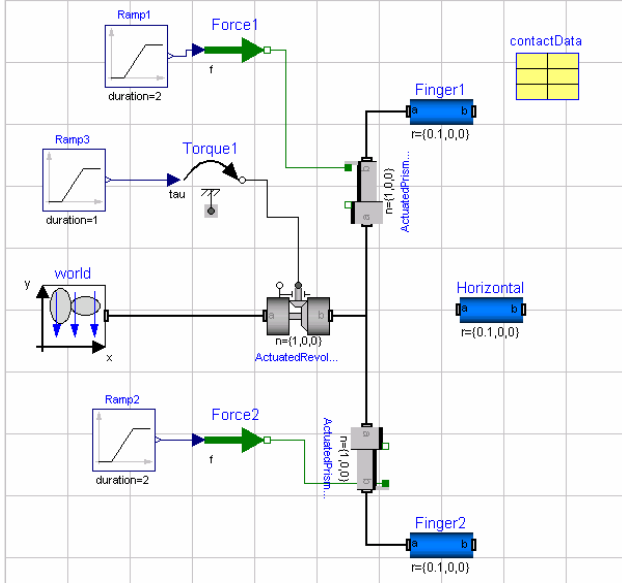
as the pyramid is falling over the edge of the table with a velocity that is almost zero.

3.3 Gripping

The sequence of images below shows two blue fingers gripping a lying red object (all cylindrical dx-defined objects). The fingers are attached with prismatic joints to a revolute joint. The lying object is gripped since it is squeezed between the fingers. Due to the friction torque between the surfaces, the red cylinder is elevated after gripping it and rotating the revolute joint.



The model of this experiment setup is shown below. “Horizontal” is the lying red cylinder and “Finger1” and “Finger2” are the two fingers. This example shows the important role of the friction torque. If this feature would not be present in the model, then the red cylinder would rotate after gripping.



4 Implementation

In this section implementation details and variants for the collision handling are discussed.

4.1 Central Collision Handling

Since distances and contact response calculations are needed between any two collision objects, a central collision handling is present in the modified World component. In order that this is possible, every collision object needs a unique Integer identifier. The surface data, position, orientation, forces and torques are copied in appropriate arrays using the corresponding Integer identifier as index in these arrays. Currently, this identifier has to be provided manually by the user as shown in section 2.

As already mentioned, a Modelica language enhancement is under development to get rid of this unnecessary user input. The current plan is to introduce the new dimension qualifier “each” and the new operator “uniqueElement(..)” to automatically provide a unique array index for objects. Examples:

```
Real vec[each,5,3];
Real subvec[5,3] = uniqueElement(vec);
Real x[each];
Real xv = uniqueElement(x);
```

The following rules apply:

(1) If a public array component, A, is declared using the subscript [each], e.g. Real A[each], it has the same access restriction as though it were protected except the “uniqueElement” operator can be applied to the array. This is the only allowed use of the uniqueElement operator and the only allowed use of the array name outside the declared scope.

(2) The size of all array components with declared size [each] starts at zero and is then increased as follows. This is performed before size() of the array can be determined (e.g. to determine the size of other arrays).

(3) For each use of the uniqueElement(..) operator the size of the array component is increased by one and a unique element of the array is referenced.

These new language elements will allow an implementation where the unique collision object indices are automatically derived without requiring them from the user. This feature will be also useful for other purposes.

4.2 Variant 1: Parametric surfaces

The first implementation variant for the collision handling uses parametric surfaces. That is, the absolute position vector \mathbf{r} to each surface point is described as a vector valued function of two parameters, called α and β .

$$\mathbf{r} = \mathbf{r}(\alpha, \beta)$$

Two tangent vectors \mathbf{e}_α and \mathbf{e}_β are defined by partial derivatives from which the normal \mathbf{n} to the surface can be computed:

$$\mathbf{e}_\alpha = \frac{\partial \mathbf{r}}{\partial \alpha} = \mathbf{r}_\alpha(\alpha, \beta)$$

$$\mathbf{e}_\beta = \frac{\partial \mathbf{r}}{\partial \beta} = \mathbf{r}_\beta(\alpha, \beta)$$

$$\mathbf{n} = \mathbf{e}_\alpha \times \mathbf{e}_\beta = \mathbf{n}(\alpha, \beta)$$

Both position and orientation of the surface patch close to a possible contact point are defined as functions of α and β . The tangential planes of two surfaces that are potentially colliding are constrained to be parallel, i.e., their normal vectors are parallel (quantities belonging to surface “a” are denoted by superscript “a”, e.g., x^a):

$$\mathbf{n}^a(\alpha^a, \beta^a) + \lambda_1 \cdot \mathbf{n}^b(\alpha^b, \beta^b) = 0$$

Furthermore, the relative position vector of the contact point candidates is constrained to be parallel to the surface normals:

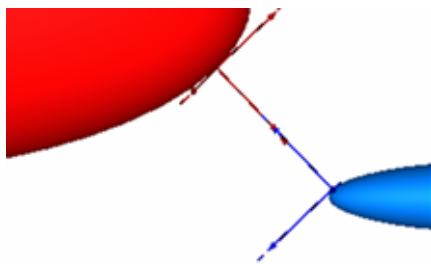
$$\mathbf{r}^b(\alpha^b, \beta^b) - \mathbf{r}^a(\alpha^a, \beta^a) = \lambda_2 \cdot \mathbf{n}^a(\alpha^a, \beta^a)$$

These constraints constitute 6 scalar equations in the unknown variables $\alpha^a, \beta^a, \alpha^b, \beta^b, \lambda_1, \lambda_2$. These equations are in general nonlinear and have to be solved per each potential collision point pair. We may encounter computational problems since multiple solutions may exist. For closed surfaces, at least 4 different solutions exist (closest-closest, closest-farthest, farthest-closest, farthest-farthest). A feasible solution must have a positive scalar product between the surface normal and the relative distance:

$$\mathbf{n}^a(\alpha^a, \beta^a) \cdot (\mathbf{r}^b - \mathbf{r}^a) > 0$$

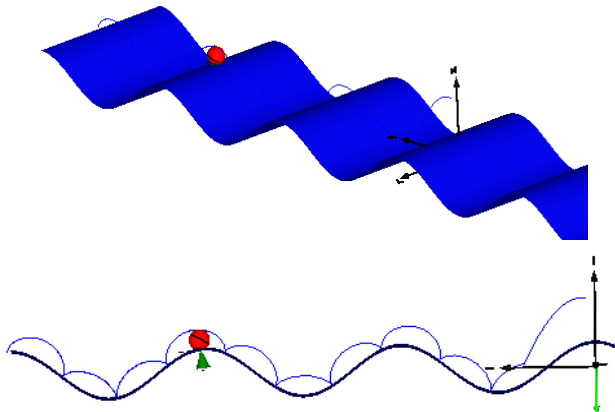
For convex surfaces, the solution of the above nonlinear system of equations, taking into account the inequality constraints, gives the two closest points when the bodies are apart. It is possible to track the correct closest points also for non-convex surfaces provided good starting values are given for the unknown variables.

The local coordinate systems at these points as well as the relative position vector are illustrated below as rendered by Dymola during animation



Certain special surfaces such as ellipsoid, plane and parabola have been implemented as parameterized surfaces. It is easy to add other surfaces by defining corresponding parametric functions.

An example for a non convex parameterized surface is shown below where a ball is thrown towards left on a “cosine” surface.



The side view shows the trajectory of one point on the ball. The ball oscillates forth and back in the

leftmost valley with the fixed point following the same path.

The major drawback of closed parameterized surfaces is the occurrences of singular points. For example, every closed surface has at least one singular point where the calculation of the normal vector fails because at least one of the tangent vectors becomes zero. For a sphere, these are the “north” and “south” pole of the sphere. Therefore, in general it is not possible to get a robust solution of the non-linear system of equations. For special cases where it is guaranteed that the singular points are outside of the operation region, a parametric surface might be used without re-parameterization, see, e.g., [12] that demonstrates collision handling of a CAM. However, for a generic collision handler, parametric surfaces are difficult to treat, since singular points require a re-parameterization of the surface description.

4.3 Variant 2: Algebraic constraint surfaces

An alternative is to describe especially closed surfaces with constraint equations

$$0 = h(\mathbf{r})$$

For example, a sphere can be defined by

$$h(\mathbf{r}) = x^2 + y^2 + z^2 - 1$$

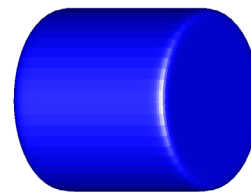
An approximate representation of a box can be made by

$$h(\mathbf{r}) = x^{20} + y^{20} + z^{20} - 1$$

and an approximation to a cylinder can be made by

$$h(\mathbf{r}) = x^{20} + (y^2 + z^2)^{10} - 1$$

as shown below.



By choosing higher values of the exponent, the edges get sharper. It is possible to define cones and pyramids as well by such closed formulas. However, it is difficult to find the closed formula for arbitrary surfaces.

This representation allows the smooth normal to be calculated as the gradient

$$\mathbf{n}(\mathbf{r}) = \text{grad}(h(\mathbf{r})) = \left\{ \frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{\partial h}{\partial z} \right\}$$

It should be noted that there are no singular points when using this surface representation. Inserting the

definition of the normals into the same constraint equations as used in variant 1, and considering the constraint equations for each surface, 8 scalar equations in the unknown variables $\mathbf{r}^a, \mathbf{r}^b, \lambda_1, \lambda_2$ are obtained. Note, that start values are easier to give in this representation since the position vectors themselves are unknowns.

This approach has the advantage to get smooth surface descriptions. The drawback are the highly nonlinear equations closed to the edges, especially for high exponents.

In both variant 1 and 2, partial derivatives of functions defining the surfaces are required in the model code. This can be achieved by a special operator in the Modelica code and automatic differentiation. For details see, [12].

4.4 Variant 3: Collision handling with SOLID

As a third variant, the collision detection of shapes and the computation of the penetration depth between shapes is performed with the software system SOLID 3.5 [3][2]. The SOLID software is free for non-commercial purposes. Commercial use requires a license. The SOLID software supports collisions between primitives such as spheres and cylinders, as well as between complex convex and non-convex objects described by a set of polytopes (point, line segment, triangle, tetrahedron, convex polygons, and convex polyhedrons).

The software provides a good interface to define "response functions" that are called when contact happens. In these response functions, contact forces and torques could be programmed. The disadvantage of this interface is that the integrator does not have information about occurred collisions and reduces the step-size around a collision only due to the sharp changes in contact response. Experiments showed that it is difficult to get a robust solution. In general, the integrator may stop for a corrector failure. The reason is that integrators require that the equations describing the system are continuous with a smooth first or higher derivative. At a contact point, these assumptions are not fulfilled and since the changes in the contact forces and torques are so drastic, it is understandable that an integrator may fail. This is also the reason why slight changes in the tolerances of the integrator or the tolerances of the contact detection may change the simulation time very significantly.

For this reason, another interface of SOLID is used to explicitly compute either the distance of two objects or the penetration depth of two colliding objects. This allows to compute indicator functions for the root finder of an integrator, in order that an event

is generated when contact occurs. The solution is more robust and usually more efficient than the solution with "response functions".

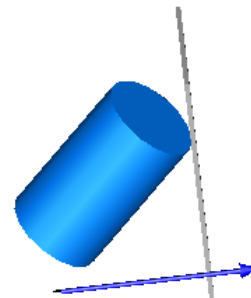
The main disadvantage is that the calling environment has to perform all distance function calls by its own. In the current implementation a brute force method is used by computing the distances between all defined objects. The "broad phase" present in the "response function" interface of SOLID to reduce the number of distance tests significantly (based on "axis-aligned-bounding-boxes" approximations of the objects), is not present with the chosen "root-finding" approach. This will be improved in the future.

The SOLID package uses a generalized version of the GJK algorithm [7] to compute the distances between convex polytopes in a finite number of steps and for other convex surfaces converges globally with a fast convergence rate. For the penetration depth calculation an algorithm is used that is based on similar principles as the GJK algorithm. Details are described in the book [2].

In order that the two algorithms can be applied, a "support mapping" of the corresponding surface is needed. This is a function s_A that maps a vector \mathbf{n} to a point on a convex shape A according to:

$s_A(\mathbf{n})$ returns a point on the surface of A
such that " $\mathbf{n} \cdot s_A(\mathbf{n}) = \max(\mathbf{n} \cdot \mathbf{r}$ for all \mathbf{r} in A)"

This definition is visualized in the next figure for a cylinder:



The arrow in this figure is vector " \mathbf{n} " of the support mapping. This vector is proposed and changed by the distance and penetration depth algorithm. The "grey" shape in the figure above is a plane that is perpendicular to " \mathbf{n} " and is moved to the cylinder such that the plane touches the cylinder. The support mapping function has to return the coordinates of this touching point. If this is not unique, one of the points is returned. For a smooth surface this just means that a point \mathbf{r} on the surface is defined as a function of its normal \mathbf{n} : $\mathbf{r} = \mathbf{r}(\mathbf{n})$.

Due to this simple basic definition of a convex object via a support mapping, a user can introduce additional base shapes in a simple way.

The "penetration depth" algorithm adds in every iteration an additional edge to a simplex that defines the penetration volume. From this simplex, two points are selected in such a way that both points are on the surface of the respective shape and the distance between these two points is as small as possible. When moving the two collided objects along the connection line of these two points, until the two points coincide, then the two shapes are in touching contact. These two points are reported as result of the penetration depth calculation. The penetration depth is then the distance between these two points and the contact normal is on the connection line along these two points.

The SOLID interface functions are used in the World.collisionHandler component that has the following basic structure:

```
equation
  // Compute signed distances
  (signedDistance, ...) =
    surfaceDistances(...);

  // Generate event when distance is zero
  for i in 1:nContactPairs loop
    contact[i] = signedDistance[i] < 0.0;
  end for;

  // Contact response calculation
  (frame_a_f, ...) =
    contactForces(contact, ...);
```

Function “**surfaceDistances(...)**” returns vector “signedDistance”. An element of this vector signals the shortest distance of two objects that are not yet in contact by a positive value. A negative element characterizes the penetration depth of two objects that are in contact. The for loop in the code fragment above triggers events whenever two objects get in contact and whenever two objects are separating. Finally, the function “**contactForces(...)**” is used to perform the response calculation. It returns the resultant forces and torques acting at appropriate reference frames of the corresponding surfaces.

4.5 Response calculation

Contact forces and torques are applied when the relative distance along the normal vector is negative, signaling an interpenetration. As already shortly discussed in section 2, two different response calculations are provided: The first one uses simple spring/damper elements. The second one discretizes the contact area and a resultant response force and torque is computed by summation of appropriate forces over the contact area. This more precise calculation is currently under development. In the following, the first option is discussed in some more detail:

The response is computed according to the following equations:

$$f_n = \min(c_m \cdot s + d_m \cdot v_n, 0)$$

$$f_t = \mu_m \cdot |f_n| \cdot \begin{cases} 1 & \text{if } |v_t| > v_{min,m} \\ |v_t| / v_{min,m} & \text{else} \end{cases}$$

$$\tau_n = d_{w,m} \cdot |f_n| \cdot \begin{cases} -1 & \text{if } \omega_n < -\omega_{min,m} \\ 1 & \text{if } \omega_n > \omega_{min,m} \\ \omega_n / \omega_{min,m} & \text{else} \end{cases}$$

$$\mathbf{f} = f_t \cdot \mathbf{e}_t + f_n \cdot \mathbf{e}_n$$

$$\boldsymbol{\tau} = \tau_n \cdot \mathbf{e}_n$$

where

f_n	contact force in normal direction
f_t	contact force in tangential direction
τ_n	contact torque in normal direction
\mathbf{f}	resultant contact force
\mathbf{t}	resultant contact torque
s	penetration depth (≤ 0)
v_n	relative velocity in normal direction
v_t	relative velocity in tangential direction
ω_n	relative angular velocity in normal direction
\mathbf{e}_n	unit vector in normal direction
\mathbf{e}_t	unit vector in tangential direction

In other words, a linear spring/damper element is used to compute the force in contact normal direction. In tangential direction a sliding friction force is taken into account, if the tangential velocity is larger as v_{min} . Below v_{min} , the friction force is reduced so that it is zero, when the tangential velocity becomes zero. Sticking is currently not implemented. For gripping operations, it is important to take into account the friction torque. This is accomplished by a linear rotational damper that is proportional to the normal force and the relative angular velocity. Finally, all force and torque parts are summed up resulting in the contact force and torque. Note, if the normal force would become positive since the damping part is too large, it is reduced to zero, since a positive normal force is physically not correct.

For the equations above material constants are needed, e.g., for the spring and the dampers. However, only material data for the respective surfaces are provided. The correct solution would be to apply, say, (1) a spring/damper element on surface A using the material constants of surface A, (2) a spring/damper element on surface B using the material constants of surface B and (3) connect the spring/damper elements of surfaces A and B in series. Due to the linear dampers, this would result in

additional differential equations depending on the number of contact points. To avoid complications and to enhance efficiency, the following approximation is used: A resultant spring constant is computed from the surface data under the assumption of a series connections of two springs. For all other data, mean values are used:

$$c_m = \frac{c_A \cdot c_B}{c_A + c_B}$$

$$d_m = \frac{1}{2} \cdot (d_A + d_B)$$

$$d_{w,m} = \frac{1}{2} \cdot (d_{w,A} + d_{w,B})$$

$$\mu_m = \frac{1}{2} \cdot (\mu_A + \mu_B)$$

$$v_{min,m} = \frac{1}{2} \cdot (v_{min,A} + v_{min,B})$$

$$\omega_{min,m} = \frac{1}{2} \cdot (\omega_{min,A} + \omega_{min,B})$$

5 Outlook

An overview was given, in which way the Modelica MultiBody library is extended with collision handling. The current stage is already useful for applications. Development continues to improve the collision handling:

- Using the “broad-phase” of SOLID to reduce the number of collision tests significantly.
- Support more than one contact point between two surfaces. This is important for non-convex surfaces.
- Optionally, provide a more detailed response calculation by discretization of the contact area.
- Reduce the limitations of “*.dxf” files.

Acknowledgements

Partial financial support of this work by the Toyota Motor Corporation is gratefully acknowledged.

References

- [1] Andreasson J. (2003): **Vehicle Dynamics library**. Proceedings of Modelica’2003, ed. P. Fritzson, pp. 11-18. Download: http://www.Modelica.org/-Conference2003/papers.shtml/h28_vehicle_Andreasson.pdf
- [2] Bergen G. van den (2004): **Collision Detection in Interactive 3D Environments**. Elsevier and Morgan Kaufmann Publishers.
- [3] Bergen G. van den (2003): **SOLID 3.5 - User’s Guide to the SOLID Collision Detection Library**. On the CD of the book of G. van den Bergen [2].
- [4] Dynasim (2005): **Dymola – Users Manual** (<http://www.dynasim.com>)
- [5] Eberly D.H., and Shoemake K. (2004): **Game Physics**. Elsevier and Morgan Kaufmann Publishers.
- [6] Ehmman S. H. (2000): **SWIFT - Speedy Walking via Improved Feature Testing**. Version 1.0. Download: <http://www.cs.unc.edu/~geom/SWIFT/>
- [7] Gilbert E.G., Johnson D.W., and Keerthi S.S. (1988): **A fast procedure for computing the distance between complex objects in three-dimensional space**. IEEE Journal of Robotics and Automation, 4(2), pp. 193-203.
- [8] Hasegawa S., and Sato M. (2004): **Real-time Rigid Body Simulation for Haptic Interactions Based on Contact Volume of Polygonal Objects**. EUROGRAPHICS 2004, ed. M.-P. Cani and M. Slater, Volume 23, Number 3. Download: <http://eg04.inrialpes.fr/Programme/Papers/PDF/paper1209.pdf>
- [9] Hippmann G. (2003): **An Algorithm for Compliant Contact between complexly shaped Surfaces in Multibody Dynamics**. Multibody Dynamics, Jorge A.C. Ambrósio (Ed.), IDMEC/IST, Lisbon, Portugal, July 14. Download: http://www.pcm.hippmann.org/-doc/eccomas03_hippmann.pdf
- [10] Leine R.I., and Glocker C. (2003): **A set-valued force law for spatial Coulomb–Contensou friction**. European Journal of Mechanics A/Solids 22, pp. 193–216.
- [11] Mattsson S.E., Olsson H., and Elmqvist H. (2001): **Methods and Algorithms for Varying Structure Hybrid DAE Simulation**. EC IST Project RealSim under contract IST-1999-11979, Internal Report 2.2.
- [12] H. Olsson, H. Tummeseit, H. Elmqvist (2005): **Modeling with Partial Derivatives of Modelica Functions and Automatic Differentiation**. Modelica’2005 conference, Hamburg, March 7-8.
- [13] Otter M., Elmqvist H., Mattsson S.E. (2004): **The New Modelica MultiBody Library**. Proceedings of Modelica’2003, ed. P. Fritzson, pp. 311-330. Download: http://www.Modelica.org/Conference-2003/papers.shtml/h37_Otter_multibody.pdf
- [14] Pacejka H.B. (2002): **Tyre and Vehicle Dynamics**. Butterworth-Heinemann.
- [15] Pfeiffer F., and Glocker C. (1996): **Multibody Dynamics with Unilateral Contacts**. John Wiley & Sons.
- [16] Smith R. (2004): **Open Dynamics Engine – V0.5 Users Guide**. Download from <http://ode.org>.
- [17] Wriggers P. (2002): **Computational Contact Mechanics**. John Wiley & Sons.

Session 1b

Chemical Systems and Thermodynamic Systems I

The Modelica Bond Graph Library

François E. Cellier
 Institute of Computational Science
 Swiss Federal Institute of Technology
 ETH-Zentrum HRS H28
 CH-8092 Zürich
 Switzerland
FCellier@Inf.ETHZ.CH

Àngela Nebot
 Llenguatges i Sistemes Informàtics
 Universitat Politècnica de Catalunya
 Jordi Girona Salgado, 1-3
 Barcelona 08034
 Spain
Angela@LSI.UPC.ES

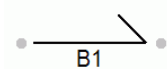
Abstract

Bond graphs offer a domain-neutral graphical technique for representing power flows in a physical system. They are particularly powerful for representing systems that operate in multiple energy domains, such as thermal models of electronic circuits, mechanical vibrations in acoustic systems, etc. A bond graph library was created for Modelica with graphical Dymola support. The library is presented in this paper. Applications from different domains are offered to document its use.

Keywords: bond graph; energy modeling; thermodynamic modeling; Biosphere 2

1 Introduction

A bond represents the flow of power, P , from one point of a physical system to another.

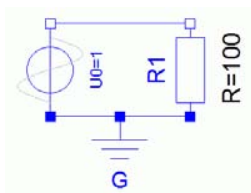


It is represented by a harpoon. There are two physical variables associated with each bond, an effort, e , and a flow, f . The product of these two variables represents the power:

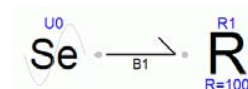
$$P = e \cdot f$$

In an electrical circuit, the effort variable is identified with the voltage, u , whereas the flow variable is identified with the current, i .

In the electrical circuit:



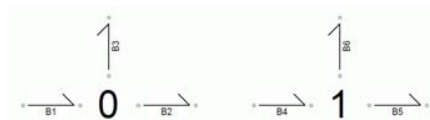
electrical power is being delivered from a voltage source to a resistor. The corresponding bond graph is:



where a source of effort, U_0 , delivers the electrical power, $P_{el} = U_0 \cdot i$, to the resistor, R_1 .

In our implementation, a third variable is also associated with each bond, a directional variable, d . This variable indicates the direction of positive power flow. It is encoded by setting $d = -1$ at the emanating bondgraphic connector and $d = +1$ at the receiving connector. The directional information is used in the computations associated with *junctions*.

Bond graphs offer two types of junctions, the 0-junction, and the 1-junction:



In a 0-junction, the efforts are set equal, whereas the flows add up to zero:

$$e[2:n] = e[1:n-1]$$

$$d' \cdot f = 0$$

In a 1-junction, the flows are set equal, whereas the efforts add up to zero.

$$f[2:n] = f[1:n-1]$$

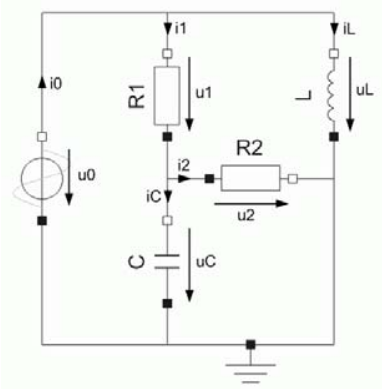
$$d' \cdot e = 0$$

Thus, the two junction types are duals of each other.

In an electrical circuit, the 0-junctions correspond to nodes, whereas the 1-junctions correspond to meshes. We are now able to translate the circuit diagram of an arbitrary electrical circuit into a corresponding bond graph.

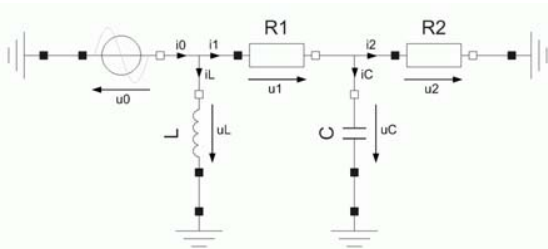
2 Bond Graphs of Electrical Circuits

Given the electrical circuit:

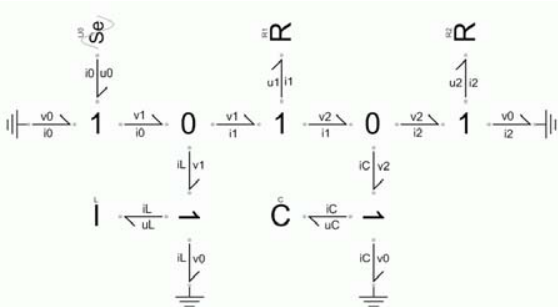


we can represent each node, except for the ground node, by a 0-junction, and we can represent each circuit element connecting two nodes by a 1-junction, from which the circuit element is suspended. The directions of positive power flows are chosen to coincide with the directions of positive current flow in the circuit diagram.

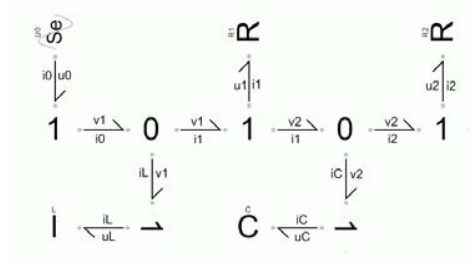
For simplicity, let us first redraw the circuit diagram with the ground node split into multiple separate nodes.



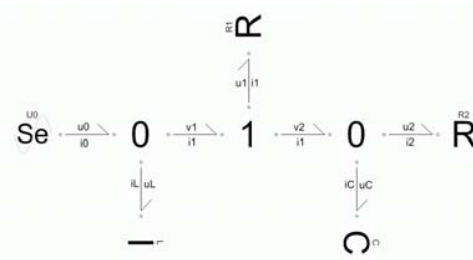
This circuit diagram can be translated directly into a corresponding bond graph:



Since the ground potential, v_0 , is equal to zero, the bonds connecting to the ground don't carry any power. They can thus be eliminated.



Finally, junctions with only two bonds attached to them can be amalgamated away. Thus, the final bond graph can be drawn as follows:



3 Causal Bond Graphs

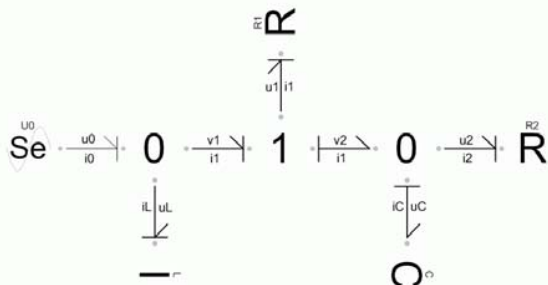
Since there are two physical variables associated with each bond, we need two equations to compute their values. It turns out that each end of the bond computes one of the two variables. We can mark the side that computes the flow variable by a short bar: the *causality stroke*.

The Modelica Bond Graph Library offers causal and a-causal bonds. Whereas the a-causal bonds were implemented as Modelica *models*, the causal bonds were implemented as *blocks*.

We recommend using causal bonds as much as possible. The causalities associated with bonds attached to sources are fixed. Since an effort source computes the effort, the causality stroke of its bond must be away from the source. Since 0-junctions are characterized by a single flow equation, there must be exactly one causality stroke at a 0-junction. Since 1-junctions are characterized by a single effort equation, there are exactly $n-1$ causality strokes at a 1-junction.

Capacitors and inductances have preferred causalities. Since we like to end up with differential equations (integral causality), capacitors like to compute the effort, whereas inductors prefer to compute the flow. Thus the preferred position for causality strokes of bonds attached to capacitors is away from the capacitor, whereas the preferred position of causality strokes of bonds attached to inductors is at the inductor. The causalities of resistive elements are free.

In the case of the given circuit, the preferred causality of all bonds is fixed. The causal bond graph can be presented as follows:



We are now capable of reading out the causal equations from the bond graph. These are:

$$\begin{aligned}
 u_0 &= f(t) \\
 i_0 &= i_L + i_1 \\
 u_L &= u_0 \\
 di_L/dt &= u_L / L \\
 v_1 &= u_0 \\
 u_1 &= v_1 - v_2 \\
 i_1 &= u_1 / R_1 \\
 v_2 &= u_C \\
 i_C &= i_1 - i_2 \\
 du_C/dt &= i_C / C \\
 u_2 &= u_C \\
 i_2 &= u_2 / R_2
 \end{aligned}$$

There is no advantage of using an a-causal bond graph instead of a circuit diagram when modeling an electrical circuit. The two representations are totally equivalent to each other. However, there is a certain advantage of using a causal bond graph, since the equations describing the circuit can be read out of the causal bond graph directly in their causal form.

Of course, there is no need to ever use causal bonds in Modelica, as Modelica is perfectly capable of determining the computational causality of all equations on its own. Yet, we recommend using causal bonds as much as possible, as they help the modeler in analyzing his or her model.

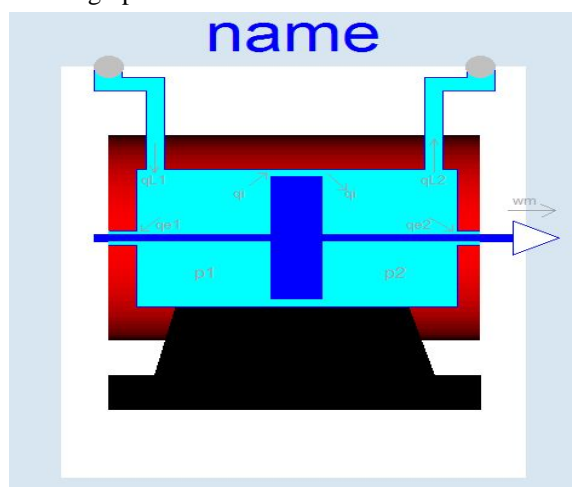
4 Algebraic Loops and Structural Singularities

When the mandated and preferred causalities of all elements do not lead to a single assignment of all causality strokes, the model contains one or several *algebraic loops*. A-causal bonds must be used whenever the causality assignment is free.

On the other hand, if not all preferred causalities can be satisfied, i.e., when the causality stroke of a bond attached to either a capacitor or an inductor is located at the incorrect end of the bond, the model contains a structural singularity, i.e., consists of a higher-index DAE system. Also in that case, a-causal bonds should be used to give Modelica a chance to reducing the perturbation index on its own.

5 A Hydraulic Motor Control System

We wish to model the following hydraulic motor by a bond graph:



In hydraulic bond graphs, it is customary to identify the pressure, p , with the effort variable, whereas the volumetric flow rate, q , is identified with the flow variable. The product of pressure and volumetric flow is the hydraulic power.

Due to the compressibility of the liquid, the change of pressure in each chamber is proportional to the difference between inflow and outflow. In terms of a bond graph, this looks like a capacitor attached to a 0-junction.

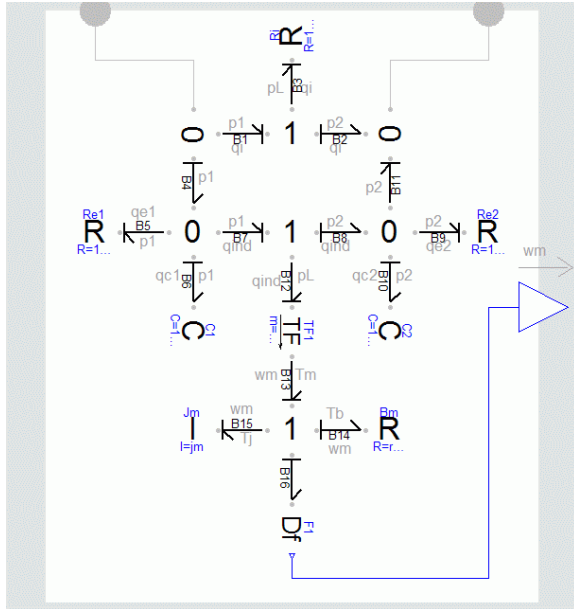
The flows q_i , q_{e1} , and q_{e2} are laminar leakage flows. They are proportional to the pressure difference. Thus, they can be represented as linear resistors.

On the mechanical side, power can be written as either force times velocity or torque times angular velocity. Among bond graph practitioners, it has become customary to identify the forces and torques with effort variables, and the velocities and angular velocities with flow variables.

Newton's law states that the change in velocity (or angular velocity) is proportional to the sum of all forces (or torques). In terms of a bond graph, this looks like an inductor attached to a 1-junction.

The two domains are coupled by a transformer, as the force on the piston (or the torque on the screw, depending on the geometry of the motor) is proportional to the difference between the pressures in the two chambers.

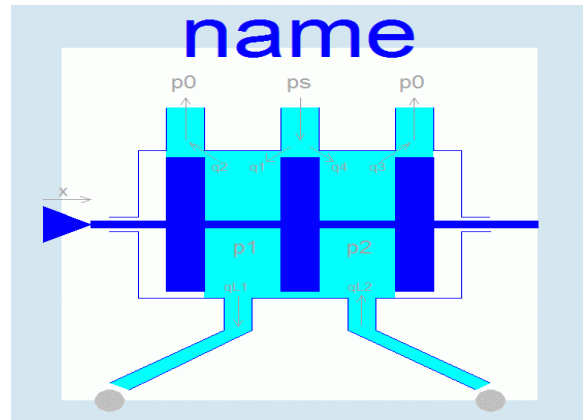
We are now ready to draw the bond graph of the hydraulic motor:



The two 0-junctions to the left and to the right represent the two hydraulic chambers with the pressures p_1 and p_2 , respectively. Each of them has been pulled apart into two separate 0-junctions connected by a bond for graphical reasons. Same sex junctions neighboring each other can always be considered as a single junction. The two capacitors symbolize the compressibility of the liquid. The three resistors at the top half of the bond graph represent the leakage flows, one of which, q_i , is an internal leakage flow, whereas the others, q_{e1} and q_{e2} , are external leakage flows.

The transformer, TF , separates the hydraulic from the mechanical side. The inductor represents the inertia of the (rotational) screw, whereas the resistor represents the friction of the screw. The flow detector element, Df , detects the angular velocity, ω_m , of the screw. It converts the bond graph representation to a signal.

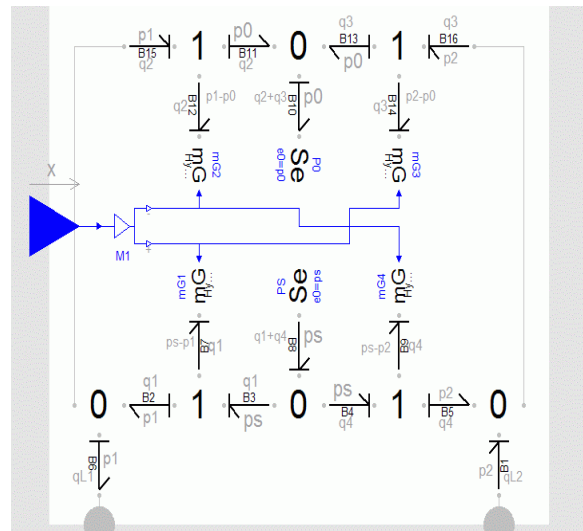
The hydraulic motor is controlled by a servo valve:



The inflow pressure, p_s , is the load pressure of the hydraulic motor. The outflow pressure, p_0 , is the ambient pressure of the environment. The four valve flows, q_1 , q_2 , q_3 , and q_4 , are turbulent flows. Hence they are proportional to the square root of the pressure difference. In terms of a bond graph, they can be represented either as nonlinear resistors (R -elements) or as nonlinear conductors (G -elements). Since the causalities are those of a conductive element, we chose the latter representation to prevent Modelica from having to turn these nonlinear equations around symbolically.

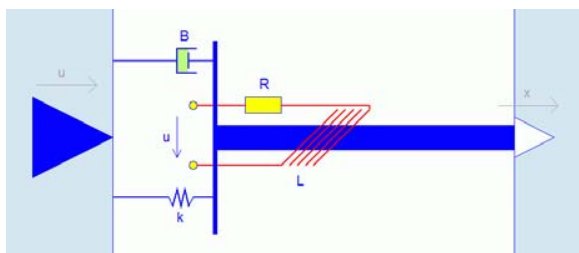
All four valve flows are modulated by the position of the tongue, x .

We are now ready to draw the bond graph of the servo valve:



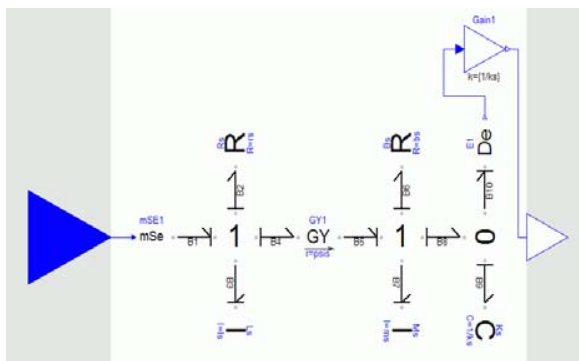
The tongue position, x , is an input signal. It influences the bond graph by means of modulation of the four hydraulic conductance elements.

We still need to model the motion of the tongue of the servo valve:



The tongue of the servo valve is an electromechanical converter. The input signal, u , modulates an effort source that generates a magnetic field in a coil. The magnetic field induces a mechanical force in the tongue that is proportional to the current through the coil. Thus, the converter can be modeled as a bond-graphic gyrator, GY . Whereas a *transformer* sets the output effort proportional to the input effort (and the input flow proportional to the output flow), the *gyrator* sets the output effort proportional to the input flow (and the input effort proportional to the output flow).

We are now ready to draw the bond graph of the device:



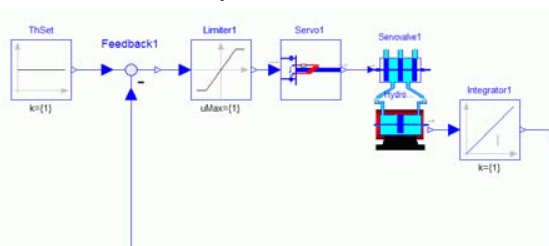
The modulated effort source translates the signal, u , to a power flow. The 1-junction to the left of the gyrator symbolizes the electrical mesh. The inductor here represents the coil, whereas the resistor represents the electrical resistance.

The gyrator converts the electrical power, $P_{el} = u_i \cdot i$, where u_i is the induced voltage, to mechanical power, $P_{mech} = f \cdot v$. The 1-junction to the right of the gyrator symbolizes the velocity of the tongue. The inductor here represents the mass of the tongue, the resistor represents the mechanical damper, and the capacitor represents the mechanical spring.

We could have attached a flow detector, Df , to the 1-junction to detect the velocity of the tongue. We could then have integrated the resulting signal to obtain the tongue position, x . Yet, we chose another route. The tongue position, x , is proportional to the spring force. Thus, we can use an effort detector, De , to detect the spring force. However, an effort detector needs to be attached to a 0-junction. To this

end, an additional 0-junction was placed between the 1-junction and the capacitor.

We are now ready to model the control circuit:



From the outside, the control circuit looks like a regular block diagram. However, three of the blocks have been modeled by bond graphs internally.

6 The Thermal Budget of Biosphere 2 without Air-conditioning

As a second example, we shall model the thermal behavior of Biosphere 2, an experimental research facility located in the vicinity of Tucson, Arizona, without air-conditioning by means of bond graphs.

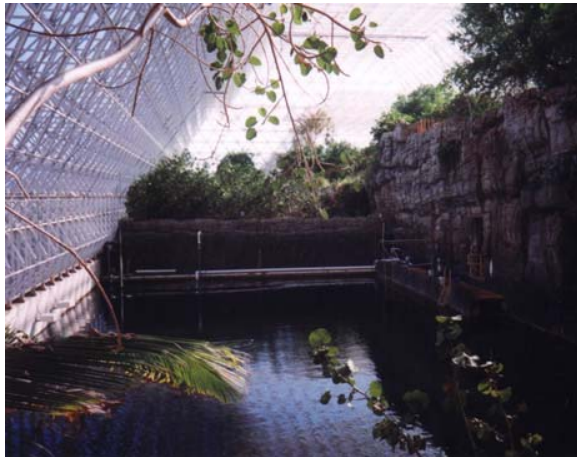
Rather than using a bottom-up approach, as we did in the previous example, we shall this time around use a top-down approach.

Biosphere 2 was designed as a materially closed structure to investigate the ability of humans to survive in a materially closed structure for extended periods of time. The main idea was to investigate whether space colonies are feasible with today's technologies.



Biosphere 2 was constructed as a large glass building on 3 acres (12.000 m²). The structure is held together by a metallic frame construction and is closed off by glass panels.

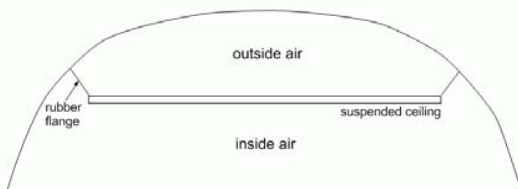
The structure contains a number of different biomes. The pyramidal structure to the right contains a tropical rain forest. The elongated structure to the left contains a pond, a savannah, saltwater marshes with mangroves, and a southwestern desert landscape.



The air pressure inside Biosphere 2 is kept constant by two lungs, one of which is shown below:

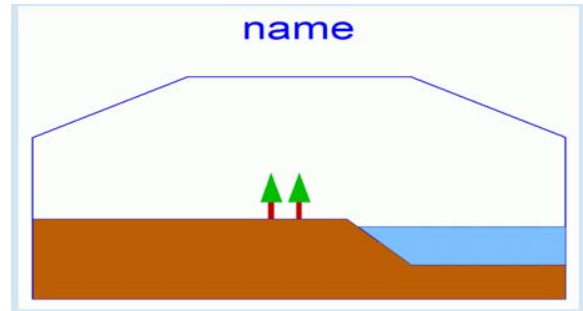


The lungs operate as follows. A heavy cement ceiling is suspended from the dome by a rubber flange. The bottom part of the lung is inside the materially closed structure, whereas the top part is outside air.



When the temperature inside Biosphere 2 rises, the inside pressure increases as well. Consequently, the ceilings in the two lungs are lifted up, thereby increasing the total volume of Biosphere 2. In this way, the inside pressure remains the same as the ambient pressure irrespective of the temperature.

Although air-conditioning keeps the temperature and humidity values different in the different biomes, Biosphere 2 was modeled by us as a single structure with one inside temperature and humidity.



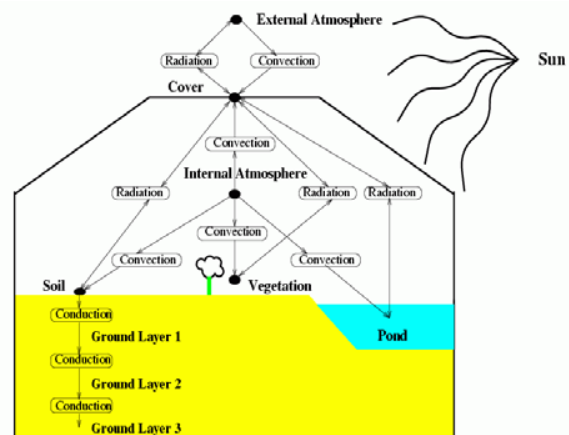
There are a number of different elements in that model: the inside air, the dome, the pond, the vegetation, and the soil, each of which are allowed to be at a different temperature. Only the inside air also contains humidity.

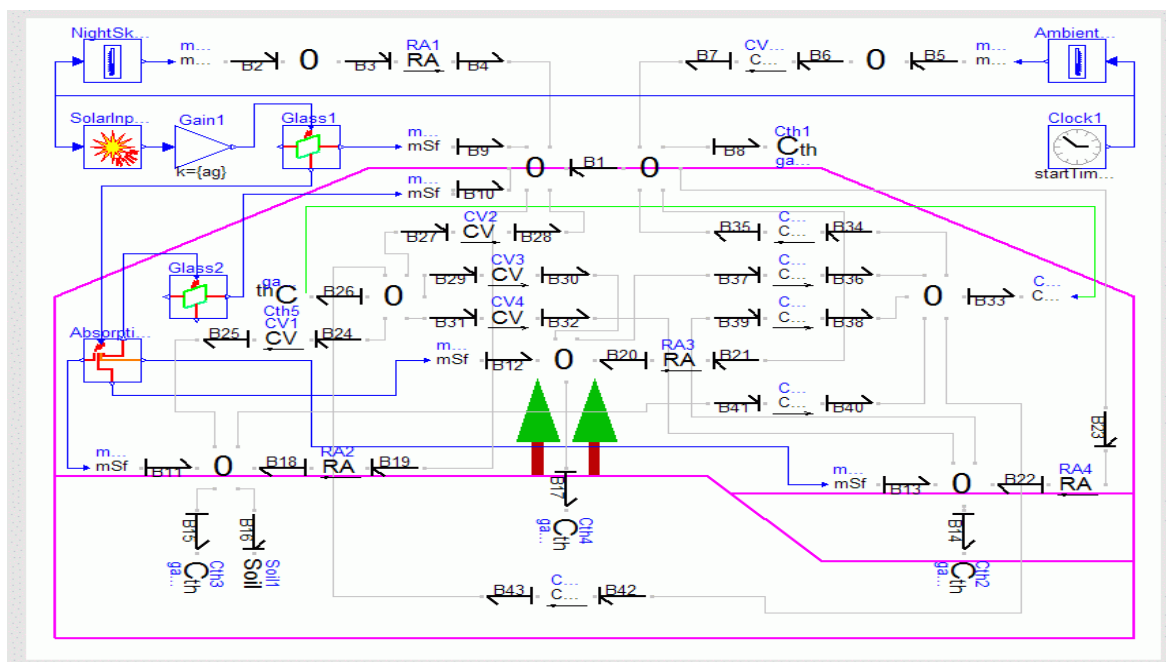
In a bond graph, thermal power, i.e., heat flow, can be written as the product of temperature and entropy flow. It is customary to identify the temperature with the effort variable, and the entropy flow with the flow variable.

Each of the five elements is represented as a 0-junction with a (non-linear) capacitor attached to represent the heat capacity of the element. Heat flows between the elements are represented as non-linear resistors modeling physical effects such as convection and radiation.

The inside air is represented by two separate 0-junctions, one modeling the temperature of the air, the other modeling its humidity. Non-linear resistors between the thermal and humidity junctions are used to model the effects of evaporation (conversion of sensible heat to latent heat) and condensation (conversion of latent heat to sensible heat).

A conceptual model of Biosphere 2 is shown below:





Each black dot represents a modeling element, i.e., a 0-junction with a heat capacity attached to it. The flows between these modeling elements are represented by two-port elements modeling the effects of conduction, convection, radiation, evaporation, and condensation.

The model starts in the upper right corner with the simulation clock. The ambient temperature and the apparent temperature of the night sky are computed by tabular look-up functions.

The temperature values are then converted to power flows by the use of modulated effort sources. Temperature sources are physically dubious, but it is okay to use them here, since the model doesn't contain any physical explanation as to how the environment reaches its temperature. The temperature values are simply being observed.

Since the dome is in physical contact with the outside air, convection takes place across the dome. Furthermore, the dome is exposed to diffuse radiation from and to the sky.

The 0-junction representing the dome was split into two separate 0-junctions connected by a bond for graphical reasons. The thermal capacitor attached to the 0-junction computes the temperature of the dome.

Biosphere 2 is also exposed to direct solar radiation. The *Solar Input* model, symbolized by the sun, computes the position of the sun in the sky, and thereby computes the total amount of direct solar radiation input reaching the Biosphere 2 structure.

Since the different glass panels have many different orientations, it would have been computation intensive to calculate accurately the amount of radiation that gets transmitted, absorbed, and reflected by each of the glass panels. Thus, a much more global approach was taken. It is assumed that roughly 60% of the solar input gets transmitted across the glass panels, 20% gets absorbed by them, whereas the final 20% get reflected to the outside.

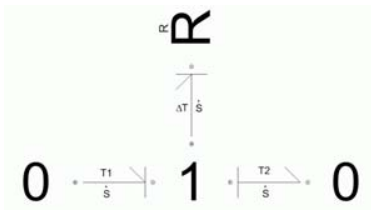
The *Glass₁*, *Absorption*, and *Glass₂* models railroad the available solar input to the individual elements, where they arrive in the form of flow (entropy) sources. The vegetation, soil, and inside air absorb all of the arriving solar input. The pond absorbs some of it, and reflects the rest. The reflected solar input is partly absorbed by the inside air, and partly reaches the dome again from the inside, where it is partly absorbed, and partly transmitted back out.

Thus, a global balance approach was used to model the direct solar input. The end effect is that each of the 0-junctions representing the five different modeling elements has a modulated flow source attached to it that models the amount of direct solar input absorbed by that element.

7 Convection

Let us now look at the processes of convection between modeling elements. Since the air-conditioning was left out of the model, there are no forced flows. Thus, the convection is simply driven by temperature

differences, i.e., by potential equilibration. This is a resistive phenomenon.

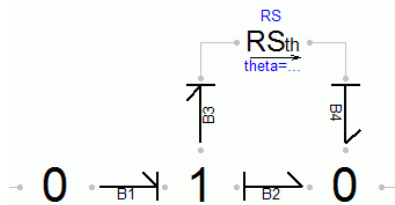


The two 0-junctions symbolize the two modeling elements that exchange heat among each other. They are at the temperature values, T_1 and T_2 , respectively.

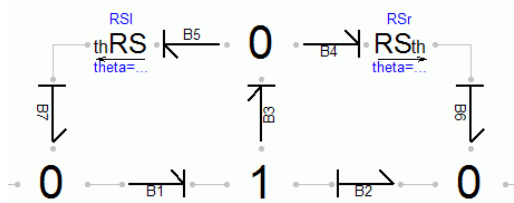
The 1-junction between them computes the temperature difference, ΔT , which drives the entropy flow.

The problem with this model should become evident at once. What happens with the power flow into the resistor? It may make sense to model with resistors in an electrical circuit, because we may not care about the entropy that is being generated by the resistor. However here, we are operating already in the thermal domain. Additional entropy is being generated by the resistor, and this entropy needs to be routed somewhere.

It has become customary to replace thermal resistors by resistive source elements, RS , and route the generated entropy to the nearest 0-junction. The so modified bond graph is shown below:

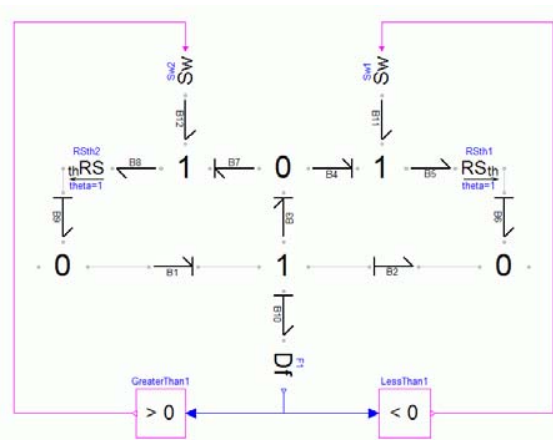


As convection is a symmetric phenomenon, we could alternatively route half of the generated entropy flow to the right and the other half to the left:



Finally, we may choose to route the generated entropy down-wind, i.e., if $T_1 > T_2$, all of the generated entropy flow is routed to the right, otherwise to the left.

To this end, we shall require a flow detector and two switch elements:



The bondgraphic switch element, Sw , has a Boolean input. If that input has a value of *true*, the switch is *open*, i.e., there is zero flow. In that case, the causality stroke is at the switch element. On the other hand, if the Boolean input has a value of *false*, the switch is *closed*, and in that case, there is zero effort. Thus by now, the causality stroke has moved away from the switch. Hence a-causal bonds must be used at the switches.

Since the 1-junctions must have $n-1$ causality strokes, another bond must also change its causality. This has to be the bond that leads to the resistive source element, RS .

8 Conclusions

In this paper, a bond graph library has been introduced that was designed to be used with Dymola. Since bond graphs are a graphical modeling tool, it may be much less desirable to use this library with Modelica alone, i.e., in an environment that is based on an alphanumerical representation of models.

This is already the second presentation of the Modelica Bond Graph Library. An earlier paper [4] had been prepared for a conference on bond graph modeling. Thus, whereas the earlier paper had been prepared for an audience that knew a lot about bond graphs, but little if anything about Modelica and/or Dymola, the current paper was written for an audience that is expected to be knowledgeable about Modelica and Dymola, but probably knows little if anything about bond graphs.

An earlier presentation of the Biosphere 2 model was published in [5]. The model presented in that paper had been developed using a much earlier version of Dymola, prior to the design of Modelica. At

that time, a strictly alphanumeric version of a bond graph library had been used [1].

Bond graphs offer a fairly low-level interface to modeling physical systems. Thus, bond graphs should be used hierarchically in the context of complex systems [2]. The Biosphere 2 model demonstrates how bond graphs can be hierarchically structured. The hydraulic motor example demonstrates how bond graphs can be hidden inside other modeling metaphors, such as block diagrams.

The primary strength of bond graphs is their domain independence. For this reason, bond graphs are particularly suitable for the description of physical systems that operate in multiple energy domains. Energy conversions can be modeled easily and conveniently using transformers and gyrators.

As with any other modeling paradigm, there is nothing unique about bond graphs. Every single one of our models could have been developed using other modeling paradigms as well. Modeling paradigms offer a means for modelers to organize their knowledge about the physical systems they wish to describe. Some researchers will find bond graphs a convenient way to organize their knowledge, whereas other researchers won't. To us, bond graphs have become the ultimate tool for understanding the basic principles covering all of physics [3].

References

- [1] Cellier, F.E. (1991), *Continuous System Modeling*, Springer-Verlag, New York, ISBN: 0-387-97502-0, 755p.
- [2] Cellier, F.E. (1992), "Hierarchical Non-Linear Bond Graphs: A Unified Methodology for Modeling Complex Physical Systems," *Simulation*, **58**(4), pp. 230-248.
- [3] Cellier, F.E. (1995), "Bond Graphs: The Right Choice for Educating Students in Modeling Continuous-Time Physical Systems," *Simulation*, **64**(3), pp. 154-159.
- [4] Cellier, F.E. and R.T. McBride (2003), "Object-oriented Modeling of Complex Physical Systems Using the Dymola Bond-graph Library," *Proc. ICBGM'03, 6th SCS Intl. Conf. on Bond Graph Modeling and Simulation*, Orlando, Florida, pp. 157-162.
- [5] Nebot, A., F.E. Cellier, and F. Mugica (1999), "Simulation of Heat and Humidity Budget of Biosphere 2 without its Air Conditioning," *Ecological Engineering*, **13**, pp. 333-356.

Biographies



François E. Cellier received his B.S. degree in electrical engineering from the Swiss Federal Institute of Technology (ETH) Zürich in 1972, his M.S. degree in automatic control in 1973, and his Ph.D. degree in technical sciences in 1979, all from the same university. Dr. Cellier joined the University of Arizona

in 1984 as associate professor, where he is currently a full tenured professor of Electrical and Computer Engineering. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 with Springer-Verlag, New York. He served as general chair or program chair of many international conferences, and serves currently as president of the Society for Modeling and Simulation International.



Àngela Nebot received licentiate and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1988 and 1994, respectively. From 1988 until 1992 she was with the Institut de Ciències de la Computació, holding a pre-doctoral research grant from

the government of Catalunya, and completing the doctoral courses in software and artificial intelligence. She joined the department of Llenguatges i Sistemes Informàtics in 1994 as an assistant professor, and since March 1998, she has been associate professor in the same department. She currently belongs to the soft-computing group of the UPC. She is an associate editor of the journal *Simulation: Transactions of the Society for Modeling and Simulation International*, and serves as a reviewer for other international journals such as the *International Journal of General Systems*, *Neurocomputing*, *Artificial Intelligence in Environmental Engineering*, and *Artificial Intelligence Communication*. Her current research interests include fuzzy systems, neuro-fuzzy systems, genetic algorithms, simulation, and e_learning.

Fuel Cell System Modeling for Real-time Simulation

Jörg Ungethüm*

German Aerospace Center, Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart

Abstract

In this paper a model of a subsystem of an automotive fuel cell power generation unit is presented. The subsystem model describes the cathode side of the fuel cell, containing the air supply system. As far as possible, standard libraries were used to accomplish a high level of compatibility with other models. The model runs in real-time on dSPACE hardware and is used in a Hardware-In-the-Loop (HIL) simulation environment.

Keywords: fuel cell, automotive simulation, real-time simulation, hardware-in-the-loop

1 Introduction

In the context of future vehicle development, models are needed for the simulation of the fuel cell system and its periphery. These models are used for off-line system simulation as well as for HIL simulation with respect to controller development. Most of the existing models are built upon MATLAB/Simulink. The model which is presented in this paper serves particularly in the evaluation of object-oriented modeling in Modelica as an alternative to modeling in MATLAB/Simulink. Apart from modeling potentialities, flexibility and simulation performance, the code administration is also relevant in this context.

In figure 1 a sketch of the modeled subsystem is shown, which covers the cathode side of a PEM fuel cell system. The fuel cell is supplied with compressed and cooled air. Due to the electro-chemical reaction in the stack, the rate of oxygen in the air is reduced and at the same time the air takes up the major part of the reaction water. Heat is rejected by means of a cooling medium. The remaining pressure difference between the stack outlet and the environment is used in an exhaust gas turbine. Therefore, and to recover reaction water, liquid water is dragged from the air in a separator. Using the recirculation valve, air can be led back

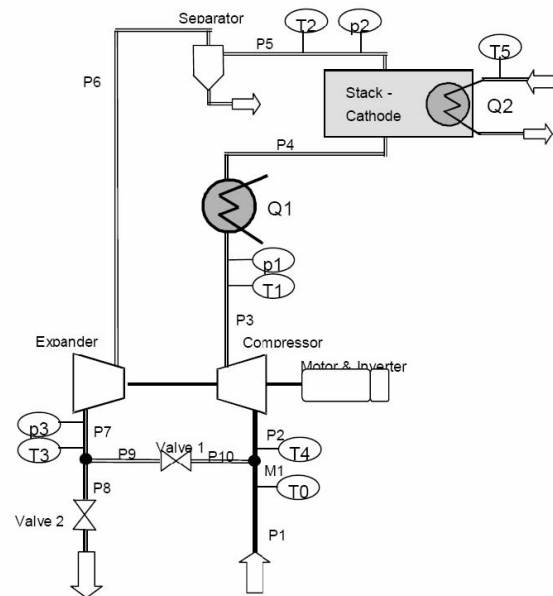


Figure 1: Sketch of the fuel cell subsystem to simulate

directly to the entrance of the compressor. Compressor and exhaust gas turbine are mounted together with an electrical drive engine on a common shaft.

2 Model design

The model covers the process engineering part of the system, whereby the focus has been the description of the thermodynamic behavior. The mechanical part is built up with simple models, the fuel cell stack is realized as thermal inertia and a source of heat and humidity. The model realistically reproduces the most important influences on the dynamic behavior of the system. These are the volumes and thermal capacities of the components and the inertia of the common shaft of compressor, exhaust gas turbine and electrical machine. The top level of the model is shown in figure 2.

*joerg.ungethuem@dlr.de

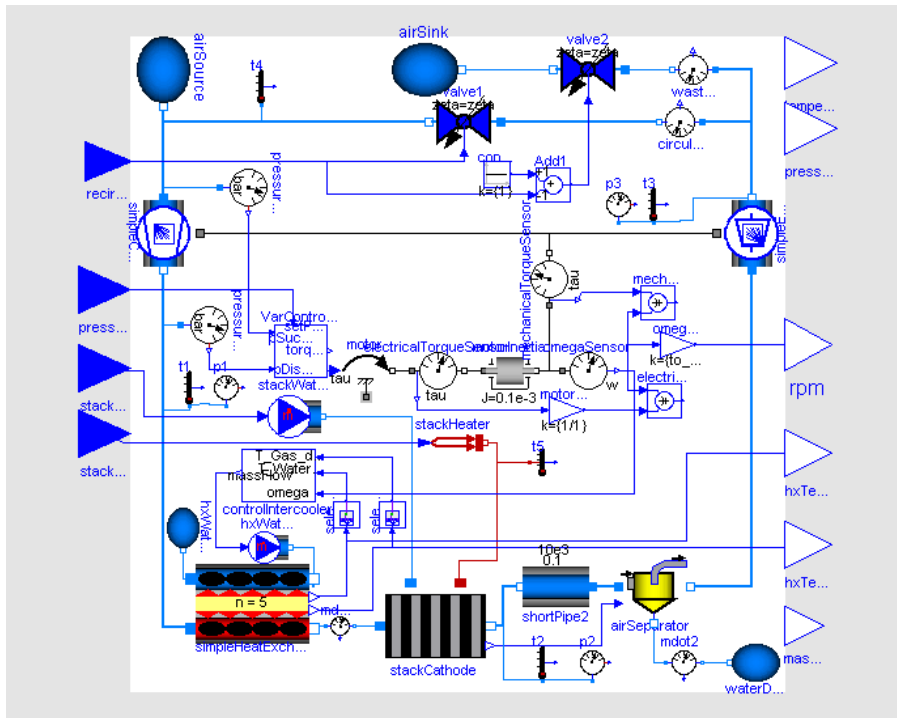


Figure 2: The Modelica system model

3 Aspects of real-time simulation

3.1 Toolchain

As the model is used in context of HIL simulation on a dSPACE system, it has to run in real-time on the target. To enable real-time simulation of potentially stiff models, Dymolas inline integration approach is used [1]. The model is embedded as a S-function into a simple Simulink wrapper model (figure 3) using Dymolas Simulink interface block. The MATLAB Real-Time Workshop and the dSPACE target compiler are needed to compile the compounded Simulink model. Visualization is realized using the dSPACE Control-Desk program.

3.2 Real-time model requirements

The central requirements of real-time modeling are deterministic computing time and high computing speed. In order to provide a deterministic computing time, iterative solution algorithms should be avoided. High computing speed is reached by keeping model equations as simple as possible.

3.2.1 Avoidance of nonlinear sets of equations

Nonlinear sets of equations must be solved by iteration, if they cannot be eliminated symbolically. How-

ever, this leads to bad performance of the simulation. In real-time simulation the situation is even worse, because the iteration can prevent the deterministic solution behavior which is required. On the other hand it is not strictly necessary to avoid any implicit equation. As long as the required number of iterations is moderate, real-time requirements can be fulfilled anyway.

3.2.2 Use of simple medium models

The evaluation of medium properties in thermodynamic models can take up a major part of the computing time. Therefore properties should not be formulated more complicated than absolutely necessary. In particular numeric problematic functions, e.g. logarithm, high polynomial degrees and broken exponents should be avoided. Within the implementation extraordinary care must be put on good performance and numeric stability. Using Horner's scheme instead of the `pow()` function for polynomial evaluation might be mentioned as an example.

3.2.3 Properties and state variables

In most cases, thermodynamic variables of state are also state variables of the model. The medium properties should be present as explicit, fast evaluable functions of the actual set of state variables. Therefore, the choice of the variables of state depends on the used

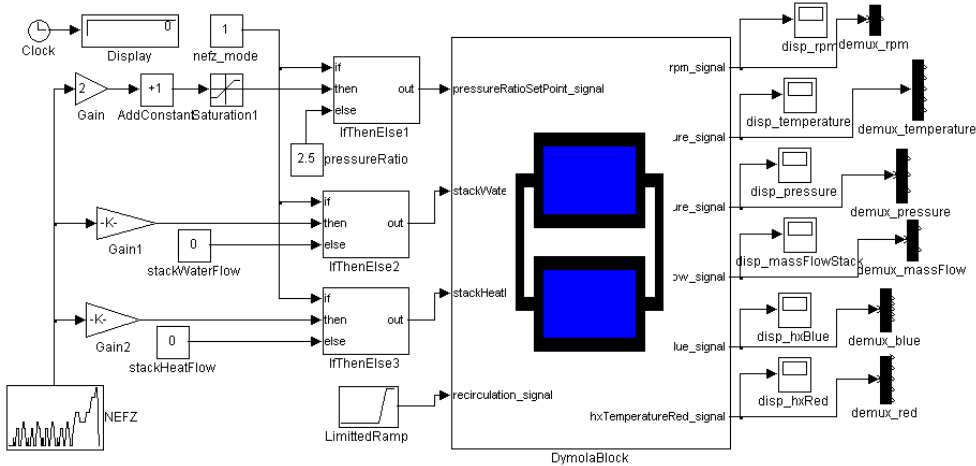


Figure 3: The Simulink wrapper model

states	derivatives
p, ρ	$\frac{\partial p}{\partial h} p, \frac{\partial p}{\partial p} h$
ρ, T	$\frac{\partial u}{\partial p} T, \frac{\partial u}{\partial T} p$

Table 1: Extra 2nd order derivatives needed in transformed balance equations

property model. Most property models are explicit in density ρ and temperature T or enthalpy h and pressure p . As a result the thermodynamic balance equations should be formulated in such a way that either density and temperature or enthalpy and pressure are computed by differential equations. Since in the primary form of mass and energy balance density ρ and internal energy u are calculated, these equations must be transformed. The transformed balance equations contain additional partial differentials of the thermodynamic state variables, which must be computed in the property model (table 1, [7]).

3.2.4 Avoidance of numeric Jacobians

Numerical Jacobian approximation is a common source of instability and inaccuracy. In Dymola numeric Jacobians are used only if the necessary partial derivatives cannot be computed symbolically. There are few cases where symbolic derivative generation is not possible yet:

- external functions (library calls, calls to routines written in C or Fortran)
- Modelica functions with the exception of one-liners (at least up to Dymola 5.2)

However, the necessary derivatives can be provided explicitly by the user [4].

3.2.5 Avoidance of numerically disadvantageous functions

Some mathematical built-in functions are problematic in numeric simulations. The square root function and the logarithm function have limited definition ranges, the derivation of the root function has an additional singularity at zero. These functions should be avoided or should be replaced by smooth approximations. Note, due to the symbolic equation treatment, the inverse functions of the above can also lead to trouble.

3.2.6 Avoidance of redundant events

Due to event propagation events require additional evaluations of the set of equations, which can lead to injury of the computing time restriction [2]. Events can be avoided if discontinuous equations and functions are replaced by continuous approximations.

4 Selection of Modelica libraries

An aim of the project is to use standard libraries as far as possible to achieve good compatibility with other models. Apart from proprietary developments [6] the ThermoFluid library [7] and in particular the new libraries Modelica.Fluid and Modelica.Media [3] are of special interest. Although the Modelica.Fluid library is still in an early development state, this library was selected as the base library. Modelica.Fluid itself is

based on the Modelica.Media library. As an unique feature Modelica.Fluid allows the implementation of models which are in fact medium-independent. In order to achieve good performance, the formulation of the thermodynamic balance equations must be adapted to the material property routines. In the ThermoFluid library this adaption is done explicitly by the user, as a suitable state transformation is selected. In the new Modelica.Fluid library this transformation takes place automatically via skillful use of the index reduction algorithm.

The Modelica.Fluid library is currently under construction. Substantially components like discretized pipes or control valves are still missing. Nevertheless the library is already usable. The interfaces (connectors), a control volume and throttling devices are already available. The Modelica.Media library is already developed further. Property models for ideal gases, several models for water (among other the IAPWS97 formulation [8]) are implemented. Mixtures are implemented likewise. However, these could not be used because of implementation problems in the version that was used. The library already offers a sufficiently good documentation including a tutorial, which makes the implementation of additional property models possible.

5 Component library

To implement the fuel cell subsystem model outlined in chapter 1, component models like heat exchanger, mixer, separator, compressor and exhaust gas turbine are needed. These component models go beyond the scope of the Modelica.Fluid library. Therefore, a new component library ModelicaFluidX is built on basis of Modelica.Fluid. The structure of the library is shown in figure 4. Due to the level of development of the base libraries, structure and implementation are still subject of change. The library is developed aiming real-time applications.

5.1 The CommonFunctions folder

The folder CommonFunctions was taken over from the ThermoFluid library. The functions were partly redesigned as one-liners in order to ensure their automatic differentiability. As an example the function ThermoRoot is shown. The actual Modelica code was generated using the C Preprocessor:

```
function ThermoRoot
  "Square root function with cubic
```

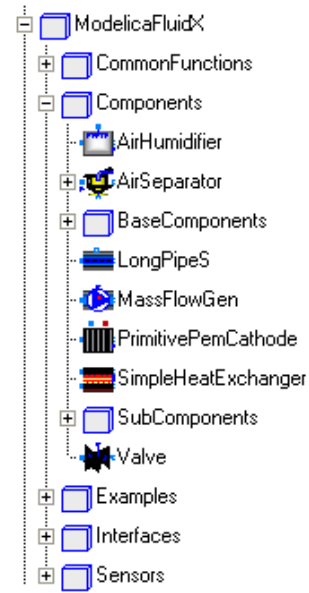


Figure 4: The ModelicaFluidX component library

```

    spline interpolation near 0"
  input Real x;
  input Real delta;
  output Real y;
algorithm
  /*
  // pipe this through 'cpp -P -' to
  // generate Modelica one-liner below
  #define adelta abs(delta)
  #define C1 (5/(4*adelta^(0.5)))
  #define C3 (-1/(4*adelta^(2.5)))
  algorithm
    y := noEvent(if (x > adelta)
      then sqrt(x)
      else
        if (x < -adelta)
          then -sqrt(-x)
          else (C1
            + C3*x*x)*x);
  // EOF
  */
  y := noEvent(if (x > abs(delta))
    then sqrt(x)
    else
      if (x < -abs(delta))
        then -sqrt(-x)
        else ((5/(4*abs(delta)^(0.5)))
          + (-1/(4*abs(delta)^(2.5))) *x*x)*x);
end ThermoRoot;
```

The function extends the root function into the range of negative arguments and avoids the singularity of the 1st derivative in the origin (see figure 5). In contrast to the implementation in the ThermoFluid library, this version can be differentiated automatically. As side effect the restriction to constant approximation radii of the original implementation is void. Since

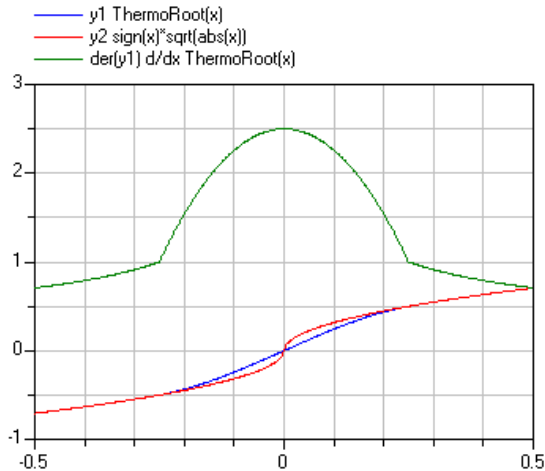


Figure 5: The ThermoRoot library function

the function is continuously differentiable, the automatic event generation can be suppressed by using the `noEvent()` function.

5.2 The Interfaces folder

The library uses mainly the container models which are available in `Modelica.Fluid`. However these are not always sufficient, so additional interfaces must be provided. This applies especially for the discretized models, since `Modelica.Fluid` does not offer vectorized interfaces yet.

5.3 The Components folder

The Components folder contains the subfolder `BaseComponents` and `SubComponents`. The first contains abstract base models for components, the second contains component models, which are used only within other components. To give an idea of the level of implementation, some component models are discussed more explicitly.

5.3.1 The LongPipeS model

Hence `Modelica.Fluid` does not contain a useful model of a discretized pipe, a provisional model was implemented, which consists of a variable number alternating successively arranged control volumes and throttling devices. These two components are available as `JunctionVolume` and `ShortPipe` in `Modelica.Fluid`. In contrast to the more sophisticated implementation in the `ThermoFluid` library only the stationary impulse balance is implemented. For convenience, the model

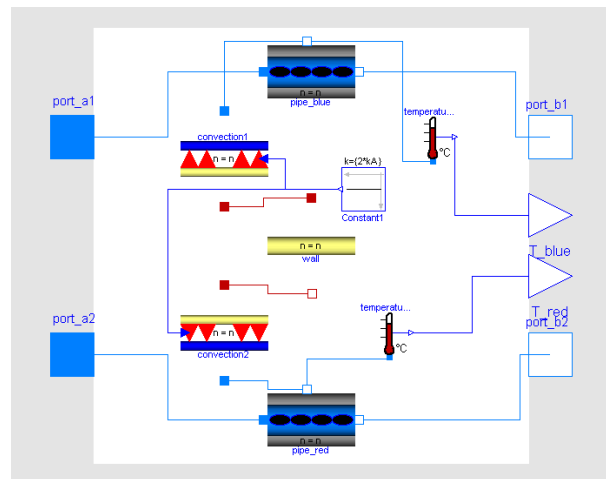


Figure 6: The SimpleHeatExchanger model

has an additional n -dimensional fluid connector. Using this connector, mass or heat can be transferred to each individual control volume.

5.3.2 The SimpleHeatExchanger model

The simple model of a heat exchanger consists of two `LongPipeS` models, the model of a massless wall and two very simple convection models. The models of the wall and of the convection are implemented in the subfolder `SubComponents` and can be replaced easily, if e.g. the thermal capacity of the wall has to be considered. On the other hand, the designs for heat exchangers are so various that generally appropriate abstract models can hardly be indicated.

5.3.3 The SimpleCompressor and SimpleExpander models

For compressors and exhaust gas turbines abstract base models are implemented. The models consider the volumes, the heat capacities and the rotating masses of the machines. The individual behavior of a machine is usually available as characteristic diagrams of mass flow and efficiency. These characteristics are implemented as replaceable class parameters, so that arbitrary machines can be modeled. The characteristic diagrams can take off either only the stationary or also the dynamic behavior of the machine. In most cases only stationary characteristics from measurements are available.

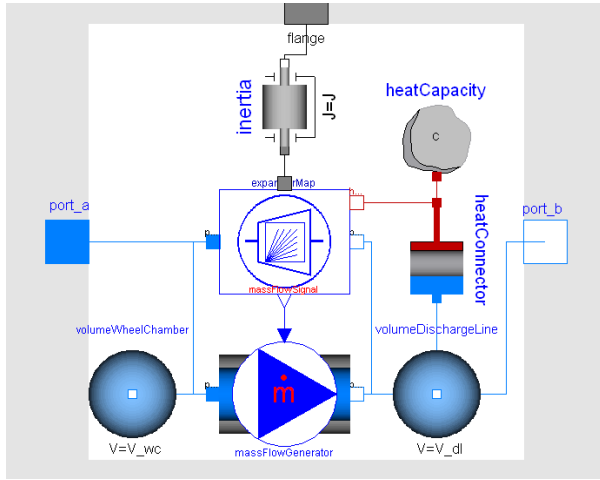


Figure 7: The BaseCompressor model

6 Medium properties

For the computation of the medium properties the Modelica.Media library is used. This library does not contain any model suitable for fuel cell systems computation yet. A model for humid air is needed, whereas both, the humidity and the gas composition are variable. The implementation of such a property model is already possible within the Modelica.Media library, this, however, did not succeed due to general problems concerning the mixture models. As a temporary solution the single component model SimpleAir is used, whereas the restriction of the temperature range was waived. This model is well suitable for real-time simulation in particular because of the simple implementation with constant heat capacity. With the exception of the compressor outlet the temperature of the air remains below 100 °C. In order to simulate the humidification and the drying process of air, a crude workaround is implemented into some component models. The heat of the condensation is computed directly in the component model and the absolute humidity of the medium is passed on as signal to the following component.

7 Simulation of the model

The model which is shown in figure 1 consists of 1761 unknowns, 704 time-varying variables and 31 continuous time states. To enable offline tests, some simple controllers were added to the model. Thus a system startup was simulated using the Dassl variable step solver. The timetable of the system startup is shown in table 2. In figure 8, 9, 10 some results are arranged. For simulation of the startup process with the Dassl

time	action
t = 0 s	The desired pressure ratio is set to 2.2, compressor and turbine starts.
t = 5 s	The stack begins to deliver heat and water.
t = 7 s	The recirculation valve begins to open.
t = 8 s	The recirculation valve reaches 25 % opening.
t = 10 s	Simulation stops.

Table 2: Simulation of the system startup

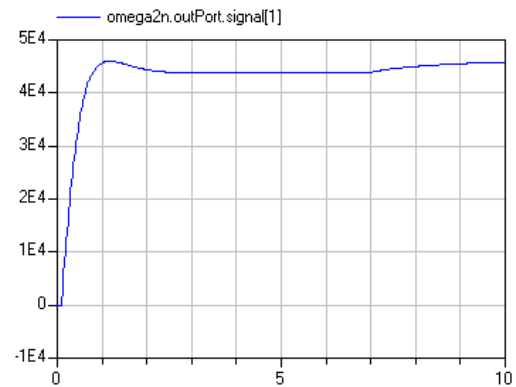


Figure 8: Simulation of system startup: angular velocity of common shaft

solver 1.01 s CPU time was used on a Intel Centrino 1400 MHz. During the entire starting process 16 state events occurred.

For comparison the same model was simulated using the mixed implicit/explicit Euler inline solver. The startup with angular velocity at zero is not possible in this case due to a division by zero. In figure 11 and 12 the deviations between the two simulation runs are shown. The difference between the simulation results is with exception of the very beginning less than approximately 2 %. Larger differences occur in the case of fast changes of the system states.

8 Real-time simulation on the dSPACE HIL target

The system model (figure 2) is inserted into a simple Simulink wrapper model (figure 3) and compiled with the help of the MATLAB Real-Time Workshop for a dSPACE target. The model runs with a stepsize of 2 ms on a dSPACE DS1005 PPC board in real-time. On the target, a certain number of overruns must be allowed, since in particular during the startup phase several overruns arises. Note, such adjustment is not pos-

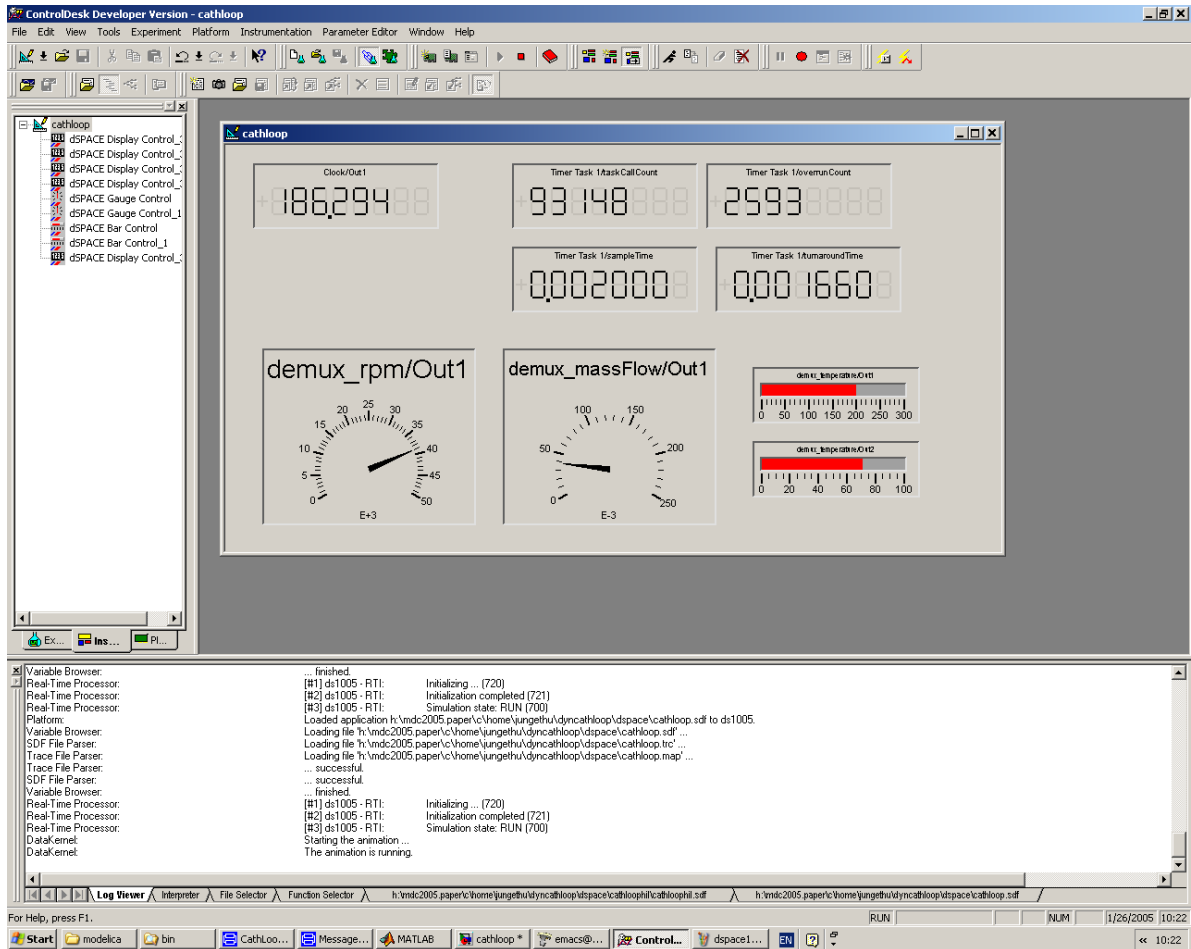


Figure 13: Screenshot of dSPACE ControlDesk with running simulation on a DS1005 target

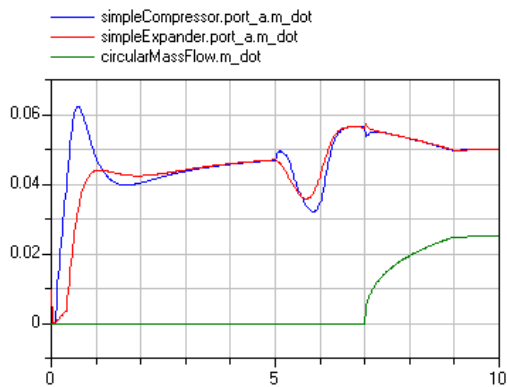


Figure 9: Simulation of system startup: mass flow rate through compressor, exhaust gas turbine and recirculation valve

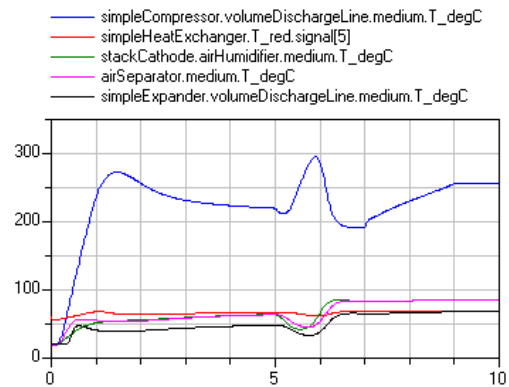


Figure 10: Simulation of system startup: air temperature at humidifier and separator, compressor, exhaust gas turbine and heat exchanger outlet

sible on any arbitrary real-time hardware. On various real-time platforms an overrun is a fatal error which aborts the simulation. In figure 13 a screen shot of the running simulation is shown. The two gages show the number of revolutions of the common shaft and the mass flow through the compressor. Right beside the

temperature in the compressor discharge line and in the outlet of the fuel cell are shown. Above the task counter, the number of overruns, the sample time and the actual turnaround time are indicated. The simulation clock is shown in the upper left corner. Although

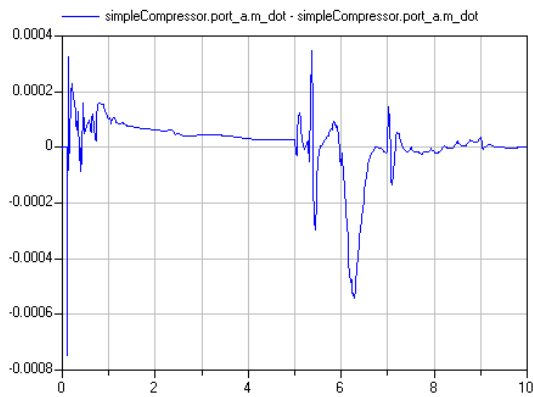


Figure 11: Comparison of simulation results using Dassl solver and fixed-step inline integration: difference of compressor mass flow

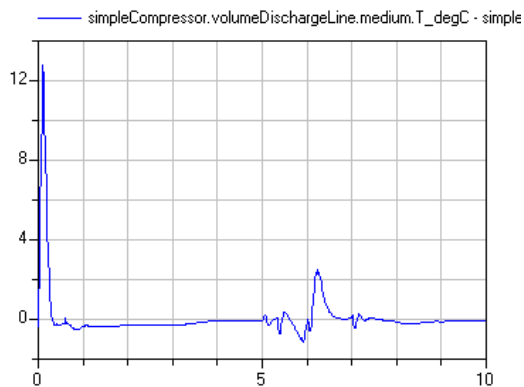


Figure 12: Comparison of simulation results using Dassl solver and fixed-step inline integration: difference of compressor outlet temperature

the mean turnaround time is more than 75 % (ca. 1.5 - 1.8 ms) of the sample time, overruns are rare after the startup phase.

9 Conclusion

On basis of a relatively simple model substantial requirements for real-time modeling of fuel cell systems in Modelica were worked out. Using the Dymola inline integration approach it is possible to use Modelica models in the HIL simulation on dSPACE hardware.

The generation and compilation of the target code with the help of the MATLAB Real-Time Workshop is a complex and expensive solution. Direct generation of the target executable without any MATLAB tools is already possible, but should be better supported by Dymola.

References

- [1] Hilding Elmqvist, Sven Erik Mattsson, Hans Olsson. New Methods for Hardware-in-the-loop Simulation of Stiff Models. In: Proceedings of the 2th Modelica Conference 2002, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002.
- [2] Hilding Elmqvist, Sven Erik Mattsson, Hans Olsson, Johan Andersson, Martin Otter, Christian Schweiger, Dag Brück. Real-time Simulation of Detailed Automotive Models. In: Proceedings of the 3rd Modelica Conference 2003, Linköping, Sweden, Modelica Association, 3-4 November 2003.
- [3] Hilding Elmqvist, Hubertus Tummescheit, Martin Otter. Object-Oriented Modeling of Thermo-Fluid Systems. In: Proceedings of the 3rd Modelica Conference 2003, Linköping, Sweden, Modelica Association, 3-4 November 2003.
- [4] Dymola User's Manual Version 5.1a. Dynasim AB, Research Park Ideon, SE-22370 Lund, Sweden.
- [5] Peter Fritzson. Object-Oriented Modeling and Simulation with Modelica 2.1. John Wiley & Sons, Inc.
- [6] Peter Treffinger, Martin Goedecke. Development of Fuel Cell Powered Drive Trains With Modelica. In: Proceedings of the 2nd Modelica Conference 2002, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002.
- [7] Hubertus Tummescheit, Jonas Eborn, Falko Wagner. Development of a Modelica Base Library for Modeling of Thermo-hydraulic Systems. Modelica 2000 Workshop, Lund, Sweden. <http://www.modelica.org/events/workshop2000>
- [8] Wolfgang Wagner, Alfred Kruse. Properties of water and steam. Berlin, Springer Verlag 1998.

FuelCellLib - A Modelica Library for Modeling of Fuel Cells

Miguel A. Rubio⁺, Alfonso Urquia*, Leandro González⁺, Domingo Guinea⁺, Sebastian Dormido*

⁺ Instituto de Automática Industrial (IAI), CSIC

Ctra. Campo Real, Km. 0,200 – La Poveda, 28500 Arganda del Rey, Madrid, Spain

E-mail: {marubio, leandrog, domingo}@iai.csic.es

* Departamento de Informática y Automática, E.T.S. de Ingeniería Informática, UNED

Juan del Rosal 16, 28040 Madrid, Spain

E-mail: {aurquia, sdormido}@dia.uned.es

Abstract

The design, implementation and use of *FuelCellLib* library are discussed. *FuelCellLib* is a Modelica library for the dynamic modeling of fuel cells (FC). It is intended to be used for: (1) enhancing the understanding of the physical-chemical phenomena involved in the fuel-cell operation; and (2) optimizing the performance of the fuel cells. Physical phenomena are modeled using different hypotheses, in order to allow different levels of detail in the fuel-cell description. *FuelCellLib* version 1.0 (release on January, 2005) is free software, and it will be available on the website of the Modelica Association.

1 Introduction

During the last decades, the planet has been suffering a serious environmental decay, partially due to the use of fossil fuels. As a consequence, a great effort is being made to find alternative sources of energy. The fuel-cells (abbreviated: FC) constitute an alternative source of energy for automotive and residential use.

The modeling and simulation of the fuel-cells is an active research field. Some existing fuel-cell libraries, developed by other authors, have been implemented by using causal simulation languages (for instance, SIMULINK [1]) and fluid-dynamic simulation programs [2]. However, these approaches do not facilitate the modeling task and the model reuse.

The approach adopted in the *FuelCellLib* implementation is different. In order to facilitate easy upgrade and reuse of the models, it has been designed and programmed following the object-oriented modeling methodology.

FuelCellLib is not the only library for fuel-cell modeling written in Modelica language. The library implemented by Steinmann and Treffinger is presented in [3]. Its models are intended for describing the steady-state behavior of the fuel cells, and they do not take into account the dependence with the spatial coordinate. On the contrary, *FuelCellLib* models are intended for describing the dynamic behavior, taking into account the spatial coordinate. The *FuelCellLib* modeling hypothesis are discussed next.

2 Fuel-cell design and operation

The fuel-cell is composed of the following seven fundamental parts: the active layers, the diffusion layers, the terminals of the anode and the cathode, and the membrane (see Fig. 1).

The membrane is placed between the catalytic layers of the anode and the cathode. Protons migrate through the membrane, from the anode to the cathode, along with water. Generally, the proton-exchange membranes are made of Nafion, which guarantees a high protonic conductivity.

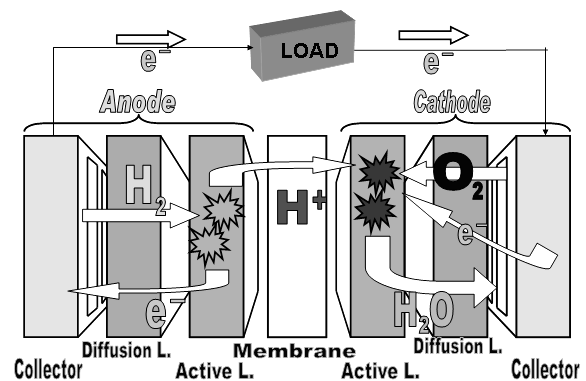


Figure 1: Schematic representation of PEMFC

Usually, the catalyst employed in the active layer is platinum. A small percentage of ruthenium is added, in order to inhibit the poisoning effect of the carbon monoxide. The catalyst is mixed with coal and electrolyte. This catalytic ink is usually deposited over the surface of the anode and cathode diffusion layers.

The diffusion layers are made of porous material to allow the gases and water transport to the catalytic layer. In addition, this diffusion material needs to be a good electric conductor, to allow the flow of electrons between the collector plates and the catalytic layers. The diffusion layer is usually manufactured from coal, paper or cloth. Their conductivity and their resistance to corrosion make these materials adequate.

The collector plates are made of metallic material or non-porous coal. The three fundamental characteristics of these materials are the following: (1) their high electric conductivity; (2) their capacity to maintain a tight cell; and (3) their ability to allow the correct distribution of reagents through the channels.

3 Phenomena modeled

The most outstanding phenomena of PEMFC (i.e., fuel cells with combustible hydrogen) take place in the cell cathode [4,5]. The physical-chemical phenomena modeled in *FuelCellLib* include the following (see Fig. 2):

Membrane:

- Transport of water in liquid and steam phase.
- Protonic conduction.

Catalytic layer of cathode:

- Transport of water in liquid and steam phase.
- Transport of oxygen in steam phase.
- Protonic and electronic conduction.
- Electro-catalytic reaction.

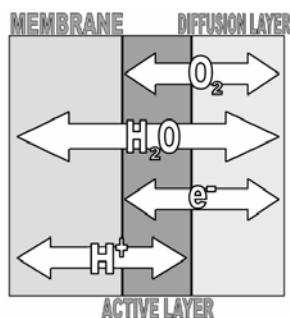


Figure 2: Species involved in the *FuelCellLib* model of PEMFC

Diffusion Layer of cathode:

- Transport of water in liquid and steam phase.
- Transport of oxygen in steam phase.
- Electronic conduction.

Therefore, the *FuelCellLib* models include the following physical-chemical phenomena: the diffusion of gases in porous media, the electronic and protonic conduction and electrochemical reactions. The method of finite volumes is applied for discretizing the PDE.

In addition, the *FuelCellLib* models can be used to simulate the steady-state behavior [6] of the fuel cells along their complete range of operation. For instance, the polarization curve (I-V) of a fuel cell model, composed using *FuelCellLib*, is shown in Fig. 3. The three operation areas (A, B and C) are indicated in the figure: A) fall due to the activation losses; B) fall due to the ohmic losses; and C) fall due to the mass transport at high current value.

The *FuelCellLib* models are based in physical-chemical principles. The balances of the species (i.e., water, oxygen, protons and electrons) are enunciated, in each physical layer of the fuel cell, by means of the definition of *control volumes*.

The properties of the medium inside the control volume are considered time-dependent, but independent of the spatial coordinates. The control volumes exchange the different species with their environment through certain *control planes*. All the interactions between the control volumes are considered *transport phenomena* in *FuelCellLib*. The physical layers of the fuel cell (i.e., the membrane and the catalytic and diffusion layers of the cathode) are modeled by decomposition into control volumes, which are connected to each other by means of transport phenomena.

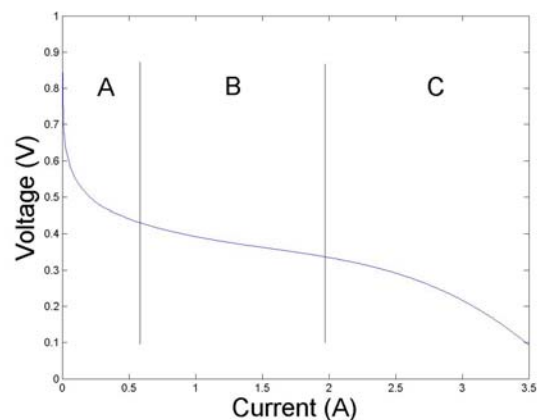


Figure 3: Polarization curve simulated using *FuelCellLib*

These control volumes and transport phenomena can be modeled with different levels of detail, i.e., using different modeling hypothesis. A feature of the *FuelCellLib* models of control volumes and transport phenomena is that they describe different modeling hypotheses. This allows the model user to select the set of hypothesis to be used in a particular simulation run.

4 Modeling hypotheses

The following considerations and approaches are taken in the model

- The FC model is composed of the membrane, the catalytic layer of the cathode and the diffusion layer of the cathode.
- The mixture of gases is considered an ideal gas.
- The flow speed and the gradients of gas pressure are considered small.
- The diffusion electrodes and the catalytic layer have a porosity and a homogeneous tortuosity, with a single pore size, which relates to the considerations of the macro-homogeneous model [9,10].
- The electrodes, the catalytic layer and the membrane are considered isotropic and homogeneous. This is equivalent to consider that the catalyst is evenly distributed in the catalytic layer.
- The model is one-dimensional. The variables change in the direction of the normal to the surface of the membrane and the electrodes. This direction is the x axis.
- The temperature of each cell layer is considered uniform. Nor the Joule effect for movement of species, neither the heat obtained in the electrochemical reaction, is taken into account.
- The movement of the gases is due to the concentration gradients and to the gas pressure in the electrodes.
- The overvoltage of the anode is considered negligible.
- The models are dynamic. They should be able to represent time-dependent behaviors.
- The proton concentration inside the membrane is considered constant.
- The crossover of oxygen in the membrane is not modeled.
- The library user is allowed to choose among the following four mutually exclusive hypotheses:

- Pseudo-capacitance of double layer in catalyst layer.
- Pore size dependence in Knudsen diffusion.
- Electro-Osmotic drag effect in electrolyte.
- Variable electrolyte conductivity with water load.

5 Modeling equations

The equations used to model the three fundamental physical parts of the PEMFC are described in this section.

The thermodynamic open-circuit voltage is calculated [9] from Eq. (1).

$$E_{oc} = E_{ref} - 0.9 \times 10^{-9} (T - 298) + \frac{RT}{2F} \ln\left(\frac{p_{O_2}^{1/2} p_{H_2}}{p_{H_2O}}\right) \quad (1)$$

5.1 Diffusion layer equations

- Balance of gas O_2

$$\frac{p_{O_2}}{RT} \frac{\partial \varepsilon_g}{\partial t} + \frac{\varepsilon_g}{RT} \frac{\partial p_{O_2}}{\partial t} = - \frac{dJ_{O_2}}{dx} \quad (2)$$

where ε_g is the pore volume of the porous media.

p_{O_2} and J_{O_2} are the pressure and the flow of oxygen respectively.

$$\varepsilon_g = \varepsilon_{g_0} \left[1 - \left(\frac{\chi_s}{\chi_s^{\max}} \right)^m \right] \quad (3)$$

The pore equations are common to all physical layers. The pore volume, calculated from Eq. (3), depends on the load of water of the porous material.

$$\chi_s^{\max} = \frac{\rho_{H_2O}^l (1 - \varepsilon_s - \varepsilon_e)}{\rho_s \varepsilon_s + \varepsilon_s} \quad (4)$$

where χ_s^{\max} is the maximum load of water of the porous media. It depends on the pore volume. In the diffusion layer, the term that corresponds to the volume of the electrolyte material does not exist.

$$\varepsilon_s = 1 - \varepsilon_{g_0} - \varepsilon_e \quad (5)$$

The total volume of any layer is equal to the sum of the pore, the solid and the electrolyte volumes, as shown in Eq. (5).

- Balance of gaseous H_2O

$$\frac{p_{H_2O}}{RT} \frac{\partial \varepsilon_g}{\partial t} + \frac{\varepsilon_g}{RT} \frac{\partial p_{H_2O}}{\partial t} = -\frac{dJ^g_{H_2O}}{dx} - \frac{\alpha_v \beta}{RT} (p_{H_2O} - p^{sat}_{H_2O}) \quad (6)$$

where p_{H_2O} and $J^g_{H_2O}$ are the pressure and the flow of water in steam phase respectively.

The condensation and evaporation of water are taken into account in the water balance equation. The two phases are considered to be in balance, and it depends on the specific surface of the porous media.

$$p^{sat}_{H_2O} = p_0^{sat}_{H_2O} e^{\left(\frac{1}{T^{sat}_0} - \frac{1}{T}\right) \frac{L_v M_{H_2O}}{R}} \quad (7)$$

$$L_v = 1.73287 \times 10^6 + 1.03001 \times 10^{-4} T - 4.47755 \times 10^1 T^2 + 7.6629 \times 10^{-2} T^3 - 5.5058 \times 10^{-5} T^4 \quad (8)$$

The saturation pressure can be calculated from Eqs. (7-8). It essentially depends on the temperature.

- Balance of liquid H_2O

$$\frac{\rho_s}{M_{H_2O}} \frac{\partial \chi_s}{\partial t} = -\frac{dJ^l_{H_2O}}{dx} + \frac{\alpha_v \beta}{RT} (p_{H_2O} - p^{sat}_{H_2O}) \quad (9)$$

where ρ_s is the density of the solid and M_{H_2O} is the molar mass of water. χ_s and $J^l_{H_2O}$ are the load and flux of liquid water respectively.

- Transport of gases

$$\frac{\varepsilon_g}{\tau^2} \frac{dp_i}{dx} = \sum_{k=1}^i \frac{RT}{p_c D_{ik}} (p_i N_k - N_i p_k) + \frac{RT J_j}{\frac{\varepsilon_g}{\tau^2} D_j} \quad (10)$$

the term τ represents the tortuosity of the porous material, p_c is the complete pressure and D_{ik} is the binary diffusion coefficient.

The gas flow is described by Eq. (10). It depends on the following two phenomena: the Stefan-Maxwell diffusion and the Knudsen diffusion.

$$D_{ik} = D_{ik_0} \left(\frac{p_c}{p_c^{ref}} \right) \left(\frac{T}{T^{ref}} \right)^{1.5} \quad (11)$$

The binary diffusion coefficient [7] can be calculated from Eq. (11). p_c^{ref} is the reference pressure and T^{ref} is the reference temperature used to measure the binary diffusion coefficient.

$$D_{Knudsen} = D_{Knudsen_0} \quad (12)$$

$$D_{Knudsen} = \left(\frac{4}{3} \right) r_p \sqrt{\frac{2RT}{\pi \cdot M}} \quad (13)$$

The Knudsen diffusion coefficient can be calculated from Eq. (12) or (13). In the first case, it is considered a constant [8]. In the second case, it depends on the pore size. The library user has to choose one of these two modeling hypotheses.

- Transport of liquid H_2O

$$J^l_{H_2O} = -\frac{\rho_s}{M_{H_2O}} D_{H_2O} \frac{d\chi_s}{dx} \quad (14)$$

where the term D_{H_2O} corresponds to the diffusion coefficient of the liquid water in the porous media.

The flow of liquid water is produced by a gradient in the load of liquid water. This is equivalent to assume that the superficial diffusion is predominant [4].

- Electronic conduction

$$J_e = -\sigma_s \varepsilon_s \frac{dV_s}{dx} \quad (15)$$

where V_s is the voltage of solid, J_e the electronic current and σ_s is the electronic conductivity of the solid.

The electronic conduction depends on the conductivity of the solid diffuser and the porosity of the porous media.

5.2 Catalytic layer

- Balance of gas O_2

$$\frac{p_{O_2}}{RT} \frac{\partial \varepsilon_g}{\partial t} + \frac{\varepsilon_g}{RT} \frac{\partial p_{O_2}}{\partial t} = -\frac{dJ_{O_2}}{dx} - \frac{1}{4F} \frac{dJ_e}{dx} \quad (16)$$

The right term of the equation constitutes the oxygen balance in the catalytic layer. The last term represents the effect of the electrochemical reaction.

- Balance of gas H_2O

$$\frac{p_{H_2O}}{RT} \frac{\partial \varepsilon_g}{\partial t} + \frac{\varepsilon_g}{RT} \frac{\partial p_{H_2O}}{\partial t} = -\frac{dJ^g_{H_2O}}{dx} + \frac{1}{2F} \frac{dJ_e}{dx} - \frac{\alpha_v \beta}{RT} (p_{H_2O} - p^{sat}_{H_2O}) \quad (17)$$

In Eq. (16) and (17), the balance of water in the catalytic layer depends on the electrochemical reaction.

- Transport of liquid H_2O

$$J^l_{H_2O} = -\frac{\rho_s}{M_{H_2O}} D_{H_2O} \frac{d\chi_s}{dx} \quad (18)$$

Two alternative modelling hypotheses are supported. The user has to decide whether include in Eq (18) or not the electro-osmotic drag term shown in Eq. (19).

$$J^l_{H_2OT} = J^l_{H_2O} + n_{drag} \frac{J_p}{F} \quad (19)$$

The liquid water flow depends on: (1) the concentration gradient of liquid water; and (2) the electro-osmotic drag, which is due to the proton conduction produced inside the active-layer electrolyte. The coefficient of electro-osmotic drag can be calculated from Eq. (20). This equation is also valid for the electro-osmotic drag in the membrane.

$$n_{drag} = \frac{2.5L_{SO_3}\epsilon_e}{22} \quad (20)$$

$$n_{drag} = 0.0029L_{SO_3}^2 + 0.05L_{SO_3} - 3.4 \times 10^{-19} \quad (21)$$

Eq. (20) is determined experimentally. The electro-osmotic drag mainly depends on the water content of the membrane [10]. Eq. (21) is also empiric [11]. The user needs to choose between Eq. (20) and (21).

$$L_{SO_3} = 14 \quad (22)$$

$$L_{SO_3} = \frac{\chi_s}{\left(\frac{\rho_s}{M_m}\right) - (0.0126\chi_s)} \quad (23)$$

The electro-osmotic drag coefficient depends on the L_{SO_3} term. This term is a function of the load of water in the electrolyte. Two possible expressions are considered in the library. The first one is the constant value shown in Eq. (22), which has been calculated from saturated water vapor at 30°C. The second one is the expression shown in Eq. (23), which takes into account the dependence with respect to the water load.

- Protonic conduction

$$J_p = -K_p \epsilon_m \frac{dV_e}{dx} \quad (24)$$

where V_e represents the electrolyte voltage, J_p the protonic current, K_p the protonic conductivity and ϵ_m the electrolyte volume.

The protonic conduction is produced by a voltage gradient in the protonic conductive material.

- Electrochemical reaction

$$\nabla J_e = a_{act} i_0 \left[\frac{p_{O_2}}{p^0_{O_2}} \exp\left(\frac{\alpha n F}{RT} \eta_D\right) - \exp\left(\frac{-1(1-\alpha)nF}{RT} \eta_D\right) \right] \quad (25)$$

where i_0 is the exchange current in open circuit, a_{act} is the specific active surface in catalytic layer, $p^0_{O_2}$ is the reference partial pressure of oxygen, α is the transfer coefficient of charge in cathode and η_D is the overvoltage between the solid and electrolyte.

Eq. (25) is the Butler-Bolmer expression of the electrochemical reaction. The anode contribution can be neglected [1,4,5].

The model allows the user to choose between Ec. (25) or (26). The pseudo-capacitance is considered in Eq. (26) [1].

$$\nabla J_{T_e} = \nabla J_e + C_{dl} \frac{\partial \eta}{\partial t} \quad (26)$$

The term ∇J_e in Eq. (26) is calculated from Eq. (25). C_{dl} is the capacitance of the double layer between the solid and the electrolyte.

The characteristics of the macro-homogeneous model lead to an underestimation of the overvoltage effect associated to the defect of mass in high current densities. To characterize this phenomenon we introduce in the model Eq. (27), (28) and (29).

$$i_o = i^{ref}_o \left(1 - \frac{J_e}{J_l} \right) \quad (27)$$

$$J_l = \left(\frac{p_{O_2}}{p^0_{O_2}} \right) J_{lim} \quad (28)$$

Four sets of alternative hypotheses are supported by *FuelCellLib*. Different hypotheses leads to different dynamic behaviour of the model (see Figs. 6 and 7).

Two case studies are presented in this work: (1) the simulation of two physical layers (membrane and active layer); and (2) the simulation of the three physical layers (membrane, active and diffusion layer). The results are discussed in the next section.

7 Validation of library

PEMFCs and DMFCs are developed in the Renewable Energy Laboratory of the Industrial Automatic Institute (IAI) of CSIC (see Fig. 8). These fuel-cells have been employed to carry out the experimental validation of the *FuelCellLib* library.

The simulation results obtained using *FuelCellLib* agree with the theoretical and the experimental data. A simulated polarization curve, obtained using *FuelCellLib*, and experimental data, measured from a PEMFC manufactured in the I.A.I, are shown in Fig. 9.

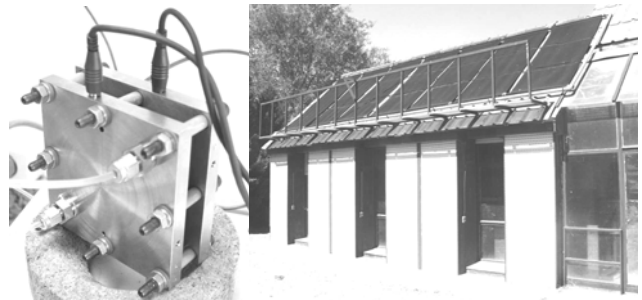


Figure 8: Fuel-cell (left) developed in the Renewable Energy Laboratory of the Industrial Automatic Institute (IAI) of CSIC, Madrid, Spain (right)

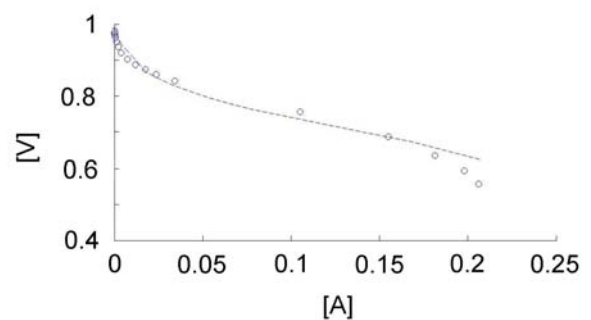


Figure 9: Experimental and simulated polarization curve, exp. (o), model (- -), (2×10^5 Pa of O_2 , $T= 340^\circ K$).

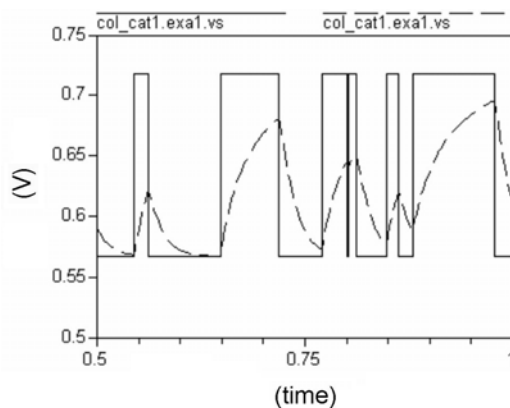


Figure 6: Dynamic simulation of voltage of FC with a pseudo-capacitance hypothesis dependence (- -) and without pseudo-capacitance hypothesis dependence (___).

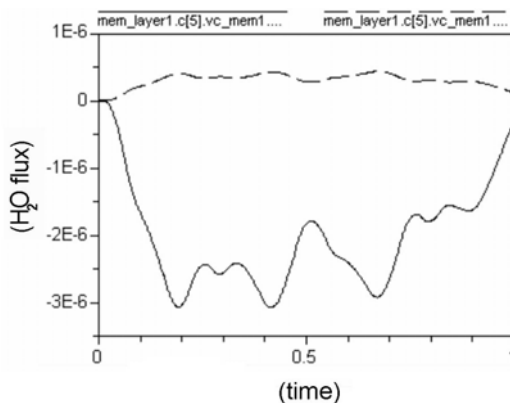


Figure 7: Dynamic simulation of flux of water in membrane with electro-osmotic drag hypothesis dependence (- -) and without electro-osmotic drag hypothesis dependence (___).

Simulation results using *FuelCellLib* are shown in Fig. 10. They can be compared with the results obtained using the model of K.Broka [5] (see Fig. 11). Both results predict the fall of the oxygen concentration along the catalytic layer, and how the fall is bigger as the current increases.

Simulation results obtained using *FuelCellLib* are compared in Fig. 12 with the experimental data presented by J. Larminie [13]. Both predict the effect of the Tafel slope on the polarization curve (see Fig.12).

The simulated voltage of the fuel-cell, obtained in response to a step change in the load, is shown in Fig. 5. It agrees with the experimental data provided by J. Larminie [13].

In addition, *FuelCellLib* can be used to predict phenomena which can not be measured experimentally. For instance, the effect of flooding of the cathode, (see Fig. 13), the water load in the catalytic layer and the membrane. The production of water in the catalytic layer, associated to the diffusion in the catalytic layer and the membrane, is shown in Fig. 13.

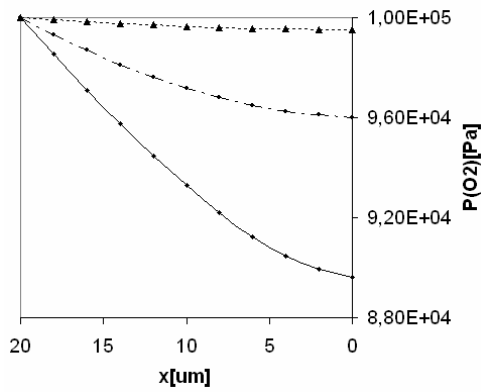


Figure 10: Pressure of oxygen along the catalyst layer with three different current density, 12mA/cm² (. . .), 90mA/cm² (- - -), 230mA/cm² (___); (2×10^5 Pa of O₂, 340°K).

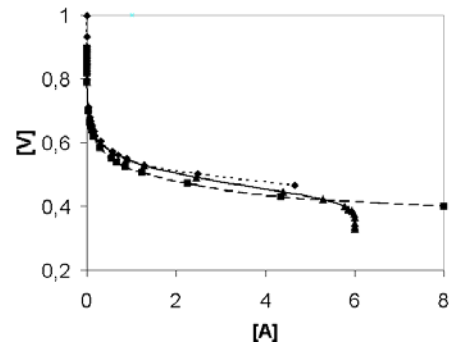


Figure 14: Effect of limit current in polarization curve (. . .) no limit current dependence, (- - -) limit current with no pressure dependence, (___) limit current with pressure dependence.

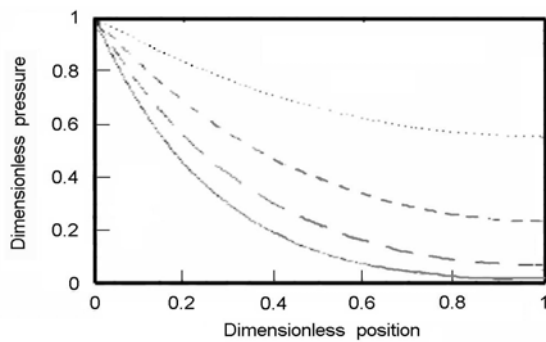


Figure 11: Oxygen pressure in the cathode in x axis in K.Broka model, (.....)250, (- - - -) 500, (_ _ _) and 1000 (___) mA /cm².

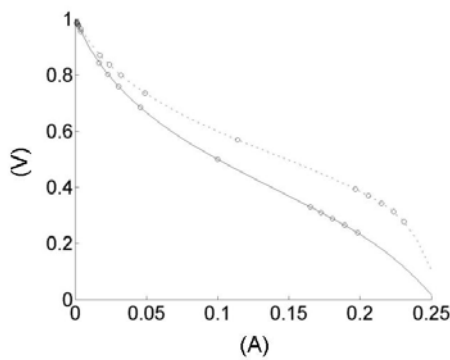


Figure 12: Influence of Tafel slope in polarization curve (I-V) in the library (---150mv),(_ 200mv).

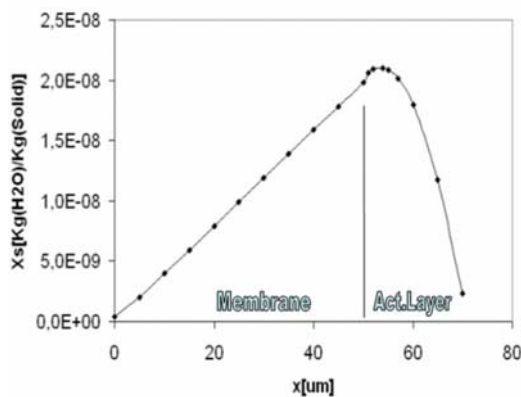


Figure 13: Water load in cathode along the x axis.

The effect of the limit current on the polarization curve, simulated using *FuelCellLib* is shown in Fig. 14.

The simulated polarization curves (see Fig.3, 9, 12, 14) are obtained by reaching a stationary value in the dynamic simulation of the models.

8 Conclusions

FuelCellLib has demonstrated to be useful for improving the understanding of the phenomena involved in the fuel-cell operation. The object-oriented design of the library facilitates the extension, modification and reuse of its code. In addition, the library structure facilitates introducing new modelling hypotheses and comparing the results obtained by using these different hypotheses. The code of the library models is easy to understand by other developers.

9 Future works

Future releases of the library will support the following capabilities: (1) 2D and 3D models, which allow to represent in a detailed way the fluid-dynamics phenomena; (2) new modeling hypotheses that describe the electro-catalytic phenomena with higher level of detail, such as the thin film or agglomerate models; and (3) new models to simulate DMFC (direct methanol fuel cell) and SOFC (solid oxide fuel cell).

Finally, we will be able to obtain a greater quantity of experimental data of the FCs, to achieve a more detailed experimental validation of the library.

References

- [1] M.Ceraolo, C.Miulli, A.Pozio, Modeling static and dynamic behaviour of PEMFC on the basis of electro-chemical description, *J. Power Sources* 113 (2003).
- [2] A.Kumar, R.Reddy, Effect of channel dimensions and shapes in the flow-field distributor on performance of PEMFC, *J. Power Sources* 113 (2003).
- [3] W.D.Steinmann, P.Treffinger, Simulation of Fuel Cell Powered Drive Trains, *Modelica WorkShop 2000 Proceedings*.
- [4] D.Bevers, M.Wöhr, K.Yasuda, K.Oguro, Simulation of polymer electrolyte fuel cell electrode. *J.Appl. Electrochem.* 27 (1997).
- [5] K.Broka, P.Ekdunge, Modelling the PEM fuel cell cathode, *J.Appl. Electrochem.* 27 (1997).
- [6] J.Larminie, A.Dicks, *Fuel Cell Systems Explained*, Wiley 2000.
- [7] A.A.Kulikovsky, *Fuel Cells* 2001,1(2).
- [8] V.Gurau, H.Liu, S.Kakac, *AIChE J.* 2000 46(10).
- [9] D.M.Bernardi, M.W.Verbrugge, *J. electrochem. Soc.* 139,9 (1992).
- [10] T.E.Springer, T.A.Zawodzinsky, *J.Electrochem.Soc.* 138 (1991).
- [11] S.Dutta, S.Shimpalee, *J.Appl.Electrochem.* (2000), 30(2).
- [12] D.B.Genevey, Thesis, F.V.P.I. (2001).
- [13] J. Larminie, A.Dicks, *Fuel Cell System Explained*, Wiley (2000).

A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems

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

In the drug industry the later a substance is discharged from the drug development pipeline, the higher the financial cost. In order to reduce the number of lead compounds a number of systems have been suggested, and in most of these systems modeling and simulation of the lead compound's effects on different metabolic pathways are essential. In these systems, substances that are expected to be harmful or lethal can be removed at an early stage. Consequently, a reduced number of promising lead compounds can be chosen for the concluding tests.

Given Modelica's previous success with huge and complex systems it is likely that it will also be suitable for modeling, simulation, and visualization of metabolic pathway systems, i.e., those systems used in the drug industry. A Modelica library designed to be used for modeling, simulation, and visualization of metabolic pathways is the special-purpose library *Metabolic*, an extension of the abstract Modelica library *BioChem*.

KEYWORDS: Metabolic pathways, pathway modeling, pathway libraries, template models, *BioChem*, *Metabolic*.

1 Introduction

There is currently a great interest in the development of novel analytical technologies for rapid screening of biological dysfunctions in pharmaceutical and clinical applications. In the drug industry the later a substance is discharged from the drug development pipeline, the higher the financial cost. Not only is it costly to test many substances, the price of the tests increase along the development pipeline. Minimizing the number of substances that are fully tested, i.e., becoming lead compounds, is therefore one of the most important aims of all pharmaceutical discovery programs [1].

In order to reduce the number of lead compounds a number of systems have been suggested, out of which some have been realized [2-5]. In most of these sys-

tems modeling and simulation of the lead compound's effects on different metabolic pathways are included. A metabolic pathway can be seen a complex web made up out of several hundred substances and more than twice as many reactions. Substances that are expected to interact in a harmful or lethal way with essential metabolic pathways can be removed at an early stage and a reduced number of promising lead compounds can be chosen for the concluding tests.

In theory, simulations of a single or a few interconnected pathways can be useful when the metabolic pathways under study are relatively isolated from each other. In practice, even the simplest and most well-studied metabolic pathways can exhibit complex behavior due to connections in-between different levels of the whole-cell or whole-organism system.

In light of this, the need for a consistent framework for modeling, simulation, and visualization of metabolic pathways is quite obvious. The object-oriented approach for large scale systems has previously been proven successful in many areas and there is no reason to believe that it should not be useful for metabolic pathway systems.

Given Modelica's previous success with huge and complex technical, physical, electrical, and thermodynamic systems it is likely that it will also be suitable language for modeling, simulation, and visualization of metabolic pathway systems.

So far two Modelica libraries for biological and biochemical applications have been specified. The first library, *BioChem*, is an abstract general-purpose library for biological and biochemical systems. The *BioChem* library is not intended, nor designed to be used directly for creating models and running simulations. The intention with the library is to provide some common basic behaviors, attributes, and environmental properties to be used in special-purpose libraries.

The second library, *Metabolic*, is a special-purpose library extended from the partial models in *BioChem*. *Metabolic* is designed to be used for modeling, simulation, and visualization of metabolic pathways. The models specified in the library describe basic sub-

stances and general reactions that are common in metabolic pathways.

Provided with the reactions in `Metabolic` it is possible to build a library of metabolic pathway templates. The idea is that these general model-templates can easily be extended and adapted to concrete species-specific models. The concrete models can then be used in standalone and connected simulations of metabolic pathways.

1.1 Outline

Modelica has so far mainly been used to model technical, physical, electrical, and thermodynamic systems. Hence the area of biology and biochemistry might be somewhat unfamiliar to some of the readers. For those readers not familiar with some basic concepts and notions in biological and biochemical science this paper will first give an introduction to the area of research. The reader will be acquainted with the concept of seeing the cell as a system and the different levels within this system. A short overview of the data used for modeling and simulation of metabolic pathways is also given. The readers who are familiar to the information presented in the first part of the paper can skip to the fourth section where the work on using Modelica for modeling, simulation, and visualization of biological and biochemical systems is presented.

The second part of the paper starts with pointing out the most significant reasons to use Modelica for biological and biochemical systems, i.e., the benefits of performing modeling and simulation of such systems using Modelica. Subsequently the development of the two Modelica libraries, `BioChem` and `Metabolic` will be in focus, i.e., out-lining the basic design idea behind the two libraries and the environment that they have been developed in. From here on, the paper is concerned with the details of the two libraries and their use. The paper is concluded with some conclusion of the work done so far, and some future work and possible improvements.

2 Introduction to the Area of Research: The Cell as a System

During the past ten to fifteen years the development and introduction of new analytical techniques in the area of biology and biochemistry have greatly increased the amount of experimental data obtained from experiments performed in the area. Automated DNA sequencing, microarray-analysis of gene expressions, and protein profiling are just a few of the methods that have made a significant contribution to the extensive amount of data available. The obtained data can be useful in modeling, simulation, and visualiza-

tion of cellular processes, addressing the whole chain of processes starting with DNA, on to the transcription of DNA into RNA, further on to the translation of RNA into proteins, and finally all the way to the end-concentrations of proteins.

2.1 Chemical Reactions

A chemical reaction involves one or more transformations of one or several substances, called substrates, resulting in one or several new substances, called products. A reaction can be either irreversible, meaning transforming substrate into product, or reversible, meaning not only transforming substrate into product but also the other way around. Strictly speaking, all reactions can be seen as reversible, but for irreversible reactions the re-transformation of substrate into product is essentially so small and/or slow that it is ignored. A reversible reaction can also be seen as two separate irreversible reactions.

Nature's struggle to reach balance is the driving force for all chemical reactions. The speed with which this balance is reached is highly dependent on the environment surrounding the substrates in question. A specific set of substrates, physical variables, and other substances present during the reaction should always result in the same reaction type, progress, and result as long as all the initial values and conditions are the same.

2.2 Reaction Networks

A number of sequential and/or parallel substance transformations can be arranged into a graph, with the edges representing the reactions and the nodes representing the substances. Depending on the reaction in focus most of the substances in a network can function both as substrates and products. Each reaction network will have in-flows and out-flow points, which in turn can be viewed as the substrates and products of a reaction network at a higher level. At this higher level, several of the more specialized reaction networks can be connected through these in-flows and out-flows to form a large super-network.

2.3 Metabolic Pathways in Cells

Cells are the basic building blocks of all living organisms. No matter if the cells are part of a multi-cellular organism, or constitute uni-cellular organisms, the processes inside them do not differ greatly. A cell's metabolism involves the uptake, decomposition, and rebuilding of different compounds and can be seen as several complex webs transporting matter and energy. These complex webs, made out of several hundred substances and more than twice as many reactions, are referred to as cellular or metabolic pathways e.g. the

Starch and sucrose metabolism, the Glycolysis, the Gluconeogenesis, and the Citrate cycle (Figure 1). Many of the reactions participating in these pathways are more or less the same in all cells, while others are highly dependent on the species, the type of cell, or even on the individual that the cell belongs to.

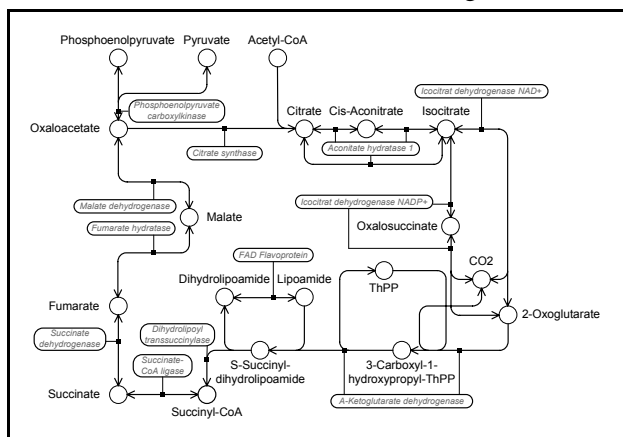


Figure 1. The metabolic pathway Citrate cycle for Baker's yeast (*Saccharomyces cerevisiae*). The enzymes that control the metabolic reactions are connected to the reaction arrows and shown in italic. The circles represent substances that participate in the pathway.

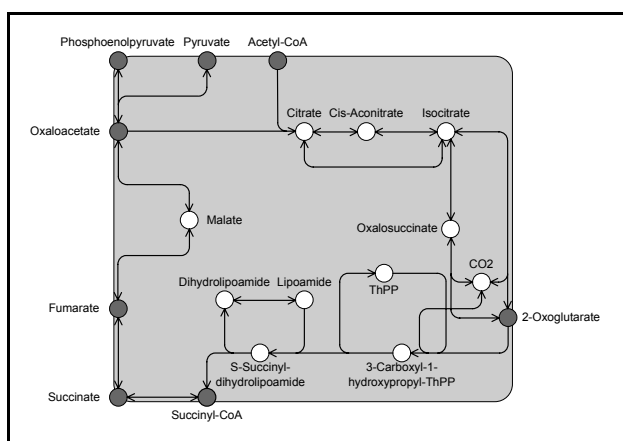


Figure 2. The metabolic pathway Citrate cycle for Baker's yeast (*Saccharomyces cerevisiae*) seen as a sub-system. The dark circles represent substances that are connection points to other metabolic pathways while the light circles represent substances that are internal with respect to the metabolic pathway. (Compare to Figure 1.)

Most of the reactions in these pathways are, in one way or another, controlled by enzymes, i.e., proteins. Proteins are the result of the transcription of DNA into RNA, and the subsequent translation of RNA into amino acid sequences. Enzymes (Figure 1) can either activate or inhibit the reaction in question and the amount of a protein in the cell is controlled by the expression of the gene that codes for that specific protein. One of the greatest challenges in the area right now is to figure out which proteins interact with which reactions and then try to find the corresponding coding gene in the DNA for these proteins.

Some of the reactions in these metabolic pathways are already well-known as well as mathematically defined. Other parts of these pathways are more or less undetermined, ranging from not being fully mathematically defined to not being fully discovered yet.

Each metabolic pathway is highly compartmentalized with a few in-flows and/or out-flows that can be connected to preceding and following metabolic pathways, e.g. the Starch and sucrose metabolism is a preceding pathway and the Citrate cycle is a following pathway of the Glycolysis while the Gluconeogenesis is both a preceding and following pathway of the Glycolysis (Figure 3).

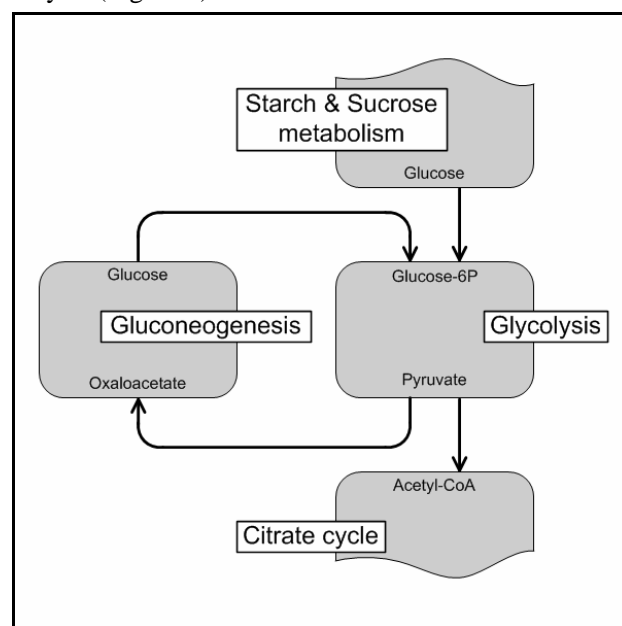


Figure 3. Interconnection of the four metabolic pathways, the Starch and sucrose metabolism, the Glycolysis, the Gluconeogenesis and the Citrate cycle. More pathways do connect to each one of the four pathways, but for simplicity these have been edited out.

2.4 Levels in the Whole-cell System and Multi-cellular Systems

The connection of all possible metabolic pathways for a cell will result in a fully functional system level in the whole-cell system, i.e., the metabolic level. But in order to understand and get a complete view of the entire whole-cell system one needs to look beyond the metabolic level. Apart from the metabolic level the whole-cell system also contains a gene-expression level. The latter level involves not only the transcription of DNA into RNA and the subsequent translation of the RNA into proteins, i.e., enzymes involved in metabolic reactions, but also all interactions in-between DNA, RNA, and proteins. Interactions in-between metabolites, i.e., substances taking part in the metabolic reactions, and DNA, RNA, and proteins are also considered to some extent at this level. Figure 4

provides a somewhat simplified view of the two levels of the whole-cell system.

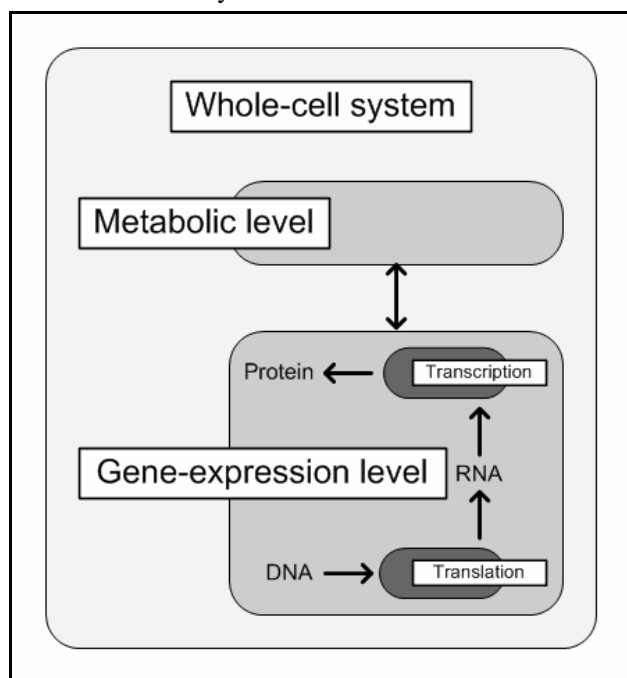


Figure 4. The whole-cell system with the gene-expression and the metabolic levels.

Beyond the whole-cell system there is a meta-level of different kinds of multi-cellular systems that all involves some kind of interchange of substances, and/or communication. Cellular specializations and/or differentiations are common in multi-cellular organisms and the assembly of them can be referred to as an organism level. Both uni-cellular and multi-cellular species can be part of large aggregated multi-species system, i.e., ecological systems.

3 Data Used for Modeling and Simulation of Metabolic pathways

Much of the data regarding metabolic pathways obtained through experiments and analysis is accessible in different public and commercial reference databases. In order to be able to model metabolic pathways one needs to know the participating substances and the reactions in-between them. The organization of entire blocks of metabolic pathways can be found in human-curated maps in public databases, i.e., KEGG [6] and BioCarta's "Proteomic Pathway Project" [7]. The equations specifying the reactions can, however not be found in those maps. This information can instead be retrieved from databases that provide data on individual enzymatic reactions, i.e., BRENDA [8] and EMP [9], and in databases that provide data on multi-step metabolic pathways, i.e., MPW [10] and EcoCyc/HumanCyc [11].

Although all the above resources together represent a good general reference in the work of modeling and simulation of metabolic pathways, they also have significant limitations. The usually non species-specific information causes many errors and inconsistencies, and in many cases the amount of data that can be found for a pathway is not enough for building accurate pathway models [12]. Yet another problem with these databases is that the data contained in different databases might be inconsistent. But even with the mentioned limitation it is still possible to perform modeling and simulation of metabolic pathways with the information provided by the above resources.

4 Benefits of Using Modelica for Biological and Biochemical Systems

Biological and biochemical systems can often easily be described using mathematical relations and expressions. This makes the equation-based Modelica [13] a suitable programming and modeling language for modeling of such systems. First of all, Modelica classes are acausal, i.e., can adapt to more than one data flow context [14], which is a great benefit when dealing with chemical reactions where the flow of matter can move in two directions.

The complexity of biological and biochemical models can be rather high, containing several hundreds of items. However, this will not be a problem since Modelica's strength as a modeling language for complex technical systems is well proven [15].

Moreover, Modelica's strong software component model also makes it ideal as an architectural description language for complex systems [15], e.g. metabolic pathway webs. It is also possible to model both discrete and continuous systems, as well as hybrids thereof [14]. Especially hybrid systems are quite common in the subject area of biology and biochemistry.

Finally, since the complexity of the biological and biochemical models can be rather high. Since Modelica is an object-oriented language the realization of the several hundreds of items within a metabolic pathway will be greatly facilitated through instantiating only a few basic components.

5 Development of the Libraries

5.1 Development Environment

The BioChem and Metabolic libraries have been developed using the MathModelica [16, 17] environment that consist of the Dymola kernel [18], the Mathematica notebook environment [19], and the graphical Model Editor.

In the MathModelica environment the Modelica code along with the documentation for each library is integrated in Mathematica notebooks. This does not only make it easier for non-computer science users to navigate the code, it also facilitates for these users to write their own Modelica classes. The Model Editor is a graphical drag-and-drop interface currently based on Microsoft Visio [20]. The user creates models in the graphical environment by dragging and dropping components from existing model libraries onto the diagram area and then connecting them in a suitable manner. Models can also be created in the Mathematica notebook textual environment, but the models must then first be transferred to the Model Editor in order to get a graphical view of the model.

Once a model has been created it can either be transferred to a notebook for further processing and documentation or simulated in the simulation environment provided by MathModelica. The Dymola kernel handles the simulations by receiving, compiling, and executing the model. The result from the simulation can then be presented with different types of diagram. The parameters and the initial values of the model can also be altered in-between simulations.

5.2 Basic Idea of Library Design

The design idea behind the BioChem library is to create a general purpose Modelica library for modeling and simulation of biological and biochemical systems (Figure 5).

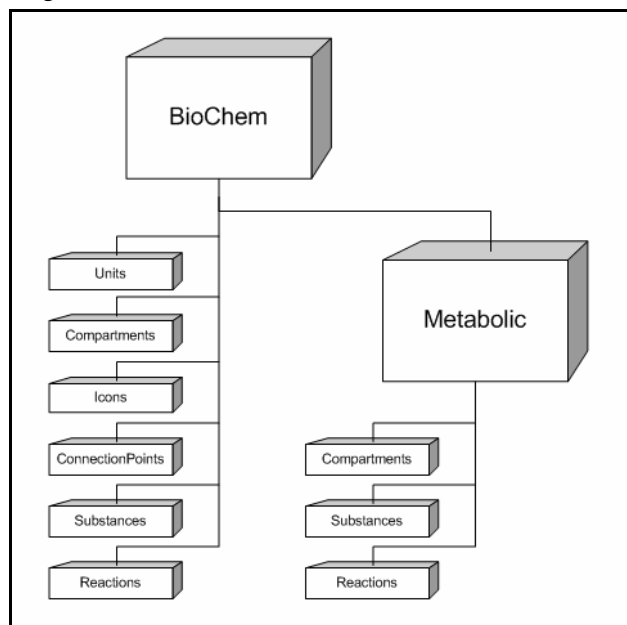


Figure 5. Simplified view of the structure of the BioChem and the Metabolic libraries.

The BioChem library is not intended, nor designed to be used directly for creating models and running simulations, but rather to provide some common basic behaviors, attributes, and environmental properties to be

used in special-purpose biological and biochemical libraries. With the basic features provided in BioChem it is easy to create new special-purpose libraries without extensive addition of new code.

So far the Modelica library Metabolic is the only library to use the features provided by BioChem (Figure 5). The design idea behind Metabolic was to create a special-purpose Modelica library for modeling, simulation, and visualization of metabolic pathways, i.e., modeling, simulation, and visualization of the metabolic level in cells. The classes implemented in Metabolic describe substances and reactions that can take place in-between these substances in a diverse number of metabolic pathways.

5.3 The BioChem library

Most substances and reactions, respectively, have some common basic features. For instance, all substances must have a concentration and all reactions must have at least one substrate and one product. The design objective behind the BioChem library is to collect these basic features of substances and reactions along with units, compartment properties, and other attributes that are commonly used in these kinds of systems in a general-purpose biological and biochemical Modelica library.

```

package BioChem
  package Units
    "Units used in sub-packages of BioChem"
  end Units;
  package CompartmentProperties
    "Properties for compartments used in sub-libraries"
  end CompartmentProperties;
  package Icons
    "Icons used in the package"
  end Icons;
  package ConnectionPoints
    "Connector interfaces used in sub-libraries"
  end ConnectionPoints;
  package ReactionNodes
    "Reaction nodes"
  package Basics
    "Basic components for reaction nodes in the package"
  end Basics;
  package Substances
    "Partial models for substances in sub-libraries"
  end Substances;
end ReactionNodes;
package Reactions
  "Reaction edges"
  package Basics
    "Basic components for reaction in the package"
  end Basics;
  package ReactionTypes
    "Reaction types for reactions in sub-libraries"
  end ReactionTypes;
end Reactions;
package Metabolic
  "Package for metabolic cellular reactions"
end Metabolic;
end BioChem;
  
```

Figure 6. Structure of the BioChem library.

In order to avoid recreating model code for the basic features of substances and reactions for each new Modelica library for biological or biochemical systems these features can instead be collected in one library. Along with substances and reactions it is also practical to define a default environmental container in which the substances are contained and where the reactions can occur. From the visualization's point of view it is also practical to define some default interfaces and

icons which later might be replaced in each sub-library. Not only the icons and interfaces are designed to be easily changed and/or replaced, most of the classes in `BioChem` are designed in such a way that they easily can be extended, and some parameters can also be replaced. The structure of the package is shown in Figure 6.

Due to the design of `BioChem` some restrictions on the types of systems that `BioChem` can be used for arise. The systems that the classes in `BioChem` can be used for are only those biological and biochemical systems that contain chemical reactions. Only for those systems fully functional models that can be used for simulation can be specified.

```

within BioChem;
package Metabolic
  "Package for metabolic cellular reactions"
  package Units
    "Units used in the package"
  end Units;
  package Compartments
    "Different types of compartments used in the package"
  end Compartments;
  package Substances;
  "Reaction nodes"
  end Substances;
  package Reactions
    "Reaction edges"
    package Kinetics
      "Kinetic reactions"
      package UniUni
        "A->B kinetic reactions"
      end UniUni;
      package UniBi
        "A->B+C kinetic reactions"
      end UniBi;
      package UniTri
        "A->B+C+D kinetic reactions"
      end UniTri;
      package BiUni
        "A+B->C kinetic reactions"
      end BiUni;
      package BiBi
        "A+B->C+D kinetic reactions"
      end BiBi;
      package BiTri
        "A+B->C+D+E kinetic reactions"
      end BiTri;
      package TriUni
        "A+B+C->D kinetic reactions"
      end TriUni;
      package TriBi
        "A+B+C->D+E kinetic reactions"
      end TriBi;
      package TriTri
        "A+B+C->D+E+F kinetic reactions"
      end TriTri;
    end Kinetics;
  package SBML
    "Reactions pre-defined in SBML"
    package MichaelisMenten
      "Michaelis-Menten kinetics reactions"
    end MichaelisMenten;
    package Hill
      "Hill kinetics reactions"
    end Hill;
    package Activation
      "Activation kinetics reactions"
    end Activation;
    package Inhibition
      "Inhibition kinetics reactions"
    end Inhibition;
    package Modifier
      "Modifier kinetics reactions"
    end Modifier;
    package Misc
      "Miscellaneous SBML-defined reactions"
    end Misc;
  end SBML;
end Reactions;
end Metabolic;

```

Figure 7. Structure of the `Metabolic` library.

5.4 The `Metabolic` library

Most classes in the `Metabolic` library extend one or more classes in the `BioChem` library. Generally the partial models specified in `BioChem` are extended, and with only a few additions, turned into fully functional

models. The structure of the `Metabolic` package is shown in Figure 7.

As mentioned earlier, many of the reactions that occur in metabolic pathways are more or less the same in all cells no matter what species one look at. This is utilized in `Metabolic` to create a collection of partial models of different metabolic pathways that through small changes and/or additions are turned into fully functional species-specific metabolic pathways.

6 `BioChem` Sub-packages

Since the design objective for `BioChem` was to provide properties and attributes that are common in biological and biochemical systems the library contains several packages holding classes and partial models. The classes can be used as they are in sub-libraries to `BioChem`, while the partial models must be further extended to fully functional models.

6.1 `BioChem.Units`

A number of physical types are needed in order to be able to declare most parameters and variables in the `BioChem` package. Some of the types can be found in `Modelica.SIunits` and are here re-defined in order to avoid long name paths. The SI-types used in `BioChem` are volume (m^3), amount of substance (mol), and concentration (mol m^{-3}).

Most of the other types in the package are non-SI types and thus need to be fully declared. In order for a reaction to actually transport something it has to have a flow of some kind. For a chemical reaction this flow is the volumetric reaction rate ($\text{mol m}^{-3} \text{s}^{-1}$). Together with the concentration, the molar flow rate of a substance (mol s^{-1}) is used in the interfaces between connected components.

6.2 `BioChem.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 container. Within this container the basic physical properties, e.g. volume and temperature, are the same for all reactions that take place and all substances contained in that container.

In `BioChem.Compartments` this is solved using the inner-outer construct, i.e., a global variable. An inner volume is declared in the partial compartment model, giving all objects placed within an extension of the partial model the same surrounding volume. The objects that need to have knowledge of the global volume can use the declaration of an outer volume to reach it. The package so far only contains partial mod-

els for some different types of containers that can be found in cells.

6.3 BioChem.Icons

The package `BioChem.Icons` contains icons used in the drag-and-drop interface of the Model Editor in MathModelica. A substance is represented by a circle and the fill color is changed depending on the type of substance represented, i.e., substance in solution, fixed concentration, gaseous substance, etc. Since the substance only come in a few flavors there is one icon for each type of node.

The reactions on the other hand come in many different variations. A reaction is represented by an arrow with two or more ends. The number of ends an arrow can have is determined by the numbers of substrates and products that are involved in the reaction. Substrate-ends are, by convention, on the left side of the arrow, while product-ends are on the right side. Arrowheads indicate the direction of the reaction, i.e., irreversible reactions only have heads on the product-ends while reversible reactions have heads on both ends.

Instead of creating one icon for each type of reaction the final graphical interface for a reaction is built out of several partial icons. The reaction arrow is divided into three parts, substrates side (left part of the reaction arrow), middle, and products side (right part of the reaction arrow). The middle is the same for all reactions, while the two other parts differentiate depending on the number of substrates and products participating in the reaction.

Enzymes can affect reactions, which is represented by a small arrow and an enzyme sign. The sign represent the type of effect that the enzyme have on the reaction, i.e., inhibition, activation, or a combination of both, and are indicated with a $-$, $+$, and **M** respectively.

6.4 BioChem.ConnectionPoints

The package `BioChem.ConnectionPoints` contains the connector `SubstanceConnector` (Figure 8) that is used when connecting the different components in a model. In order to be able to make simulations using a connected model, the connector has to have a flow variable. For chemical reactions this flow variable is the molar flow rate of a substance (mol s^{-1}). There is also a non-flow variable in the connector, the concentration of a substance. The concentration is later on used in equations with relations to the reaction rate in reaction models.

The connector is used in several partial models in `BioChem.ConnectionPoints`. Each partial model relate to the graphical interface of at least on icon in `BioChem.Icons` (Not more than one icon at a time

though.). For the reaction arrows, connectors are placed at each intended connectable end. For the enzymes regulating the reactions the connectors are placed at the enzyme signs. Finally for substances, eight connectors are placed on the rim of the circle that represents the node of substance.

```

within BioChem.ConnectionPoints;
connector SubstanceConnector
  "Connection point for substance transfer"
  extends Icons.SubstanceConnector;
  Units.Concentration c
  "Concentration of substance at the connection";
  flow Units.MolarFlowRate r
  "Molar flow rate of substance at the connection";
end SubstanceConnector;

```

Figure 8. `SubstanceConnector`, the connector used in `BioChem` and later on also in `Metabolic`.

6.5 BioChem.Substances

The package `BioChem.Substances` contains partial models of different kinds of nodes needed to represent substances in biological and biochemical systems. The basic attributes corresponding to the properties that are studied during simulations, i.e., the amount and the concentration of the substance, are declared in these partial models. All partial substance models also extend the partial model `BioChem.ConnectionPoints.Node`, which contains the connector interface.

6.6 BioChem.Reactions.Basics

All reactions need some basic components in order to work properly. In the package `BioChem.Reactions.Basics` these basic components are collected in a partial reaction model, `Reaction`.

`BioChem.Reactions.Basics` also contains components that are not needed in all types of reactions, but can rather be seen as roles assigned in some reactions while left vacant in others. Using the role-approach, the directions of a reaction can be seen as two roles. The role for a forward directed reaction is almost always appointed, while the role for a backward directed reaction only is assigned for reversible reactions.

The different types of enzymes that can affect a reaction can also be seen as a set of roles. When no enzymes affect the reaction, all enzyme roles are vacant. The different roles that are possible to assign are activator, inhibitor, and modifier. A modifier is a situation dependent enzyme that can react as either an inhibitor or an activator, depending on the environmental context. These roles are also directional, i.e., they can be appointed in both a forward and a backward context.

In `BioChem.Reactions.Basics` model for all the above roles are defined.

6.7 BioChem.Reactions.ReactionTypes

`BioChem.Reactions.ReactionTypes` contains a collection of partial models for different types of reactions that can take place in biological and biochemical systems. The reaction types are obtained by combining some different types of classes from other packages in `BioChem`. First, there is the combination of substrates and products. Then there is the appointment of the two reaction-direction roles. Finally, there is the possibility to appoint an enzyme role. At this point only three substrates and three products are allowed and only one of the enzyme roles can be appointed at a time.

Given the above restrictions four irreversible and seven reversible reaction types for each possible combination of substrates and products are generated, giving 99 different reaction types to choose from in the sub-libraries.

Parts of the graphical interface for the `MathModelica Model Editor` are also defined in this package. Each partial model has a graphical representation in the form of a reaction arrow. If the role for the backward directed reaction is appointed, all the arrow-ends have heads, otherwise only the product-ends have heads. A small arrow perpendicular to the reaction arrow is used to indicate that there is an enzyme-role assigned in the reaction. An enzyme-arrow above the reaction arrow indicate that the enzyme is involved in the transformation of substrate into product, while an enzyme arrow below the reaction arrow indicate that the enzyme is involved in the reverse transformation.

Along with the graphical interface the partial models for connector interfaces in `BioChem.ConnectionPoints` are also extended. Since each of the connector interfaces have been defined in relation to an icon the extensions are quite straightforward.

7 Metabolic Sub-packages

The `Metabolic` library consists of several sub-packages containing fully functional classes that can be used for building models and running simulations of metabolic systems.

7.1 Metabolic.Compartments

The `Metabolic.Compartments` package contains models for some of the different types of containers that can be found in cells when dealing with modeling and simulation of metabolic pathways. The partial compartment models in `BioChem.Compartments` are extended in order to obtain the basic properties of a compartment.

In order to be able to run a simulation of a model all substances, reactions, and other constructs in the model must be placed within a compartment model. Otherwise the global volume cannot be reached with the outer-declaration.

Reactions and substances that require different properties than the ones provided by the main-compartment can be placed in new compartments within or adjacent to the main-compartment.

7.2 Metabolic.Substances

The package `Metabolic.Substances` contains different types of nodes needed for representing a substance in a metabolic pathway. The substance models are specified by extending the partial models of substance nodes in `BioChem.Substances` and adding some additional attributes and equations. Thus both normal substance nodes and nodes with different types of restrictions, e.g. on the concentration of the substance, can be specified.

Typically the concentration in a substance node is allowed to change without restrictions during a simulation, while the total amount of substance in the node is conserved at all times. Some of the models have an assert statement that checks that the concentration never drop more than the tolerance below zero than. The tolerance is a parameter and can thus be changed for every node in a model as well as for each simulation run.

7.3 Metabolic.Reactions

`Metabolic.Reactions` contains a collection of models for different types of reactions that can take place in metabolic pathway systems. The reactions are obtained by extending at least one of the 99 reaction types in `BioChem.Reactions.ReactionTypes` and then adding an equation for the relation between the reaction rate and the participating substances, i.e., substrates, products, and interacting enzymes.

Using more or less all of the possible the reaction types in `BioChem.Reactions.ReactionTypes` four irreversible and sixteen reversible reaction types for each possible combination of substrates and products are generated, giving 180 different reaction models to choose from in the drag-and-drop interface in `MathModelica`.

The `Systems Biology Markup Language (SBML)` is a computer-readable format for representing models of biological and biochemical systems. SBML is, amongst other, applicable to metabolic pathways, cell-signaling pathways, and genomic regulatory networks [21, 22]. SBML has some predefined reactions, which are common in SBML-models of metabolic pathways. All these 32 SBML-reactions are also included in `Metabolic.Reactions` in order to facilitate the

translation of SBML-models into Modelica, and vice versa. The translation of models is performed with a two-way Modelica-SBML parser [23].

8 Conclusions

During the work with the `BioChem` and the `Metabolic` libraries some limitations of the Modelica language has forced us to re-design the libraries' structure at several points. The original `BioChem` library [24, 25] was at a point divided into two libraries, i.e., `BioChem` and the `Metabolic`, which made a significant improvement of the library design and hence the underlying library structure. The design that is presented in this paper is currently being extensively tested and has not shown any major shortcomings this far.

9 Future Work

The `BioChem` package will probably have few additions of classes and models in the future, while there will surely be more packages added. As mentioned before, the main purpose of `BioChem` is to serve as a general-purpose package for biological and biochemical Modelica-packages. Some work with ecological

models in Modelica has been done with inspiration from the `BioChem` library [26]. These models can now easily be added as a sub-library under `BioChem`.

As for `Metabolic`, the limitations on the number of substrates and products for a reaction will be removed. The construction of a library with metabolic pathway templates will also continue. The idea is that these model templates can easily be extended and adapted to concrete models. The concrete models can then be used in standalone and connected simulations. For all of the above tasks, the data contained in the different resources mentioned in Section 3 will be useful.

Acknowledgments

The authors would like to thank MathCore Engineering AB for supplying the MathModelica tool and Andreas Idebrant for providing essential software support. Morgan Ericsson, Växjö universitet, has provided valuable feedback on the text. Emma Larsdotter Nilsson was in part funded by the Swedish National Graduate School in Computer Science (CUGS).

References

- [1] Nicholson, J.K., et al., *Metabonomics: a platform for studying drug toxicity and gene function*. Nat. Rev. Drug. Discov., 2002. **1**(2): p. 153-161.
- [2] Klopman, G., M. Dimayuga, and J. Talafous, *META. 1. A program for the evaluation of metabolic transformation of chemicals*. J Chem Inf Comput Sci, 1994. **34**(6): p. 1320-5.
- [3] Talafous, J., et al., *META. 2. A dictionary model of mammalian xenobiotic metabolism*. J Chem Inf Comput Sci, 1994. **34**(6): p. 1326-33.
- [4] Darvas, F. and G. Dormán, *High-throughput ADMETox estimation : in vitro and in silico approaches*. 2002, Westborough, MA: Eaton Pub. ; BioTechniques Press. x, 89.
- [5] Greene, N., et al., *Knowledge-based expert systems for toxicity and metabolism prediction: DEREK, Star and METEOR*. SAR QSAR Environ Res, 1999. **10**(2-3): p. 299-314.
- [6] Kanehisa, M., *The KEGG database*. Novartis Found Symp, 2002. **247**: p. 91-101; discussion 101-3, 119-28, 244-52.
- [7] BioCarta, *Proteomic Pathway Project*, <http://www.biocarta.com>. 2004.
- [8] Schomburg, I., et al., *BRENDA, the enzyme database: updates and major new developments*. Nucleic Acids Res, 2004. **32 Database issue**: p. D431-3.
- [9] Selkov, E., et al., *The metabolic pathway collection from EMP: the enzymes and metabolic pathways database*. Nucleic Acids Res, 1996. **24**(1): p. 26-8.
- [10] Selkov, E., Jr., et al., *MPW: the Metabolic Pathways Database*. Nucleic Acids Res, 1998. **26**(1): p. 43-5.
- [11] Karp, P.D., et al., *The EcoCyc and MetaCyc databases*. Nucleic Acids Res, 2000. **28**(1): p. 56-9.
- [12] Bugrim, A., T. Nikolskaya, and Y. Nikolsky, *Early prediction of drug metabolism and toxicity: systems biology approach and modeling*. Drug Discov Today, 2004. **9**(3): p. 127-35.
- [13] ModelicaAssociation, *Modelica webpage*, www.modelica.org.
- [14] Fritzson, P., *Principles of Object-Oriented Modeling and Simulation with Modelica*. 2003: IEEE Press and Wiley.
- [15] Fritzson, P. and P. Bunus. *Modelica - A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation*. in *The 35th Annual Simulation Symposium*. 2002. San Diego, California, USA: IEEE.
- [16] Fritzson, P., J. Gunnarsson, and M. Jirstrand. *MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming*. in *The 2nd International Modelica Conference*. 2002. Oberpfaffenhofen, Germany.
- [17] MathCoreAB, *MathModelica website*, www.mathcore.com.
- [18] DynasimAB, *Dymola website*, www.dynasim.se/dymola.htm.
- [19] Wolfram, S., *The Mathematica Book*. 2003: Wolfram Media.
- [20] Microsoft, *Visio website*, <http://office.microsoft.com/en-us/FX010857981033.aspx>.
- [21] Hucka, M., et al., *The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models*. Bioinformatics, 2003. **19**(4): p. 524-31.
- [22] Finney, A. and M. Hucka, *Systems biology markup language: Level 2 and beyond*. Biochem Soc Trans, 2003. **31**(Pt 6): p. 1472-3.
- [23] Vollmar, D., *A two-way SBML translator and graphical icons for BioChem*, in *Department of Computer and Information Science*. 2004, Linköpings universitet: Linköping.
- [24] Larsdotter Nilsson, E. and P. Fritzson. *BioChem - A Biological and Chemical Library for Modelica*. in *The 3rd International Modelica Conference*. 2003. Linköping, Sweden.
- [25] Larsdotter Nilsson, E. *Simulation of Biological Pathways using Modelica*. in *The Huntsville Simulation Conference 2003*. 2003. Huntsville (AL), USA.
- [26] Edelfeldt, S., *Evaluation and comparison of ecological models simulating nitrogen processes in treatment wetlands, implemented in Modelica*, in *Department of Science and Technology*. 2005, Linköpings universitet: Norrköping.

Session 1c

Methods I

Exploiting Weak Dynamic Interactions in Modelica

Francesco Casella

Dipartimento di Elettronica e Informazione

Politecnico di Milano

Piazza Leonardo da Vinci, 32 - 20133 Milano ITALY

e-mail: casella@elet.polimi.it

Abstract

The simulation of complex systems can be made more efficient by splitting the system into several, weakly interacting subsystems, and then integrating their equations separately. The paper discusses the required extension to the Modelica language, as well as the corresponding integration algorithms. Possible applications include real-time integration of large systems, distributed simulation, and the integration of Modelica with external simulators.

1 Introduction

Consider a complex dynamical system, obtained by connecting fast and slow subsystems. The simultaneous integration of the corresponding large set of DAEs can become highly inefficient, especially when a fixed-step algorithm is employed (e.g., for real-time simulation): the fast dynamic subsystems force the adoption of a very small integration time step, and the system of coupled DAEs to be solved at each step is very large.

In some cases, however, the system can be decomposed into two (or more) weakly interacting subsystems, whose describing DAEs can be solved independently at each time step. This allows to split the overall numerical integration task into many smaller tasks, which can be carried out more efficiently; moreover, it opens the way to distributed simulation, where each integration task runs on a different CPU. This method can offer very significant performance improvements in real-time, fixed time-step simulators, e.g. for hardware-in-the-loop and training applications.

The weak dynamic interaction method has been extensively investigated in our research group in the past 10 years. The first notable example is the general-purpose, object-oriented modelling and simulation environment MOSES [1], which relied on an object-oriented database for system modelling, and on

DASSL-RT to perform variable step-size integration. The second example is the ProcSim package [2], a simulation environment for power plant simulation, based on the visual LabView environment, relying on ad-hoc, implicit Euler integration algorithms.

A brief paper on the basic concepts of weak interactions in object-oriented modelling appeared several years ago on the Eurosim Simulation News Europe magazine [3]; the purpose of this paper is to propose an implementation of those concepts in the Modelica framework. The mathematical foundations of weak dynamic interaction are first introduced in Section 2. The extension of the Modelica language is then discussed in Section 3, with reference to a simple example; the problem of generating the corresponding numerical simulation code, as well as the application to distributed simulation are also dealt with. Section 4 follows with further discussion, including the comparison of the proposed approach to other existing methods to speed up real-time simulations. Section 5 concludes the paper with some proposals for future work.

2 Weak Dynamic Interaction: Mathematical Foundations

The concept of weak dynamic interaction is introduced with the aid of a simple example. Consider the electrical circuit represented in Fig. 1.

The complete set of the component and connection DAEs is:

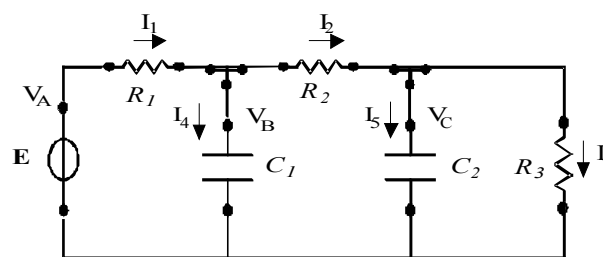


Figure 1: A simple electrical circuit

$$\begin{aligned}
 E - V_A &= 0 \\
 V_A - V_B &= R_1 I_1 \\
 C_1 \dot{V}_B &= I_4 \\
 V_B - V_C &= R_2 I_2 \\
 C_2 \dot{V}_C &= I_5 \\
 V_C &= R_3 I_3 \\
 I_1 - I_4 - I_2 &= 0 \\
 I_2 - I_5 - I_3 &= 0
 \end{aligned}
 \tag{1}$$

In this particular case, it is straightforward to solve the algebraic equations for the algebraic variables, and to rewrite the system in ODE form:

$$\dot{x} = Ax + Bu \tag{2}$$

where $x = [V_B \ V_C]'$ and $u = E$.

Suppose that the system (2) is solved by Euler's *implicit* formula; at each time step, the following linear system has to be solved:

$$H x_{k+1} = x_k + f_{k+1} \tag{3}$$

where $H = I - A\Delta t$, $f = Bu\Delta t$, and Δt is the integration time step. The integration algorithm is A-stable for any Δt . Note that, in the case of a non-linear system, the role of matrix H would be played by the system Jacobian, which should be inverted at (almost) each time step.

If the system is solved by Euler's *explicit* formula, the solution is given by

$$x_{k+1} = F x_k + f_k \tag{4}$$

where $F = I + A\Delta t$. In this case, no matrix inversion is needed; on the other hand, the algorithm is A-stable only if $\Delta t < 2T_{min}$, where T_{min} is the minimum time constant of matrix A . Explicit integration formulae are not convenient if the system is stiff, i.e. if there are mixed fast and slow dynamics, as the fast one dictates the maximum possible time step.

Suppose now that both R_2 and C_2 are sufficiently large: the variation of the voltage V_C within a time step is likely to be small, compared to the voltage drop across the resistor, thus having a *weak* influence on the current I_2 ; conversely, the current I_2 cannot vary V_C substantially over a single time step, and thus has a *weak* influence on V_C . With reference to the R2-C2 connection, the voltage is thus a *weak connection variable* on the resistor side, while the current is a *weak connection variable* on the capacitor side. In other words, an approximate dynamic decoupling can be applied at the connection between R2 and C2. It is then possible split the model in two parts, as shown in Fig. 2.

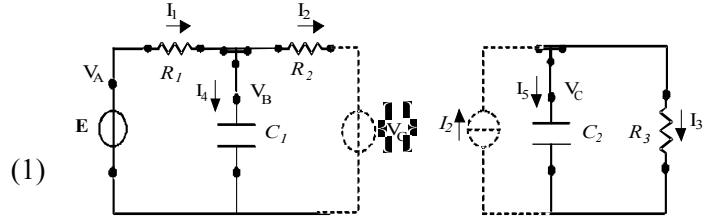


Figure 2: The electrical circuit split into two interacting subsystems.

The left part of the circuit is connected to a fictitious voltage generator V_C , and the right part of the circuit is connected to a fictitious current generator I_C .

The idea is now to adopt a mixed implicit-explicit integration algorithm to the system of Fig. 2. At each time-step, the equations of each sub-circuit are integrated using Euler's *implicit* formula, while taking into account the *last computed value* of the boundary variable I_C or V_C . The corresponding integration formula is:

$$G_L x_{k+1} = G_R x_k + f_{k+1} \tag{5}$$

where G_L and G_R are the following triangular matrices:

$$G_L = \begin{bmatrix} 1 + \frac{\Delta t}{R_1 C_1} & 0 \\ -\frac{\Delta t}{R_2 C_2} & 1 + \frac{\Delta t}{R_3 C_2} \end{bmatrix}$$

$$G_R = \begin{bmatrix} 1 & \frac{\Delta t}{R_2 C_1} \\ 0 & 1 - \frac{\Delta t}{R_2 C_2} \end{bmatrix}$$

As matrix G_L is now triangular (due to the dynamic decoupling), the solution of equation (5) is trivial, and has almost the same computational weight of the explicit formula (4). However, the integration formula remains A-stable for much larger time steps than the fully explicit formula; for large values of R_2 , it is even unconditionally stable. For example, if the following values are taken:

$$R_I = 0.1; C_I = 1; R_3 = 1; C_2 = \{1, 3, 10\}; R_2 = [0.1-10];$$

the stability regions shown in Fig. 3 and 4 are obtained; the stable region is below the limit curve for each value of C_2 . It is apparent how the algorithm based on the weak interaction method allows far larger integration time steps than the explicit algorithm, while requiring a comparable computational time, as no matrix inversion is required.

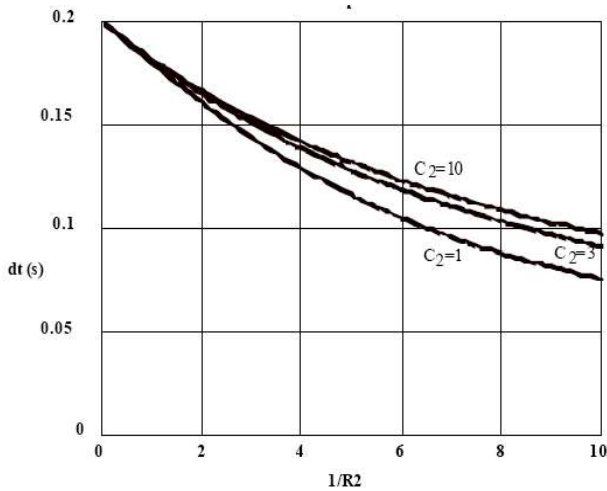


Figure 3: Stability regions of explicit Euler's integration algorithm.

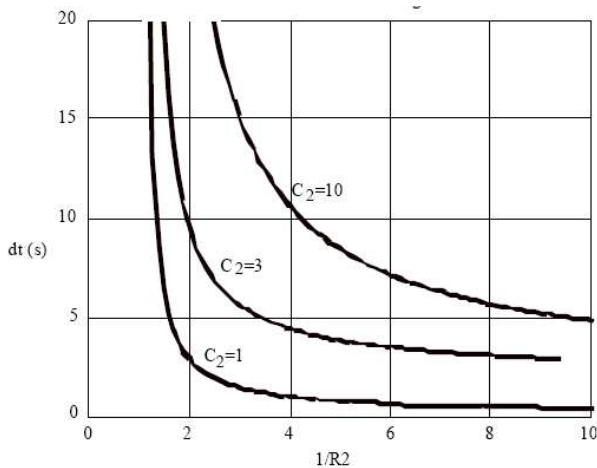


Figure 4: Stability regions of the weakly interacting integration algorithm.

Going beyond the simple example shown here, the advantages of the proposed modelling and numerical integration strategy become more and more substantial if the systems are nonlinear, and as the order of the weakly coupled subsystems grows larger.

Finally, if the dynamics of the weakly coupled subsystems is very different (i.e. one is fast and one is slow), a *multirate* integration algorithm could be adopted, in which the time step of the slow subsystem is a multiple of the time step of the fast subsystem. This kind of strategy, whose detailed analysis is outside the scope of this paper, can lead to even larger computational savings, if the fast subsystems have a few states, while the slow subsystems contains the majority of the states.

3 Weak Interaction in the Modelica Framework

3.1 The weak modifier

The *weak interaction* method will now be introduced in the Modelica framework with the help of a representative example, i.e., the simulation of a power generation system. The system (see Fig. 5) is composed by the connection of two main sub-systems: the mechanical power generation unit (e.g. a gas turbine unit, or a boiler-steam turbine unit), and a synchronous generator, connected either to the grid or to local loads. The aim of the simulation is to simultaneously represent the control system dynamics of both units, with time scales ranging from less than a tenth of a second (swing dynamics of the synchronous generator), to several minutes (boiler pressure dynamics).

Assuming that the rotational inertia has been lumped on the *MechGen* side, the high mechanical inertia of the gas turbine shaft provides a dynamic decoupling between the thermo-mechanical system on the left side of the connection and the electro-mechanical system on the right side of the connection. In other words, a step variation of the torque applied to the shaft by the electrical generator cannot vary the shaft speed substantially over time scales smaller than 0.2-0.5 seconds; this means that the torque is a weak connection variable on the mechanical generator side. On the other hand, since the shaft speed cannot vary substantially over such a time scale, the shaft angle is a weak connection variable on the electrical generator side.

If the two subsystems are connected by means of standard `Modelica.Mechanics.Rotational` connectors, this situation could be described by adding the **weak** modifier to the Modelica language, and by introducing the concept of *weak connection*:

```
connect (MechGen.Shaft (weak tau) ,
        ElecGen.Shaft (weak phi)) ;
```

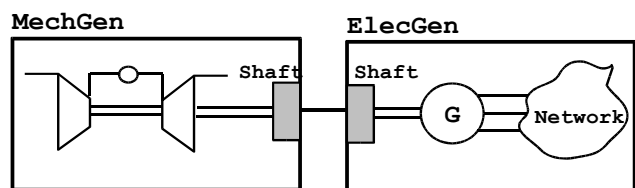


Figure 5: Model of a power generation system.

Alternatively, the **weak** attribute could be directly built in the two sub-system connectors, which should be declared as follows:

```
import Modelica.Mechanics.Rotational.*;
model MechGen
  Interfaces.Flange_a Shaft(weak tau);
  ...
end MechGen;

model ElecGen
  Interfaces.Flange_b Shaft(weak phi);
  ...
end ElecGen;
```

It should also be possible to apply the **weak** modifier to variables declared inside a model, if their influence on the object behaviour across a single time-step is small; for example, the air temperature at the gas turbine air intake.

3.2 Integration Algorithms

The weak variables can now be exploited by the integration algorithm. The basic idea is that every weak variable can be treated as an input variable by the numerical integration algorithm, regardless of its actual physical causality; the corresponding input value will correspond to last computed value available to the integration algorithm.

If the **weak** variables are replaced by **input** variables, the Modelica compiler could then easily decompose the dynamics of the whole system in two weakly interacting subsystems, as shown in Fig. 6. The numerical integration task could then be decomposed into two smaller sub-tasks. Each sub-task will read the last computed value available of its input variables to compute the next value of its output variable, as explained by the following pseudo-code (an explicit integration algorithm is assumed for simplicity):

```
T := 0.05; // Time step length
loop
  (phi, x_mech) :=
    MechGenInt(tau, x_mech, T);
  (tau, x_elec) :=
    ElecGenInt(phi, x_elec, T);
end loop;
```

where x_mech and x_elec are the state vectors of the two subsystems.

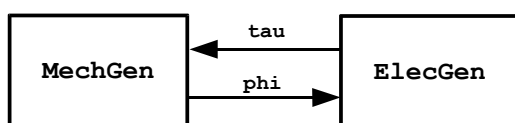


Figure 6: Weakly interacting subsystems.

Each subsystem can be integrated with the method of choice, represented here by the `MechGenInt` and `ElecGenInt` functions. If implicit algorithms are used, the dynamic decoupling leads to two smaller sets of nonlinear equations, to be solved sequentially. This has beneficial effects on the computation time, as already discussed in Section 2.

In the simple example shown above, the two integration algorithms are synchronous (i.e., they share the same time step). However, since the two corresponding subsystems are characterised by widely different dynamic time constants, it is also possible to choose different step lengths for each one, as shown in the following pseudo-code:

```
Tmech := 0.1;
Tel := Tmech/N;
loop
  (phi, x_mech) :=
    MechGenInt(tau, x_mech, Tmech);
  for h in 1:N loop
    (tau, x_elec) :=
      ElecGenInt(phi, x_elec, Tel);
  end for;
end loop;
```

This scheme has a big potential for the real-time, fixed time-step simulation of complex systems: instead of dealing with a huge coupled system, which must be integrated with a small time step dictated by its fastest subsystem, it is possible to split the integration task into many independent sub-tasks, each one having the appropriate step length. In this way, each subsystem is neither under-sampled or over-sampled, and the computational resources are used efficiently.

3.3 Distributed Simulation

If a system can be decomposed into several, weakly interacting subsystems, it is also straightforward to devise a distributed simulation strategy: the integration tasks of each sub-system can be allocated on different CPUs, communicating e.g. through a shared memory database or TCP/IP sockets (Fig. 7).

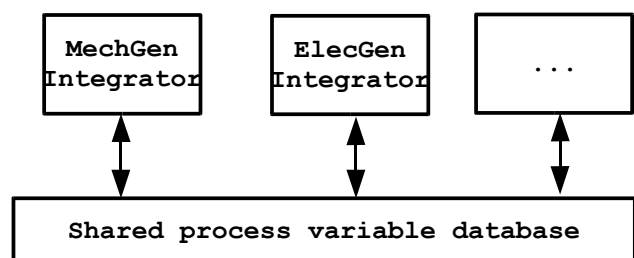


Figure 7. Distributed simulation architecture.

A further possibility is to support co-simulation, by introducing “external simulator” objects, which are an extension of the already existing Modelica external functions. In this case, a Modelica “wrapper” model, defining the input, output, and weakly interacting connector variables, could be written and then linked to external simulation applications. The actual communication between the Modelica-based simulator and the other ones could then be performed via TCP/IP sockets, DLLs, DDE, or any other inter-process communication mechanism. In this way, it would become possible to merge parts of a simulator obtained from a Modelica model with other parts provided by different simulators (e.g. a CFD code, or a digital controller emulator), or by interactive user interfaces.

3.4 Further extensions to the proposed method.

When introducing the example in Section 3.1, the assumption was made that the shaft inertia was entirely contained in the `MechGen` model; in other words, that the electrical generator model inside `ElecGen` had no inertia at all. If this is not true, the weakening method cannot be applied directly as explained: the `ElecGen.Shaft.phi` variable, far from being weakly interacting with the `MechGenShaft.phi` variable, has some inertia rigidly connected to it. The corresponding decoupled integration algorithm would probably be unstable for rather small time steps.

This case could be easily recognised by the symbolic manipulation engine: the weakly interacting model (Fig. 6) would have more state variables corresponding to the weak connection variable `phi`, if compared with the original, rigidly connected model. It could then be possible to devise a symbolic manipulation procedure to remove the inertia contribution from the weak side (i.e., `ElecGen`) and add to the other side (i.e., `MechGen`). This could widen the range of applicability of the proposed method.

4 Discussion and Outlook

The proposed method requires a very limited modification to the Modelica language, i.e., introducing the **weak** modifier to the variable declarations. It also requires a very limited intervention to the user's models, i.e. redeclaring some connector variables as weak, in order to obtain faster simulation code. The **weak** modifier can be thought of as a modelling attribute, stating that a particular variable has a weak dynamic influence on the model behaviour, as well

as a hint to the compiler on how to produce more efficient simulation code.

The main drawback is that an “expert” user is needed, who knows by experience which connection (or model) variables are good candidates to be considered as weak, in order to speed up the simulation. In fact, if the “wrong” weak variables are selected, the integration algorithm could become unstable even for small values of the time step.

The stability analysis of the simulation of an electrical or hydraulic network split into two weakly interacting sub-networks is discussed in detail in [4]. The stability criterion is formulated in terms of the impedances of the two sub-networks; a heuristic rule can then be derived, stating that the sub-network with a weak flow connection variable should have low resistance and/or high capacitance, while the one with a weak effort connection variable should have high resistance and/or low capacitance. A similar, physical-based analysis could be carried out for mechanical connections, such as the case discussed in Section 3.

Another possibility could be to devise numerical indicators (for generic models), based on the analysis of the linearised equations, to help the user determine which connection variables can be considered weak. An exhaustive search of all the potential candidates should not be computationally too expensive, as the number of connections is limited; moreover, only connections between higher-level subsystems could be considered, e.g. the shaft connection in the example of Fig. 5, while ignoring the connections inside the `MechGen` and `ElecGen` models.

The proposed numerical method has already been tested with both variable step-size [1] and fixed step-size [2] integration algorithm. Although a reduction around 30% has been reported in a variable step-size robotic simulation, the application of the weak interaction method to variable step-size, higher-order integration algorithm can only give a limited benefit, as it is hampers the lengthening of the time step, as well as the switching to higher order formulae; moreover, devising multirate, variable step-size algorithms is very difficult. The most significant performance enhancements can be obtained with fixed time-step algorithm, in particular for real-time applications. A typical case is a large physical system with mixed slow and fast dynamics, possibly controlled by digital control systems with widely different sampling times. Such systems are often encountered in industrial practice.

The proposed method can readily benefit from the inline integration method [5], which can be directly applied to the integration of the weakly interacting sub-systems.

There are some similarities between the weak dynamic interaction method and the mixed-mode integration method proposed in [6], as both try to exploit mixed implicit-explicit algorithms to break large non-linear systems of equations into smaller ones. The method proposed in [6] tries to suitably partition the overall model into a “fast” and a “slow” part, by linear eigenvalue analysis and some heuristics to limit the search space, while the method presented here focuses on a system-level approach, i.e. tries to exploit the weak coupling between sub-systems at their connections. It is nevertheless possible that the two methods could be suitably integrated, as their approaches are somehow complementary to each other.

Another method which has been proposed to decouple the equations of large systems is the Transmission Line Method [7, 8]. In this case, the finite propagation speed of physical quantities along connecting elements is exploited to allow the implicit integration of each connected subsystem, using only past values from the other ones. The main advantage in this case is that decoupling comes naturally from the system equations, which are exact; in some cases, such as hydraulic networks or high-frequency electrical circuits, this can be very convenient. On the other hand, there are cases where the “physical” propagation speed is too high, so that the TLM method could unnecessarily introduce fast dynamics when there is no need to; moreover, the exact value of the integration time step becomes tightly coupled with the connecting element parameters. It could be interesting to investigate if the stability analysis studies carried out for the TLM method could be somehow extended to the weak interaction method.

5 Conclusions

The introduction of the **weak** variable concept allows to widen the applicability of the Modelica framework in these directions:

- efficient fixed-step simulation of weakly interacting complex systems, possibly having different time scales;
- distributed simulation;
- co-simulation of Modelica-based simulators and other simulation engines;
- user interaction by means of standard SCADA tools;

while retaining all the advantages of a fully object-oriented description.

Possible applications range from real-time hardware-in-the-loop simulation, to interactive training simulators, to the interfacing of system-level models with detailed (e.g. CFD) models of specific system components.

The implementation of this concept would require the following steps:

1. Extend the definition of the Modelica language to include the **weak** modifier keyword.
2. Implement the symbolic manipulation algorithm to split weakly coupled subsystems, as well as decoupled numerical integration algorithms, into an existing Modelica compiler (e.g., Dymola or OpenModelica)
3. Validate the method on selected case studies (e.g. robotics, power generation systems).

Step 2. could be initially limited to very simple integration schemes, such as synchronous explicit Euler, and then possibly extended to more sophisticated solutions, such as implicit, high-order, multi-step, and multirate algorithms.

References

- [1] C. Maffezzoni, R. Girelli “MOSES: Modular Modelling in an Object Oriented Database”, *Mathematical Modelling of Systems*, v. 4, pp 121-147, 1998.
- [2] A. Leva, A. Bartolini, C. Maffezzoni: “A process simulation environment based on visual programming and dynamic decoupling”, *Simulation*, v.71, n. 3, pp.183-193, 1998.
- [3] F. Casella, C. Maffezzoni: "Exploiting Weak Interactions in Object Oriented Modeling", *EUROSIM Simulation News Europe*, Mar. 1998, pp. 8-10.
- [4] F. Casella: “Modelling, Simulation, and Control of a Geothermal Power Plant”, *Ph.D. Dissertation*, Politecnico di Milano, Italy, 1999, pp. 27-46.
<http://www.elet.polimi.it/upload/casella/tesi.pdf>
- [5] H. Elmquist, S.E. Mattsson, H. Olsson: “New Methods for Hardware-in-the-loop Simulation of Stiff Models”, *Proceedings of the Modelica Conference 2002*, Oberpfaffenhofen, Germany, March 18-19 2003, pp. 59-64.

- [6] A. Schiela, H. Olsson: “Mixed-Mode Integration for Real-Time Simulation”, *Proceedings of the Modelica 2000 Workshop*, October 23-24 2000, pp 69-75.
- [7] D. M. Auslander: “Distributed System Simulation with Bilateral Delay-Line Models”, *Journal of Basic Engineering, Trans. ASME* pp 195-200, June 1968.
- [8] B. Johansson, P. Krus: “Modelica in a Distributed Environment Using Transmission Line Modelling”, *Proc. Modelica Workshop 2000*, October 23-24 2000, pp. 193-198.

Using Automatic Differentiation for Partial Derivatives of Functions in Modelica

Hans Olsson¹Hubertus Tummescheit²Hilding Elmqvist¹¹Dynasim AB, Lund, Sweden (Hans.Olsson@Dynasim.se, Elmqvist@Dynasim.se)²Modelon AB, Lund, Sweden (Hubertus.Tummescheit@Modelon.se)

Abstract

The Modelica language has been enhanced with a notation for partial derivatives of Modelica functions. This paper presents how Dymola [4] enables the use of partial derivatives in certain modeling applications in the Modelica language. It is shown that using partial derivatives is natural and supported in Dymola, and solves several advanced modeling problems.

1 Introduction

Partial derivatives of functions arise naturally in a number of modeling applications. Accurate fluid property functions can be expressed as partial derivatives of a Gibbs- or Helmholtz function with respect to a few variables, e.g. for single-substance fluids as $g(T,p)$, the Gibbs free energy, or $f(T,p)$, the Helmholtz energy of the fluid, see [3]. Partial derivatives are also required to handle non-linear constraints in MultiBody mechanics and contact handling. For contact handling involving parametric surface descriptions, the tangents of each surface is required to specify the constraint equations for the contact point. The tangents are the partial derivatives of the parametric surface description function with regards to the two independent parameters.

These examples demonstrate that partial derivatives of functions occur in several modeling domains. Recently, the Modelica Design group took up this need and a language extension has been made to express partial derivatives of functions in the Modelica language. For this to be actually useful, a Modelica tool like Dymola has to have efficient techniques to generate computationally efficient code for the partial derivatives. In the following sections of the paper we are going to elaborate on the necessary techniques of code generation and give a few application examples

of that. The examples are using the implementation of partial derivative generation in Dymola.

2 Automatic Differentiation of Modelica Functions

Using the Gibbs-function as an illustrative example, we will explain how partial derivatives are generated and used. Since the other thermodynamic properties of a fluid are described as partial derivatives the most natural way of expressing these partial derivatives is to directly express them in Modelica and let the tool, Dymola, differentiate the expressions.

Some simple examples are given in the table:

Property	Formula
Specific volume	$v = (\partial g / \partial p)_T$
Specific entropy	$s = -(\partial g / \partial T)_p$

Modelica 2.2 has thus been extended with the syntax:

```
function Gibbs_pp=der(Gibbs, p, p);
function Gibbs_pT=der(Gibbs, p, T);
```

to express that `Gibbs_pp` is the partial derivative of the function `Gibbs` with respect to `p` and `p`, and `Gibbs_pT` is the partial derivative of `Gibbs` with respect to `p` and `T`. Dymola's existing symbolic differentiation of expressions has been extended with a symbolic variant of automatic differentiation [1], which works for almost any Modelica function (including the Gibbs-functions). It uses forward-mode automatic differentiation. For systems of equations and index-reduction Dymola automatically uses additional derivatives (time-derivatives), and for index-reduction they are also constructed automatically. For the future it is planned to also automatically construct these when needed for Jacobians. Note that

even if time-derivatives and partial derivatives are different they share the same underlying framework.

The Gibbs-function is often fitted as a special polynomial in two variables (and some additional expressions). Generating optimized code for these special polynomials require special care, and is currently only implemented for the simple case of positive and negative *integer* powers. Extending it to handle rational powers will be straightforward. Due to the sparseness of the powers a simple use of Horner's scheme is not optimal, and obviously the pow-function is ruled out because of computational cost.

It is important that the automatic differentiation is done symbolically prior to the code generation since the code for special polynomials can reuse expressions for computing powers. Later this will be extended to include intra-function optimizations between the different partial derivatives. Using code optimization is the key to making symbolic differentiation an efficient form of automatic differentiation, see e.g. [2].

2.1 Basics of automatic differentiation

We will here present the basics of automatic differentiation in a general setting, even though we have only implemented the features needed for partial derivatives of functions. We will then consider the implementation choices, and the special cases in Modelica.

2.1.1 Forward mode

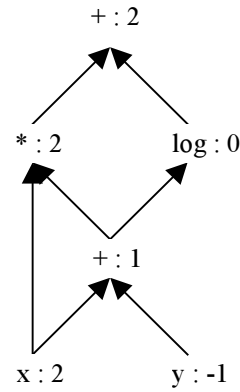
When performing automatic differentiation real variables and expressions are replaced by Taylor/power-series¹ (in one variable 't' – representing a directional derivative) whereas non-reals, e.g. the conditions of if and while-clauses are kept unchanged.

The rules for propagating Taylor-series through functions and expressions can be found in [6], and we will here only consider a trivial example

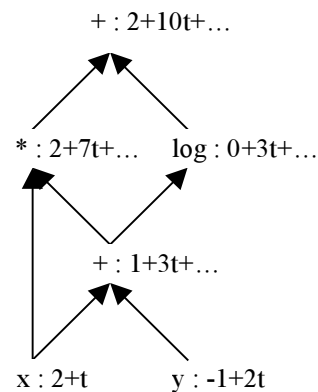
$$z := x \cdot (x + y) + \log(x + y)$$

The computation of this expression can be computed from its corresponding directed acyclical graph by propagating the numerical values.

¹ For higher order differentials a different scaling of the coefficients is more efficient – we will ignore that in this paper.



Thus if $x = 2, y = -1$ we get $z = 2$. For automatic differentiation we replace the values at the bottom by Taylor-series and propagate these upwards:



Thus if $x = 2, \partial x / \partial t = 1, y = -1, \partial y / \partial t = 2$ we obtain $z = 2, \partial z / \partial t = 10$ and by including higher terms we would get higher order derivatives. For each node we only have to consider the values on the arrows entering it, which lead to efficient computations of directional derivatives, and is thus efficient for computing both partial derivatives and time-derivatives.

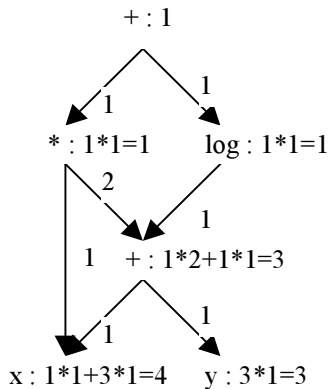
The result of this assignment-statement (in terms of Taylor-series) can then be used directly in the next statement in Modelica.

2.1.2 Reverse mode

Reverse-mode automatic differentiation [1] is an efficient technique for computing the derivative of one variable with respect to many. Since it is not as efficient for computing partial derivatives we will only present an example of how it works and not the underlying theory.

Consider the above example and augment the graph with the partial derivatives for each primitive operation (using the numerical values). Then start from the top where $\partial z / \partial z = 1$, and for each node compute the sum of the products of nodes above times the value along the edge.

In order to indicate that values are propagated downwards the arrows have been reversed:



The interpretation of the result is that the value at each node represents the partial derivative of z with respect to this value: $\partial z / \partial x = 4$, $\partial z / \partial y = 3$ (which is consistent with the result above).

To implement reverse mode one first go through the algorithm once to compute the values for each node, and then once in reverse order using these values. This requires that all intermediate results are stored.

2.1.3 Implementation choices

Automatic differentiation can be implemented in several ways [1, 5], and forward-mode is in general simpler to implement than reverse-mode. We have selected to perform forward-mode symbolically in the Dymola kernel.

Another possibility that has attracted attention recently is to generate code that numerically propagates derivatives e.g. by overloaded operators in C++ or by modifying the code-generation for each primitive operation.

The derivative annotation in Modelica was designed with this in mind and can thus be used when computing Jacobians for non-linear systems using the ‘time-derivative’ of the function and also internally to compute partial derivatives.

Furthermore the ‘time-derivative’ functions (i.e. functions propagating a numerical directional derivative) can be constructed automatically by Dymola by automatic differentiation. This could have been implemented by modifying the code generation for each operation to also numerically propagate directional derivatives.

However, the symbolic variant has the advantage [5] that:

- Expressions independent of e.g. T do not have to propagate the derivative of T .

- The symbolic derivative is a new function that can be manipulated further by Dymola’s kernel (e.g. to compute another derivative).
- No need to modify the code-generation in Dymola, and the generated code can be compiled with compilers on realtime platforms where C++-compilers are not always available.

The fact that the symbolic derivative is a new function is also used in this paper since it allows us to present the result of automatic differentiation as Modelica functions.

2.1.4 Special cases in Modelica

In Modelica, functions can contain simple expressions, matrix expressions, expression with iterators and if-, while- and for-clauses. The symbolic differentiation handles all of them, which has required special care, e.g. the rules for simple differentiable expressions with iterators are (where we use the special notation $x' = \partial x / \partial t$):

- $\{x(j)+x'(j)*t \text{ for } j \text{ in } 1:n\} = \{x(j) \text{ for } j \text{ in } 1:n\} + \{x'(j) \text{ for } j \text{ in } 1:n\} * t$
- $\text{sum}(x(j)+x'(j)*t \text{ for } j \text{ in } 1:n) = \text{sum}(x(j) \text{ for } j \text{ in } 1:n) + \text{sum}(x'(j) \text{ for } j \text{ in } 1:n) * t$
- $\text{product}(x(j)+x'(j)*t \text{ for } j \text{ in } 1:n) = \text{product}(x(j) \text{ for } j \text{ in } 1:n) + \text{sum}(\text{product}(\text{if } j==k \text{ then } x'(j) \text{ else } x(j) \text{ for } k \text{ in } 1:n) \text{ for } j \text{ in } 1:n) * t + \dots$

The Modelica expert will note that we have not included the rules for min- and max-expressions with iterators, since these are more complex to compute, often discontinuous, and currently not needed.

If the function being differentiated contains calls of other functions the directional derivative is propagated through it by its derivative-function, which is either specified in the derivative-annotation, or constructed by Dymola through automatic differentiation of the function (the latter case assumes the function is non-external).

2.1.5 Non-differentiable functions

A basic limitation of automatic differentiation is that it can provide a derivative even at points where the function does not have a derivative. Verifying that a function with branches (if-statements, if-expressions, or while-statements) is continuous is a difficult problem, see [7]. Furthermore in this reference it is shown that automatic differentiation may produce incorrect results for specific inputs if the function contains equality tests on real values.

For functions declared as partial derivatives (as is the focus of this paper) one can view the continuity as

the responsibility of the modeler declaring the partial derivative function. When using automatic differentiation for index-reduction this is not feasible, and the declarative approach in Modelica is that automatic differentiation for index reduction requires the function to specify the degree of continuity.

3 Fluid Property Modeling using Gibbs- or Helmholtz functions

The Gibbs-function is often fitted as a polynomial in two variables:

$$g(T,p) = \sum a_i p^{J_i} T^{K_i}$$

This type of function can be differentiated efficiently with the implementation of automatic differentiation in Dymola. The code for expressing the other thermodynamic properties has become much shorter than in previous implementations of fluid properties. Furthermore, conditions such as the phase equilibrium are also expressed as an equation involving the partial derivatives. Thus it is possible to describe phase equilibrium conditions, e.g. between gas and liquid phases, in a completely declarative way, without resorting to special algorithms. Initial numerical experiments seem to indicate that this works for typical working fluid in thermodynamic cycles, e.g. water or refrigerant R134a.

3.1 Definition of Thermodynamic Properties

A complete application example is the definition of phase equilibrium in two phase fluids. In particular when validity above the critical point is necessary, Helmholtz functions are used to describe high accuracy thermodynamic surfaces. A typical equation from [3] is the one for R134a using a dimensionless Helmholtz energy composed of an ideal (a_{id}) and a residual term (a_{res}) with the general form:

$$\frac{a_{res}(\tau, \delta)}{RT} = \sum_i^{l-4} \sum_j^{k_i-k_{i+1}} n_j \tau^{t_j} \delta^{d_j} \exp(-\delta^{d_i})$$

$$\frac{a_{id}(\tau, \delta)}{RT} = \ln(\delta) + a_1 \ln(\tau) + \sum_{i=1}^3 a_{i+1} \tau^{t_i}$$

Where τ is a reduced inverse temperature and δ is a reduced density. This Helmholtz function uses fractional powers and for the test implementation it was more efficient to transform all exponents into integers via a variable substitution. All properties of interest are then computed by automatic differentiation

using the new syntax form, and by computing the partial derivatives of $a_r(\tau, \delta)$ and $a_i(\tau, \delta)$ first, e.g.:

```
function ar_t=der(ar, tau);
function ar_tt=der(ar, tau, tau);
function ar_d=der(ar, delta);
```

and so on for all partial derivatives up to order two. The properties themselves are then defined as functions of these partial derivatives, e.g. the pressure

$$p = RT\rho \left(1 + \rho \frac{\partial}{\partial \delta} a_{res}(\tau, \delta) \right)$$

Which results in a nice, compact definition in Modelica:

```
function pressure "pressure"
  input SI.Temperature T;
  input SI.Density d;
  output SI.Pressure p;
protected
  Real delta = d/DCRIT "dim-less density";
  Real tau = TCRIT/T
  "dimensionless inverse temperature";
algorithm
  p := R*T*d*(1+delta*ar_d(tau,delta));
end pressure;
```

The Gibbs energy is computed as:

```
function gibbsEnergy "Gibbs free energy"
  input SI.Temperature T;
  input SI.Density d;
  output SI.SpecificEnergy g;
protected
  Real delta = d/DCRIT "dim-less density";
  Real tau = TCRIT/T
  "dimensionless inverse temperature";
algorithm
  g := R*T*(1+a0(tau,delta)+ar(tau,delta)
    + delta*ar_d(tau,delta));
end gibbsEnergy;
```

In an equivalent manner, all other properties of interest are defined.

3.2 Declarative Definition of Phase Equilibrium conditions

All current Modelica libraries define two phase fluids for dynamic simulation via auxiliary equations, e.g. splines generated from accurate phase boundary data, that are a very good approximation to the correct thermodynamic equilibrium conditions. There is one fundamental drawback to that approach: the approximation accuracy is fixed and has to be chosen quite high to prevent numerical inconsistencies at tight solver tolerances. From a perspective of a de-

clarative, equation based language, a declarative definition of the phase equilibrium condition has the advantage that it is always solved to the current accuracy of the numerical solver. This means that in contrast to current implementations that have a maximum accuracy that is limited by the accuracy of the phase boundary approximation, a declarative definition does not have this drawback. Even though in Modelica it would be possible to define an iterative scheme to compute phase equilibrium conditions, the algorithm would need the current solver tolerance as a user-defined input, again an undesirable drawback. The declarative definition for phase equilibrium are the two equations:

$$p(T, \rho_{liquid}) = p(T, \rho_{vapour})$$

$$g(T, \rho_{liquid}) = g(T, \rho_{vapour}),$$

i.e. equality of the pressures and Gibbs energies computed from the same saturation temperatures and the liquid and vapour densities ρ_{liquid} and ρ_{vapour} respectively.

With the property functions defined in the last sections, the phase equilibrium conditions only need the variables and equations in the following code fragment:

```
SI.Density dl(start = 1500.0) "liquid";
SI.Density dv(start = 5.0) "vapour";
SI.Temperature T(start = 270.0);
equation
p = pressure(T,d);
h = enthalpy(T,d);
pressure(T,dv) = pressure(T,dl);
gibbsEnergy(T,dl) = gibbsEnergy(T,dv);
```

From these equations the non-linear solver will compute the liquid and vapour densities for the saturation temperature T . The equations are taken from the context of a dynamic control volume model that assumes the pressure p and the enthalpy h as dynamic states.

3.3 Discussion

These equations have so far been tested in simple setups, and robustness, speed and convergence were excellent, provided that the initial values for ρ_{liquid} and ρ_{vapour} were close to the initial equilibrium point.

Unfortunately, there are still some unsolved problems that prevent to use this formulation in many applications: above the critical point these equations lose a meaning and it was numerically not possible to obtain meaningful results for dynamic simulations

that come from a supercritical state and go to a subcritical state. The main problem here is that the equilibrium conditions, for the Helmholtz equation above, has several non-physical, numerically valid solutions in the unstable region inside the two-phase dome.

Dymola can handle inequality-constraints on the solutions of non-linear systems, but we have not yet determined the best way to specify these inequalities in Modelica, because we need to be sure to find only the thermodynamically stable solutions outside of the spinodal lines. There are inequality conditions on some partial derivatives for these non-physical solutions, see [3], that could be used to disambiguate unwanted solutions.

A combination of some auxiliary functions for start values and to disambiguate non-physical solutions with the declarative definition of the phase boundary through equations is likely to be a compromise that works robustly under all conditions and avoids the disadvantages of both approaches.

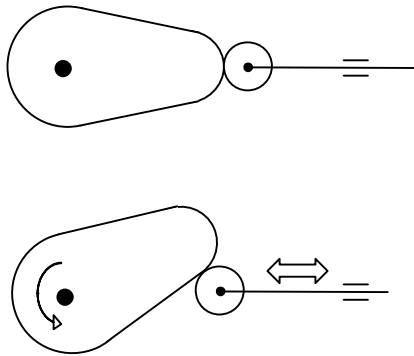
In the test implementation, Jacobians for the non-linear equations have not yet been derived automatically. This would further improve the robustness and solution speed. In dynamic simulation, Dymola uses the last solution point as a start for the next iteration and that makes the otherwise time consuming equilibrium iterations quite fast. For large systems of non-linear equations arising from steady-state problems, the method of using auxiliary functions for the saturation pressure and temperature is likely to be more robust and more flexible. Often the auxiliary functions can be chosen in a way that explicit evaluation is possible in cases when the proper thermodynamic definition inevitably leads to a non-linear equation system.

The main advantage of automatic differentiation for medium properties is the economy of code: the thermodynamic surface for R134a, the computation of the phase boundary and all derived properties is less than 5% of the amount of code with conventional programming “by hand” and auxiliary functions representing the phase boundary.

4 Non-linear Constraints in Multi-Body Mechanics and Contact handling

Partial derivatives are sometimes required to handle non-linear constraints for contact handling in Multi-Body mechanics, see also [9].

As a detailed example, we will consider the CAM mechanism shown below, [8]. The CAM has straight flanks between the circle segments. The follower is roller-ended.



The CAM shape will be described by a replaceable function defining the two dimensional position vector of every point of the circumference as a function of a free parameter theta.

```
partial function shapeFunction
  input Real theta;
  output Real r[2];
end shapeFunction;
```

The straight flanks CAM can be defined as follows:

```
function straightFlanksCam
  extends shapeFunction;
  input Real R1=1 "Base circle radii";
  input Real R2=0.5 "Nose radii";
  input Real d=2 "Centre distance";
  import Modelica.Math.*;
protected
  constant Real pi=Modelica.Constants.pi;
  Real fi0;
  Real fi1;
  Real fi2;
  Real L;
  Real x;
  Real y;
  Real thetamod;
algorithm
  thetamod := atan2(sin(theta),
    cos(theta))
  "to get angle in interval -pi..pi";
  fi0 := asin((R1 - R2)/d);
```

```
if thetamod > 0 then
  fi1 := pi/2 - fi0 - thetamod;
else
  fi1 := - pi/2 + fi0 - thetamod;
end if;
fi2 := atan2(R2*cos(fi0), d +
  R2*sin(fi0));
if abs(thetamod) > pi/2-fi0 then
  x :=R1*cos(theta);
  y :=R1*sin(theta);
elseif abs(thetamod) >= fi2 then
  L := R1/cos(fi1);
  x := L*cos(theta);
  y := L*sin(theta);
else
  L := d*cos(abs(thetamod)) +
  sqrt(R2^2 - d^2*sin(abs(thetamod))^2);
  x := L*cos(theta);
  y := L*sin(theta);
end if;
r := {x,y};
end straightFlanksCam;
```

The mechanism can be described by the following equations (three dimension vectors are used for convenience although the mechanism is planar).

$$\mathbf{r}(\theta) = \text{shape}(\theta)$$

$$\mathbf{T}(\varphi) \cdot (\mathbf{r}(\theta) + \mathbf{n}_n(\theta) \cdot (R + \text{dist})) = \{x, y, z\}$$

$$\mathbf{r}_\theta(\theta) = \text{shape_theta}(\theta)$$

$$\mathbf{n}(\theta) = \mathbf{r}_\theta(\theta) \times \{0, 0, 1\}$$

$$\mathbf{n}_n(\theta) = \frac{\mathbf{n}(\theta)}{\sqrt{\mathbf{n}(\theta) \cdot \mathbf{n}(\theta)}}$$

$$\mathbf{T}(\varphi) \cdot F_n \cdot \mathbf{n}_n(\theta) = \{F_x, F_y, F_z\}$$

where

\mathbf{r}	vector to closest point
θ	angle to closest point
φ	angle of CAM
\mathbf{T}	Transformation matrix for rotation around z axis
x, y, z	position of center of follower
R	radii of follower
dist	distance between CAM and follower
\mathbf{n}	normal of CAM shape
\mathbf{n}_n	normalized normal
F_x, F_y, F_z	force on follower
F_n	normal force

The partial derivative of the replaceable shape function is needed. The Modelica language has recently been extended to allow the der-operator to define partial derivatives of Modelica functions.


```

replaceable function shape =
  shapeFunction;
function shape_theta =
  der(shape, theta);

```

This allows to write equations in the following form:

```

r = shape(theta);
r_theta = shape_theta(theta);

```

A tool needs to use automatic differentiation to obtain the required partial derivative. Dymola generates the derivative function in Modelica format:

```

function shape_theta
  input Real theta;
protected
  Real r[2];
public
  input Real R1 := 1 "Base circle radii";
  input Real R2 := 0.5 "Nose radii";
  input Real d := 2 "Centre distance";
protected
  constant Real pi := 3.14159265358979;
  Real fi0;
  Real fi1;
  Real fi2;
  Real L;
  Real x;
  Real y;
  Real thetamod;
  Real theta_d13 := 1;
public
  output Real r_d13[2];
protected
  Real fi0_d13, fi1_d13, fi2_d13;
  Real L_d13, x_d13, y_d13;
  Real thetamod_d13;
algorithm
  thetamod_d13 := theta_d13;
  thetamod := arctan2(sin(theta), cos(theta));
  fi0_d13 := 0;
  fi0 := arcsin((R1-R2)/d);
  if (thetamod > 0) then
    fi1_d13 := -(fi0_d13+thetamod_d13);
    fi1 := 0.5*pi-fi0-thetamod;
  else
    fi1_d13 := fi0_d13-thetamod_d13;
    fi1 := fi0-0.5*pi-thetamod;
  end if;
  fi2_d13 := -((d+R2*sin(fi0))*R2*fi0_d13*
    sin(fi0)+R2*cos(fi0)*R2*fi0_d13*cos(fi0))/
    ((R2*cos(fi0))^2+(d+R2*sin(fi0))^2);
  fi2 := arctan2(R2*cos(fi0), d+R2*sin(fi0));
  if (abs(thetamod) > 0.5*pi-fi0) then
    x_d13 := -R1*theta_d13*sin(theta);
    x := R1*cos(theta);
    y_d13 := R1*theta_d13*cos(theta);
    y := R1*sin(theta);
  elseif (abs(thetamod) >= fi2) then
    L_d13 := R1*fi1_d13*sin(fi1)/cos(fi1)^2;
    L := R1/cos(fi1);
    x_d13 := L_d13*cos(theta)-L*theta_d13*
      sin(theta);
    x := L*cos(theta);
    y_d13 := L_d13*sin(theta)+L*theta_d13*
      cos(theta);
    y := L*sin(theta);
  else

```

```

    L_d13 := -(d*thetamod_d13*simplesign(thetamod)*
      sin(abs(thetamod))+d^2*sin(abs(thetamod))*
      thetamod_d13*simplesign(thetamod)*
      cos(abs(thetamod))/sqrt(R2^2-
      d^2*sin(abs(thetamod))^2));
    L := d*cos(abs(thetamod))+sqrt(R2^2-
      d^2*sin(abs(thetamod))^2);
    x_d13 := L_d13*cos(theta)-L*theta_d13*sin(theta);
    x := L*cos(theta);
    y_d13 := L_d13*sin(theta)+L*theta_d13*cos(theta);
    y := L*sin(theta);
  end if;
  r_d13 := {x_d13, y_d13};
end shape_theta;

```

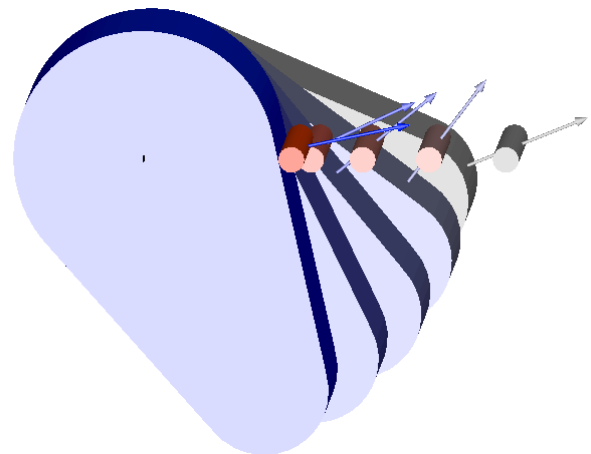
The *dist* variable can be used to define a spring acting when there is penetration:

```

F_n = if dist < 0 then k*(-dist) else 0;

```

Consider the follower connected to a mass which is connected to a spring and damper to ground and that the CAM is rotating at a fixed angular velocity. For high speeds, the follower will leave the nose and bounce back on the flank. Such a case is illustrated below in several frames from an animation.



It should be noted that the redeclared shape function is also used to define the parametric surface used for the animation.

If the contact model also contains damping, the derivative of the *shape_theta* function is also needed during index reduction. Such contact models are very stiff. In certain cases an idealized contact model with the constraint *dist=0* might be sufficient. In such a case, *F_n* is the constraint force. Index reduction will in that case require the second derivative of *shape_theta*. It is clear already by inspection of the automatically generated *shape_theta* function that automatic differentiation to obtain this function and its derivatives saves the modeler much tedious and error-prone work.

5 Conclusions

The paper demonstrates that extending Modelica with partial derivatives of functions is natural and solves a number of advanced modeling problems. Dymola automatically handles the differentiation of the partial derivatives of the functions thus reducing the work needed by the modeler, while preserving the efficiency of the generated simulation code.

References

- [1] Juedes D.W. (1991): **Taxonomy of Automatic Differentiation tools**, in Automatic Differentiation of Algorithms. SIAM
- [2] Char B.W. (1991): **Computer Algebra as a Toolbox for Program Generation and Manipulation**, in Automatic Differentiation of Algorithms. SIAM
- [3] Span R. (2000): **Multiparameter Equations of State, an accurate Source of Thermodynamic Property Data**. Springer Verlag.
- [4] Dynasim (2005): **Dymola User's Manual**, www.dynasim.se
- [5] Olsson H. (1993): **Applications of automatic and symbolic differentiation in numerical computations**. Master Thesis, Department of Computer Science, Lund Institute of Technology, Lund Sweden.
- [6] Rall L.B. (1981): **Automatic differentiation: Techniques and applications**. vol. 120 of Lecture Notes in Computer Science, Springer-Verlag.
- [7] Fischer H. (1991): **Special problems in Automatic Differentiation**, in Automatic Differentiation of Algorithms. SIAM.
- [8] Hannah J., Stephens R. C. (1972): **Mechanics of Machines**. Second edition, SI units, Edward Arnold.
- [9] Otter M., Elmqvist H., Lopez J.L. (2005): **Collision Handling for the Modelica MultiBody Library**. Modelica'2005 conference, Hamburg, March 7-8.

A Framework for Describing and Solving PDE Models in Modelica

Levon Saldamli*, Bernhard Bachmann†, Hansjürg Wiesmann‡, and Peter Fritzson*

Abstract

Currently, the Modelica language [3, 4] has limited support for solving partial differential equations (PDEs). There is ongoing work for introducing PDE support at the language level [5, 6]. This paper describes a prototype for describing PDE problems using the Modelica Language without any extensions, as an intermediate step. The goal is to define standard PDE models independent of specific domains, boundary conditions or any spatial discretization, and allow a user to reuse this without manual discretization. Modelica packages are used to define continuous domain boundaries, domains, and field variables over domains. Corresponding space discrete version of these packages are used to solve the space discretized PDE problem.

1 Introduction

A PDE problem can be specified and solved as follows using the approach in this paper (see Section 6 for more details):

```

model GenericBoundaryPoissonExample
  parameter BoundaryCondition.Data dirzero(
    bcType=BoundaryCondition.dirichlet,
    g=0);

  parameter BoundaryCondition.Data dirfive(
    bcType=BoundaryCondition.dirichlet,
    g=5);

  package myBoundaryP = MyGenericBoundary;
  parameter myBoundaryP.Data mybnd(
    bottom(bc=dirzero),
    right(bc=dirfive),
    top(bc=dirzero),
    left(bc=dirfive));

  package omegaP = Domain(
    redeclare package boundaryP=myBoundaryP);
  parameter omegaP.Data omega(boundary=mybnd);

```

*Dept. of Computer and Information Science, Linköpings universitet, Linköping, Sweden. {levsa,petfr}@ida.liu.se

†Fachhochschule Bielefeld, Fachbereich Mathematik und Technik, Studiengang Mathematik, Bielefeld, Germany. bernhard.bachmann@fh-bielefeld.de

‡ABB Corporate Research, CH-5405 Baden, Switzerland. hj.wiesmann@bluewin.ch

```

package PDE =
  PDEbhl.FEMForms.Equations.Poisson2D(
    redeclare package domainP = omegaP);
  PDE.Equation pde(
    domain=omega,
    g_rhs=1);
end GenericBoundaryPoissonExample;

```

First, two Dirichlet boundary conditions are declared, `dirzero` and `dirfive` with the right-hand side values 0 and 5, respectively. Then, a boundary component `mybnd` of type `MyGenericBoundary` (see Section 6) is declared, and the boundary conditions are assigned to the boundary components `bottom`, `right`, `top` and `left` of `mybnd`. A domain named `omega` is then declared using the boundary object `mybnd`. Finally, the PDE model is instantiated using `omega` as its definition domain.

Each declaration requires two actual declarations, one for the package and one for the data of the object, as explained in the following section.

2 The Package Approach

In order to use an object-oriented approach with polymorphism in Modelica, we use packages for defining new types such as `Domain`, `Field`, and `Boundary`. Each type `Type` contains at least a record called `Data`, containing the member variables needed in objects of type `Type`. The member functions are declared in the `Type` package. Each member function has at least one input argument, of the record type `Type.Data`. Thus, when calling member functions on objects, the objects data is passed as the input record argument. Declaring an object of type `Type` is implemented by declaring a local package, e.g. `typeP` which extends `Type` and possibly modifies parts of it, and then declaring a component of type `typeP.Data` which contains the data of the object of the modified type. This way, replaceable functions can be declared in a package, and replaceable packages extending these can exchange the functions as required. In other words, the packages define the class hierarchy for lookup of functions to work, and the data records define the object hierarchy storing the object instance data.

When types contain instances of other types, they declare a local package extending the other types package, and declare the objects data inside the `Data` record. For example, an equation model declares a domain and a field as follows:

```

model Equation "Poisson equation 2D"
  replaceable package domainP = Domain;
  parameter domainP.Data domain;

  package fieldP = Field (
    redeclare package domainP = domainP);
  parameter fieldP.Data u(domain=domain);
end Equation

```

This approach allows the equation model to be reused with any domain without changing other parts of the model. The package `fieldP` redeclares the replaceable domain package in the `Field` package. This way, the package hierarchy is correctly set up. The actual data records are declared separately, in order to build the object hierarchy of the model. The record `u` has the type `fieldP.Data` and will contain the correct domain data type from the given package `domainP`. The domain data must be initialized with the local values though, which is done with the modification when declaring the record `u`. When the domain is to be discretized, the shape function of its boundary package is called. Since the boundary package of the domain package is replaced when the domain is declared, the correct shape function is called. This is handled automatically through the package `DiscreteDomain`, explained in Section 5.1.1, which is declared in the discrete parts of the equation models.

The drawback of this approach is that each instantiation requires definition of a local package extending the type package, together with a declaration of the `Data` record of the local package. The advantage is that the local package can be declared as replaceable, and the correct version of the package will be used without knowing the type in advance.

3 Continuous Model Description

This section describes the packages used for continuous model description of domains and fields. These are `Boundary`, `Domain`, and `Field`, which are discretization independent information needed for the PDE problem. An overview of the packages in the framework is shown in Figure 1.

The geometry of a domain is described using continuous parametric curves, which is a fairly general representation and is easy to discretize. Each domain object contains a boundary object describing its boundary, i.e., the boundary defines the domain. The direc-

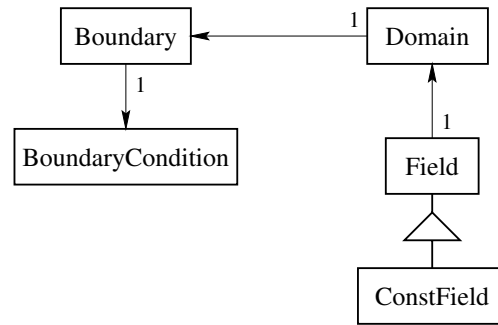


Figure 1: Overview of the packages for continuous domain and field description. `ConstField` inherits `Field`, i.e., it is a field with a known value (time-dependent or time-independent). Arrows represent aggregate, e.g. a domain object contains one boundary object, which contains one boundary condition object.

tion of the parametric curve representing the boundary decides on which side of the boundary the domain resides. Usually, the domain is on the left of the boundary, i.e., the curve is followed in counter-clockwise direction around a point in the domain.

3.1 Boundary Definition

A boundary contains a shape function representing the parametric curve defined for parameters in the range $[0, 1]$. The shape function can be seen as a mapping from a real value, the parameter, to the coordinate vector x . In two dimensions, one parameter suffices, in three dimensions, two parameters are needed. The specific return value of the shape function has the type `BPoint`, which is a point with additional information about the boundary conditions in that point.

The base package for all boundary types is the package called `Boundary`:

```

package Boundary
  replaceable function shape
    input Real u;
    input Data data;
    output BPoint x;
  end shape;

  replaceable record Data
    parameter BoundaryCondition.Data bc;
  end Data;
end Boundary;

```

The formal parameter `data` to the shape function contains the actual data of the specific boundary object. The record `BPoint`, representing a point with boundary condition information, is defined as follows:

```

type BPoint = Real[3] "x, y and boundary part index";

```

Here, index 1 and 2 represent the coordinates in two-dimensions, while the third value is the boundary con-

dition index, needed by the discretization and solution steps.

3.2 Domain Definition

A base domain type called `Domain` is declared with a boundary instance defining the actual geometry of the domain:

```
package Domain
  replaceable package boundaryP = Boundary
  extends Boundary; // base class restriction

  replaceable record Data
    parameter boundaryP.Data boundary;
  end Data;

  function discretizeBoundary
    input Integer n;
    input boundaryP.Data d;
    output BPoint p[n];
  algorithm
    for i in 1:n loop
      p[i, :] := boundaryP.shape((i - 1)/n, d);
    end for;
  end discretizeBoundary;
end Domain;
```

The restriction specifies that if the package `boundaryP` is replaced, the replacing package must be a subtype of `Boundary`.

The function `discretizeBoundary` must reside in the `Domain` package in order that the correct shape function is called, depending on the replaceable package `boundaryP`. The discretization simply calculates a given number of points uniformly distributed on the boundary. The data record of the domain contains the boundary record, which is the actual data record of the selected boundary type.

3.3 Fields

A field represents a mapping from a domain to scalar or vector values. The domain is declared as a replaceable package, which can be replaced by a package extending the `Domain` package described in Section 3.2. The replaceable type `FieldType` determines the value type of the field. The data record contains the data of the domain:

```
package Field
  replaceable type FieldType = Real;
  replaceable package domainP = Domain
  extends Domain; // base class restriction

  replaceable record Data
    parameter domainP.Data domain;
  end Data;

  replaceable function value
    input Point x;
    input Data d;
    output FieldType y;
  algorithm
    y := 0;
```

```
  end value;
end Field;
```

The function `value` represents the mapping, which can be defined when specifying fields with known values. Fields with unknown values that must be solved for during simulation may use value functions that interpolate the values for given coordinates.

3.3.1 A Field Example

An example showing a field with time-constant values follows:

```
model FieldExample
  function myfieldfunc
    input Point x;
    input myFieldP.Data d;
    output myFieldP.FieldType y;
  algorithm
    y := cos(2*PI*x[1]/6) + sin(2*PI*x[2]/6);
  end myfield;

  package omegaP =
    Domain (redeclare package boundaryP=Circle);
  package myFieldP =
    Field (redeclare package domainP=omegaP,
          redeclare function value=myfieldfunc);

  parameter Circle.Data bnd(radius=2);
  parameter omegaP.Data omega(boundary=bnd);
  parameter myFieldP.Data myfield(domain=omega);
end FieldExample;
```

The field function `myfieldfunc` defines the mapping from the space coordinates to the field values of type `Real`.

3.4 Included Boundaries

Some predefined boundaries can be found in the package `Boundaries`. All these packages extend the basic package `Boundary`. Therefore the data records in each boundary contain the parameter `bc` of type `BoundaryCondition.Data`, containing the boundary condition information. Boundary conditions are described in Section 4.3. An overview of the included boundaries can be seen in Figure 2. They are also briefly described in the following sections.

3.4.1 Line

`Line` is a straight line defined by two points, the start and the end points of the line. The data record of `Line` follows:

```
redeclare record extends Data
  parameter Point p1;
  parameter Point p2;
end Data;
```

The shape function simply interpolates the points linearly between the end points:

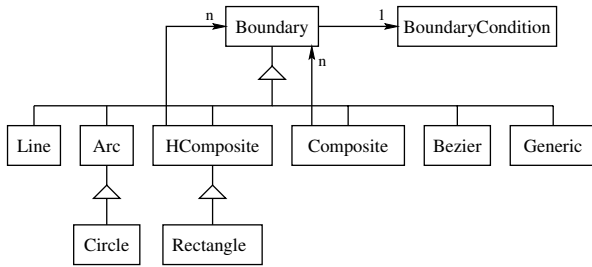


Figure 2: Predefined boundaries contained in the package `Boundaries`. `Composite` is a boundary consisting of boundary parts of different types. `HComposite` (homogeneous composite) consists of boundary parts of the same type. `Generic` is a boundary type that can represent the other concrete boundary types and is used in the `Composite` boundary.

```

redeclare function shape
  input Real u;
  input Data d;
  output BPoint x;
algorithm
  x[1:2] := d.p1 + u*(d.p2 - d.p1);
  x[3] := d.bc.index;
end shape;
  
```

The boundary condition index is passed through to the points on the boundary.

3.4.2 Arc

An arc is part of a circular boundary with given start and end angles around a center and with a given radius:

```

redeclare record Data
  extends Boundary.Data;
  parameter Point c={0,0};
  parameter Real r=1;
  parameter Real a_start=0;
  parameter Real a_end=2*pi;
end Data;
  
```

Default values for the parameter gives a full circle. The shape function calculates the position using `sin` and `cos` functions:

```

redeclare function shape
  input Real u;
  input Data d;
  output BPoint x;
protected
  Real a=(d.a_end - d.a_start);
algorithm
  x[1:2] := d.c + d.r*{cos(d.a_start + a*u),
                    sin(d.a_start + a*u)};
  x[3] := d.bc.index;
end shape;
  
```

3.4.3 Circle

A circle is simply defined by extending `Arc` and giving the angles for a full circle:

```

package Circle
  extends Arc(Data(a_start=0, a_end=2*pi));
end Circle;
  
```

3.4.4 Rectangle

A rectangle declares four lines as components, with the names `bottom`, `right`, `top` and `left`. For example `bottom` is declared as follows:

```

parameter Line.Data bottom(
  p1=p,
  p2=p + {w,0},
  bc(index=1, name="bottom"));
  
```

The parameters of the rectangle are `p`, `w` and `h`, representing the bottom left corner, the width and the height, respectively.

The rectangle class extends the `HComposite` package, which is a container for several boundary parts of the same type, as described below. The `Rectangle` package is defined as follows:

```

package Rectangle
  extends HComposite(
    redeclare package PartType = Line);

  redeclare record
  extends Data(bnddata(
    n=4,
    parts={bottom, right, top, left}));
  parameter Line.Data bottom(
    p1=p,
    p2=p + {w,0},
    bc(index=1, name="bottom"));
  // right, top and left defined similarly
end Data;
end Rectangle;
  
```

Hence, `bnddata` is a data record inside the rectangle record, with the parts initialized to the vector containing the four declared lines, and as the `PartType` declared as `Line`, accordingly.

3.4.5 Bézier

The Bézier boundary package uses a number of control points given as parameters to calculate the coordinates of the points on a Bézier curve, using De Casteljau's Algorithm [2]. The data record for `Bezier` package follows:

```

redeclare record extends Data
  parameter Integer n=1;
  parameter Point p[n];
end Data;
  
```

The shape function implements the algorithm for calculating the coordinates of a point on the curve, given the parameter `u`:

```

redeclare function shape
  input Real u;
  input Data d;
  output BPoint x;
protected
  
```

```

Point q[:]=d.p;
algorithm
  for k in 1:(d.n - 1) loop
    for i in 1:(d.n - k) loop
      q[i, :] := (1 - u)*q[i, :] + u*q[i + 1, :];
    end for;
  end for;
  x[1:2] := q[1, :];
  x[3] := d.bc.index;
end shape;

```

3.4.6 Generic

The boundary package `Generic` is needed in order to define composite boundaries containing boundary parts of different types. Since there are no pointers or union types in Modelica, it is not possible to declare a container for boundary parts where each part can be a subclass of `Boundary` which is not known at the time of library development. Hence, the `Generic` package contains an `enum` parameter deciding the type of the boundary part, and data records for each of the existing types that can be selected. This leads to a lot of overhead, since only one of the records are actually used, but unused parameters are optimized away during the compilation and this does not affect the resulting simulation code. In future implementations, union types or other solutions for polymorphism might allow more efficient implementation of generic boundary types. The enumeration type and the data record for the `Generic` boundary type follows:

```

type PartTypeEnum = enumeration (
  line ,
  arc ,
  circle ,
  rectangle );

redeclare replaceable record Data
  parameter PartTypeEnum partType;
  parameter Line.Data line;
  parameter Arc.Data arc;
  parameter Circle.Data circle;
  parameter Rectangle.Data rectangle;
end Data;

```

Because of lack of polymorphism, e.g. virtual functions, the shape function must check the enumeration variable and call the correct shape function:

```

redeclare function shape
  input Real u;
  input Data d;
  output BPoint x;
algorithm
  if d.partType==PartTypeEnum.line then
    x := Line.shape(u, d.line);
  elseif d.partType==PartTypeEnum.arc then
    x := Arc.shape(u, d.arc);
  elseif d.partType==PartTypeEnum.circle then
    x := Circle.shape(u, d.circle);
  elseif d.partType==PartTypeEnum.rectangle then
    x := Rectangle.shape(u, d.rectangle);
  end if;
end shape;

```

3.4.7 Composite

The `Composite` boundary simply uses a given number of `Generic` boundaries to build a complete boundary using parts of different types:

```

package PartType = Boundaries.Generic;

redeclare replaceable record extends Data
  parameter Integer n=1;
  parameter PartType.Data parts[n];
end Data;

```

The shape function simply calls the shape function in the `Generic` boundary package, using the index calculated by dividing the formal parameter `u` uniformly among the existing parts:

```

redeclare function shape
  input Real u;
  input Data d;
  output BPoint x;
protected
  Real s=d.n*u;
  Integer is=integer(s);
algorithm
  x := PartType.shape(s - is, d.parts[1 + is]);
end shape;

```

Here, `is` contains the part index corresponding to the value of the formal parameter `u`, and `s-is` is the new parameter value scaled to map to the parameter range of that particular boundary part. For example, if the shape function is called for a boundary containing four parts with $u = 0.8$, the value of `is` will be $integer(4 * 0.8) = 3$ and the value of `s-is` will be $4 * 0.8 - 3 = 0.2$, mapping to the `u` value on the fourth boundary part.

`HComposite` is a simplified version of the `Composite` boundary, containing only parts of the same type.

4 Equation Models

The `Equation` models contain all the different components of the PDE model, and handle the spatial discretization and the declaration of the discrete model equations. The continuous components of the model, i.e., the domain, its boundary, the boundary conditions and the field, are declared here. Their discrete counterparts are declared and initialized automatically from the continuous components, using given discretization parameters. The spatial discretization is done by calling the finite element solver, which can be implemented in Modelica, or an external solver called through the Modelica external function interface. The declared equations use the spatially discretized model.

4.1 The Poisson Equation

The Poisson equation is a simple example of a stationary (time-independent) model. In differential form, the equation is

$$-\nabla \cdot (c \nabla u) = f \quad \text{in } \Omega \quad (1)$$

where u is the unknown field, c is a space-dependent coefficient, f is the source term and Ω is the domain.

4.2 The Diffusion Equation

The diffusion equation for a field u is:

$$\frac{\partial u}{\partial t} - \nabla \cdot (c \nabla u) = f \quad \text{in } \Omega \quad (2)$$

where c is a space-dependent coefficient, f is the source term and Ω is the domain.

4.3 Boundary conditions

In both cases the boundary conditions may be Dirichlet, Neumann or mixed. The Dirichlet boundary conditions is used where the value of the unknown field is known on the boundary:

$$u = g \quad \text{on } \Omega \quad (3)$$

The Neumann boundary condition is used when the value of the normal derivative of the field is known on the boundary:

$$\frac{\partial u}{\partial n} = g \quad \text{on } \Omega \quad (4)$$

The mixed boundary condition, also called the Robin boundary condition, contains both the value of the field and the normal derivative:

$$a \frac{\partial u}{\partial n} + bu = g \quad \text{on } \Omega \quad (5)$$

5 Discretization

So far only the continuous parts of the packages have been discussed. These are independent of the discretization, and thus also the solution method, e.g. the finite element method or the finite difference method. The method for discretization of the domain depends on which solution method is used. The finite element package is described in the following section. Packages for the finite difference method exist for an earlier prototype of the framework. Also, packages for the finite volume method are being considered.

5.1 The Finite Element Package

For the finite element solver, the domain is represented by a triangular mesh. The mesh generator used in this work requires a polygon describing the boundary of the domain as input. This polygon is generated by discretizing the domain boundary using the shape function. A simple discretization function sampling a given number of points uniformly on the boundary is implemented as follows:

```
function discretizeBoundary
  input Integer n;
  input boundaryP.Data d;
  output BPoint p[n];
algorithm
  for i in 1:n loop
    p[i,:] := boundaryP.shape((i-1)/n, d);
  end for;
end discretizeBoundary;
```

The resulting polygon is given to the mesh generator `bang` [1]. The triangulation is then imported to `Modelica`. Figure 3 shows the overview of the packages used in this process.

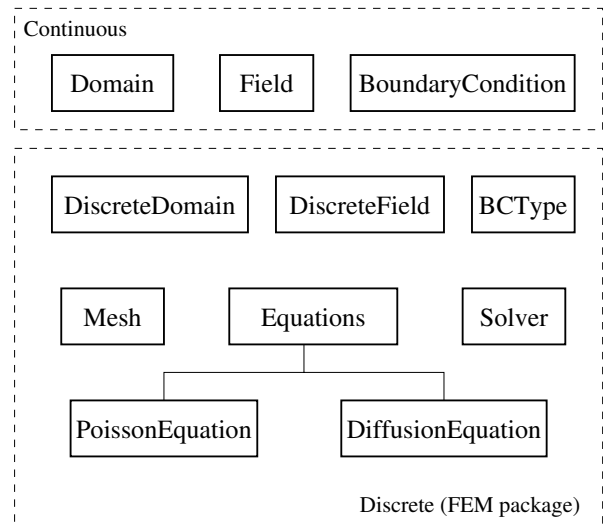


Figure 3: Packages involved in the discretization using the finite element method. The user has only to deal with the continuous part when using the equation packages.

The complete discretization and solution process is depicted in Figure 4. The external stiffness matrix calculation can be exchanged with internal code, i.e., functions implemented in `Modelica`. A prototype implementation in `Modelica` exists for discretization of the Poisson equation with homogeneous Dirichlet boundary conditions.

5.1.1 DiscreteDomain

`DiscreteDomain` is the discrete version of `Domain`. It contains a replaceable package `domainP`, representing the continuous version of the domain.

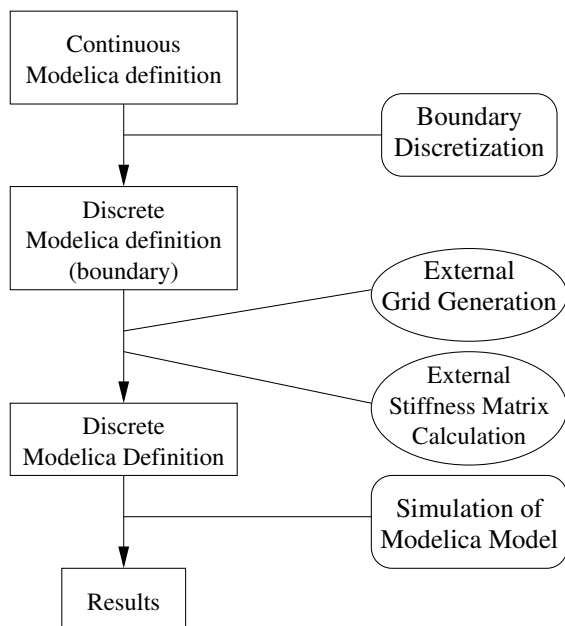


Figure 4: Solution diagram. The boxes on the left show the data flow. The rounded boxes show tools implemented in Modelica, and ellipses show external tools.

The discretization is done automatically, once `DiscreteDomain` is declared with a given `Domain` package. `DiscreteDomain` is defined as follows:

```

package DiscreteDomain
  replaceable package domainP = Domain
    extends Domain; // base class restriction

  replaceable record Data
    parameter Integer nbp;
    parameter domainP.Data domain;
    // A parameter to the mesh generator
    // specifying detail level, lesser means
    // more triangles
    parameter Real refine=0.7;

    // Array of discrete points on the boundary
    parameter BPoint boundary[nbp]=
      domainP.discretizeBoundary(nbp,
        domain.boundary);

    parameter Mesh.Data mesh(
      n=size(boundary, 1),
      polygon=boundary[:, 1:2],
      bc=integer(boundary[:, 3]),
      refine=refine);
    parameter Integer boundarySize=
      size(boundary, 1);
  end Data;
end DiscreteDomain;

```

The actual mesh generation is done when the `mesh` component is instantiated by the compiler, i.e., the `Mesh` package contains the actual calls to the external mesh generator.

5.1.2 DiscreteField

The package `DiscreteField` is incapsulates the conversion of a continuous field to a discrete field, us-

ing a given discrete domain. A discrete field contains two separate arrays of discrete points in the domain, one array containing the unknown values and one containing the known values, e.g. from given boundary conditions. This representation corresponds to the representation used in Rheolef [7], in order to simplify the solver interface. Both arrays are indirect, e.g. they contain indices of the actual points in the mesh representation. The `DiscreteField` package is defined as follows:

```

package DiscreteField
  replaceable package fieldP = Field;
  replaceable package ddomainP = DiscreteDomain;

  replaceable record Data
    parameter ddomainP.Data ddomain;
    parameter fieldP.Data field;
    parameter FEMSolver.FormSize formSize;
    parameter Integer u_indices[formSize.nu];
    parameter Integer b_indices[formSize.nb];
    fieldP.FieldType val_u[formSize.nu](
      start=zeros(formSize.nu));
    fieldP.FieldType val_b[formSize.nb];

    parameter Integer fieldSize_u=size(val_u, 1);
    parameter Integer fieldSize_b=size(val_b, 1);
  end Data;

end DiscreteField;

```

Here, the default start values for the unknowns are set to zeros. This value is overridden in the discrete parts of the equation models, for appropriate initial value setting. `FormSize` contains the sizes of the two arrays of discrete values, and is imported from the external solver since the sizes depend on the boundary conditions actually used in the model. Basically, Dirichlet and mixed boundary conditions decides the number of known variables.

5.1.3 Equation Discretization

The spatial derivatives in the equations are discretized using the external solver Rheolef [7], which is automatically called from the equation models through external functions. Rheolef performs the assembling of the matrix needed for the space discrete DAE system. The result of the assembly is a coefficient matrix for the unknown field values at the discrete points of the domain. The resulting matrices are imported to Modelica through external functions and used in the actual equations in the equation models. The final, possibly time-dependent, equation system, is simulated in Dymola. An example solved using this framework is shown in the following section.

6 Example

The result of the discretization of the equation, i.e., the assembly step, is a coefficient matrix for the unknown field values at the discrete points of the domain. The discrete part can be completely handled by the equation model, hiding the details from the user, as shown in the example using the PoissonEquation model:

```

model GenericBoundaryPoissonExample
  import PDEbhl.Boundaries.*;
  import PDEbhl.*;

  parameter Integer n=40;
  parameter Real refine=0.5;
  parameter Point p0={1,1};
  parameter Real w=5;
  parameter Real h=3;
  parameter Real r=0.5;
  parameter Real cw=5;

  package myBoundaryP = MyGenericBoundary ;

  parameter myBoundaryP.Data mybnd(
    p0=p0,
    w=w,
    h=h,
    cw=cw,
    bottom(bc=dirzero),
    right(bc=dirfive),
    top(bc=dirzero),
    left(bc=dirfive));

  package omegaP = Domain(
    redeclare package boundaryP=myBoundaryP);
  parameter omegaP.Data omega(boundary=mybnd);

  parameter BoundaryCondition.Data dirzero(
    bcType=BoundaryCondition.dirichlet,
    g=0,
    q=0,
    index=1,
    name="dirzero");

  parameter BoundaryCondition.Data dirfive(
    bcType=BoundaryCondition.neumann,
    g=5,
    q=1,
    index=2,
    name="dirfive");

  parameter BoundaryCondition.Data bclist[:] =
    { dirzero,
      dirfive };

  package PDE =
    PDEbhl.FEMForms.Equations.Poisson2D
    (redeclare package domainP = omegaP);

  PDE.Equation pde(
    domain=omega,
    nbp=n,
    refine=refine,
    g0=1,
    nbc=size(bclist, 1),
    bc=bclist);
end GenericBoundaryPoissonExample ;

```

Here, two different boundary conditions are assigned to different parts of the boundary. The boundary used here is defined as follows:

```

package MyGenericBoundary
  extends Boundary ;

  redeclare record extends Data
    parameter Point p0;
    parameter Real w;
    parameter Real h;
    parameter Real cw;

    parameter Real ch=h;
    parameter Point cc=p0 + {w,h/2};

    parameter Line.Data bottom(
      p1=p0,
      p2=p0 + {w,0});
    parameter Line.Data top(
      p1=p0 + {w,h},
      p2=p0 + {0,h});
    parameter Line.Data left(
      p1=p0 + {0,h},
      p2=p0);

    parameter Bezier.Data right(
      n=8,
      p=fill(cc, 8) +
        { {0.0,-0.5},{0.0,-0.2},{0.0,0.0},
          {-0.85,-0.85},{-0.85,0.85},{0.0,0.0},
          {0.0,0.2},{0.0,0.5}
        } * { {cw,0}, {0,ch} });

    parameter Composite.Data boundary(
      parts1(line=bottom,
        partType=PartTypeEnumC.line),
      parts2(bezier=right,
        partType=PartTypeEnumC.bezier),
      parts3(line=top,
        partType=PartTypeEnumC.line),
      parts4(line=left,
        partType=PartTypeEnumC.line));
  end Data ;

  redeclare function shape
    input Real u;
    input Data d;
    output BPoint x;
  algorithm
    x := Composite.shape(u, d.boundary);
  end shape ;

end MyGenericBoundary ;

The basic contents of the Poisson2D equation model
used above is defined as follows:

package Poisson2D "Poisson problem 2D"
  package uDFieldP = DiscreteField(
    redeclare package ddomainP = ddomainP,
    redeclare package fieldP = uFieldP);

  uDFieldP.Data fd(
    ddomain=ddomain,
    field=uField,
    formsize=formsize,
    u_indices=u_indices,
    b_indices=b_indices,
    val_u(start={1 for i in 1:formsize.nu}));
  equation
    laplace_uu * fd.val_u
      = mass_uu*g_rhs.val_u + mass_ub*g_rhs.val_b
      - laplace_ub*fd.val_b;
    fd.val_b = bvals; // known boundary values
  end Equation;
end Poisson2D ;

```

The matrices `laplace_uu`, `mass_uu`, `mass_ub` and `laplace_ub` are retrieved from the external solver Rheolef. Also `bvals` is calculated by the external solver. For diffusion problems, additional matrices are retrieved for the coefficients for the time derivatives of the unknowns.

The plot of the simulation result can be seen in Figure 5. For comparison, same model is exported to and solved in FEMLAB. Figure 6 shows the result generated by FEMLAB. The triangulation of the domain in both cases can be seen in Figure 7.

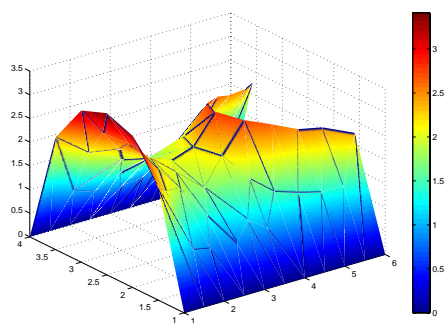


Figure 5: Results from solving the Poisson equation (steady-state) in Dymola.

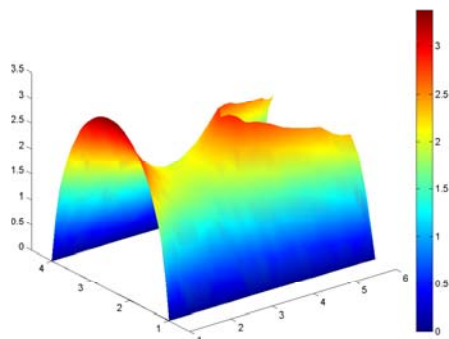


Figure 6: Results from solving the Poisson equation (steady-state) in FEMLAB.

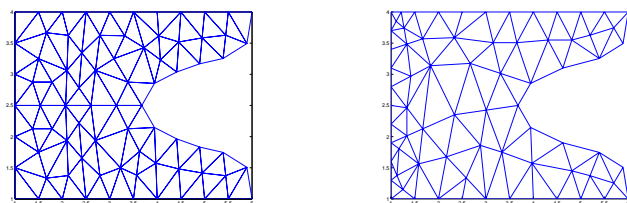


Figure 7: The meshes automatically generated from domain description in Modelica. Bamg mesh on the left, FEMLAB solution mesh on the right.

7 Conclusion and Future Work

The packages presented here give a general framework for easily defining general domains over which the predefined PDE models from the framework can be solved. New boundaries are easy to define using the existing boundaries as components, as shown in Section 6. Additional standard boundaries can also be added to the `Boundaries` package for future use.

New PDE models are also easy to add to the framework. Models that can be formulated using forms as described in the Rheolef User Manual [8] can be added to the framework by using the external function interface and implementing necessary extensions.

Further work is needed on the finite difference and the finite volume packages and adapt them to the current continuous definition framework. Also, the finite difference solver can be improved to support non-rectangular domains.

A simple extension of the framework is to include domains that use the existing standard boundaries. For example, a `CircularDomain` can be defined in the framework as follows:

```
package CircularDomain
  extends Domain(
    redeclare package boundaryP = Circle);
end CircularDomain;
```

Such a domain can be used directly when defining new problems, instead of declaring a general domain each time and replacing the boundary manually.

8 Acknowledgments

This work has been supported by Swedish Foundation for Strategic Research (SSF), in the VISIMOD project.

References

- [1] BAMG home page. <http://www-rocq.inria.fr/gamma/cdrom/www/bamg/eng.htm>.
- [2] Gerald Farin and Dianne Hansford. *The Essentials of CAGD*. A K Peters, 2000.
- [3] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2003. <http://www.mathcore.com/drmodelica/>.
- [4] Modelica Association. *Modelica – A Unified Object-Oriented Language for Physical Systems*

Modeling - Language Specification Version 2.1,
Jan 2004. <http://www.modelica.org/>.

- [5] L. Saldamli and P. Fritzson. Field Type and Field Constructor in Modelica. In *Proc. of SIMS 2004, the 45th Conference on Simulation and Modelling*, Copenhagen, Denmark, September 2004.
- [6] Levon Saldamli. *PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations*. Linköping Studies in Science and Technology, Licentiate Thesis No. 990, December 2002.
- [7] Pierre Saramito, Nicolas Roquet, and Jocelyn Etienne. Rheolef home page. <http://www-lmc.imag.fr/lmc-edp/Pierre.Saramito/rheolef/>, 2002.
- [8] Pierre Saramito, Nicolas Roquet, and Jocelyn Etienne. *Rheolef users manual*. 2002. <http://www-lmc.imag.fr/lmc-edp/Pierre.Saramito/rheolef/usrman.ps.gz>.

A Task Merging Technique for Parallelization of Modelica Models

Peter Aronsson, Peter Fritzson

PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, S-581 83 Linköping, Sweden

Abstract

This paper presents improvements on techniques of merging tasks in task graphs generated in the ModPar automatic parallelization module of the OpenModelica compiler. Automatic parallelization is performed on Modelica models by building data dependency graphs called task graphs from the model equations. To handle large task graphs with fine granularity, i.e. low ratio of execution and communication cost, the tasks are merged. This is done by using a graph rewrite system(GRS), which is a set of graph transformation rules applied on the task graph. In this paper we have solved the confluence problem of the task merging system by giving priorities to the merge rules. A GRS is confluent if the application order of the graph transformations does not matter, i.e. the same result is gained regardless of application order.

We also present a Modelica model suited for automatic parallelization and show results on this using the ModPar module in the OpenModelica compiler.

1 Introduction

Parallel computers have been used in simulations for a long time. In fact, many of the large simulation applications are driving the parallel computer industry, like modeling and simulation of atomic explosions, or modeling and simulation for weather forecasting. These models are typically hand written for dedicated parallel computers. Modeling of such systems requires both knowledge of the modeling domain and knowledge in parallel programming. Thus, such models are mostly used by experts and the models tend to be used for a long period of time, since it is too expensive to change them.

In this paper we instead present techniques that enable a fully automatic approach to parallel simulation. We have developed an automatic parallelization tool for Modelica that can translate a Modelica model into a platform independent parallel simulation program. By having a fully automated process of producing the parallel simulation code, parallel simulation is opened up

to a new set of users, with little or no knowledge of parallel programming or even parallel computers.

Our parallelization tool is plugged into the OpenModelica compiler developed at the Programming Environments Laboratory (PELAB) at Linköping University. Figure 1 presents an overview of the components of the OpenModelica compiler and the parallelization tool which is called ModPar. The OpenModelica compiler reads Modelica models and produces a set of variables, equations, algorithms, blocks, etc. This is fed into the ModPar module which performs a set of optimizations on the equations. First, simple algebraic equations on the form $a=b$ are removed, which can substantially reduce the number of equations and variables of the system.

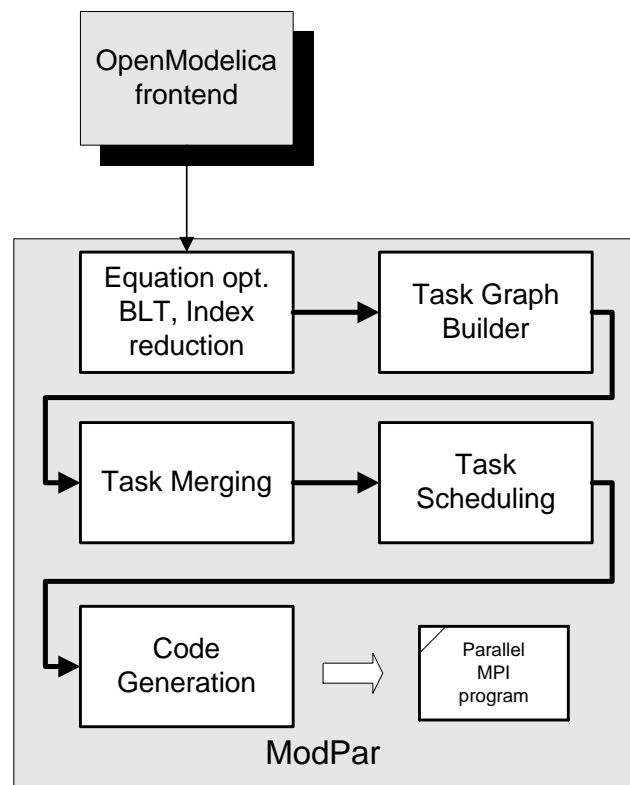


Figure 1. The ModPar Architecture.

The next optimization performed on the equations is the equation sorting. Equations are sorted in a Block Lower Triangular(BLT) form, resulting in a set of

equation blocks, where each block consists of one or more equations that need to be solved simultaneously.

In conjunction with sorting the equations, index reduction using dummy derivatives is applied[6]. Index reduction is used on high index systems of equations, where some equations need to be differentiated in order to solve the system. The *index* of a system corresponds to how many times some equations needs to be differentiated before the set of equations can be transformed into an ODE (also called the underlying ODE).

A *task graph* is built, based on the sorted BLT form. A task graph is a Directed Acyclic Graph (DAG), with costs associated with edges and nodes. It is described by the tuple $G = (V, E, c, \tau)$ where

- V is a set of vertices (nodes), i.e. tasks in the task graph. A task is generated for each sub expression in the model equations. For instance, an addition between two scalar values ($a+b$) or a function call ($\sin(x)$) constitutes a task. In this paper tasks and nodes are used with the same meaning.
- E is a set of edges, which imposes a precedence constraint on the tasks. An edge $e = (v_1, v_2)$ indicates that node v_1 must be executed before v_2 and send data (resulting from the execution of v_1) to v_2 .
- $c(e)$ gives the communication cost of sending the data along an edge $e \in E$.
- $\tau(v)$ gives the execution cost for each node $v \in V$.

The immediate predecessors (or parents) of a node n are all nodes having an edge leading to the node n . They are denoted by $\text{pred}(n)$. The immediate successors (or children) of a node n are all nodes having an edge leading to it from node n . They are denoted by $\text{succ}(n)$. Similarly the predecessors of a node n is the transitive closure of $\text{pred}(n)$, i.e. the set of all tasks having a path leading to the node n . Analogously, the successors of a node n are all the tasks having a path leading to them from the node n . These sets are denoted $\text{pred}^m(n)$ and $\text{succ}^m(n)$ respectively.

Blocks containing more than one equation need to be solved before the task graph can be built. Such a block can either be a linear system of equations or a non-linear system of equations. For certain blocks the solution cannot be found at compile time and thus a numerical solver is integrated in the task graph itself. For example, the solution of a linear system of equations can be done in parallel, making the corresponding task possible to execute on more than one processor. Such tasks are referred to as *malleable tasks*.

The next step in the ModPar tool is to perform task merging and task clustering. Task clustering performs a mapping of tasks to virtual processors by forming clus-

ters of tasks. This means that tasks that belong to the same cluster have a communication cost of zero, while tasks between clusters still have their original communication cost. Task merging differs from task clustering in the sense that tasks of the task graph are collapsed into a single node that represents the complete computational work of the included tasks. The data packets sent to and from the merged task are also combined. The goal of a task-merging algorithm is to increase the granularity, i.e., the relation between communication and execution cost of the task graph. This paper presents improvements on a task-merging algorithm based on earlier work in [1].

The result from the Task Merging algorithm is a new task graph with a smaller number of tasks (with larger execution costs). This is fed into the task-scheduling algorithm that maps the task graph onto a fixed number of processors. Each task in the task graph is assigned a processor(s) and starting time(s).

The final stage in the ModPar module is code generation. The ModPar outputs simulation code with MPI (Message Passing Interface) calls[7] to send and receive code between processors. Processor zero runs the numerical solver. In each integration step, work is distributed to other slave processors, which then calculate parts of the equations and send the result back to processor zero. Model parameters are only read once from file and distributed to all processors at the start of the simulation.

The rest of the paper is organized as follows. Section 2 introduces the method of merging tasks using a graph rewrite system formalism. Section 3 presents a Modelica application example suitable for parallelization, followed by results in section four. Section 5 presents the conclusions of the work and section six shows how the work relates to other contributions.

2 Task Merging using Graph Rewrite Systems

In previous work we have proposed a task-merging algorithm based on a graph rewrite system (GRS). A

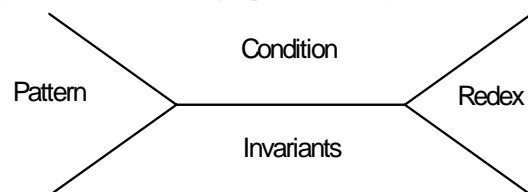


Figure 2. The X notation for GRS.

GRS is a set of graph transformation rules with a pattern, a condition, and a resulting sub-graph (called *redex*). We use a graphical notation (called the X-notation) depicted in Figure 2.

A GRS applies the transformation rules on the graph until there are no more matching patterns found in the graph. When this happens the GRS *terminates*. The termination of a GRS is an important property both theoretically and in practice. If it is not terminating, the GRS must be interrupted somehow in a practical implementation.

Our task merging rewrite rules are based on the condition that the *top level* of a task should not increase. The top level of a task is defined as the longest path from the task to a task without any ingoing edges, accumulating execution cost and communication cost along the path. The communication costs are described using two parameters, the bandwidth B and the latency, L . The communication cost of sending n bytes becomes $n/B + L$. The transformation rules, first presented in [2] are given below.

1. The first and simplest rewrite rule is given in Figure 3. It merges a parent task that has only one child with the child. This can always be performed, i.e., without any condition, since such transformation will not reduce the level of parallelism in the task graph.

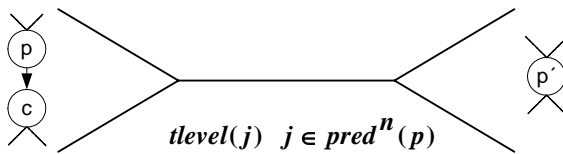


Figure 3. Merging of single children rule, called singlechildmerge.

2. The second rule handles *join nodes*, i.e., a task that has several incoming messages from a set of parent tasks, see Figure 4. The condition for this rule to apply is that the top level of task c does not increase when the transformation is performed. However, it is also necessary to make sure that *other successors* of the parents of the join node (p_{ij}) are not increasing their top levels. The rule therefore divides the parents into two disjoint sets, one that has successors fulfilling the condition and one that has successors increasing their top level by the merge and therefore not fulfilling the condition. The parents not fulfilling the condition are therefore not merged into the join task, c .

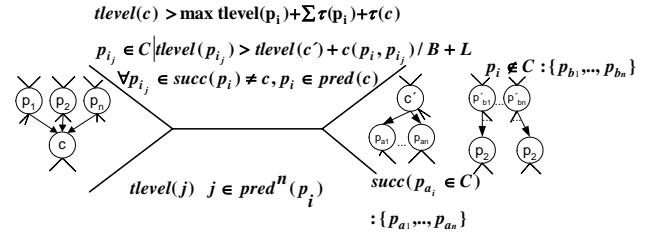


Figure 4. Rule of Merging of all parents to a task, called mergeallparents.

3. The third and final rewrite rule deals with *split nodes*. A split node is a node with several successors, or children. The transformation will replicate the split task and merge it with each individual successor task, c_i . However, the successor tasks can also have *other predecessors* for which the top level cannot be allowed to increase. Therefore, analogously as for the join task rewrite rule we also divide the successor tasks into two disjoint sets. The successor tasks that have other predecessors not increasing the top level are put in the set C . Thus, predecessors belonging to C are replicated and merged with the task c , while predecessors not belonging to C are left as they are.

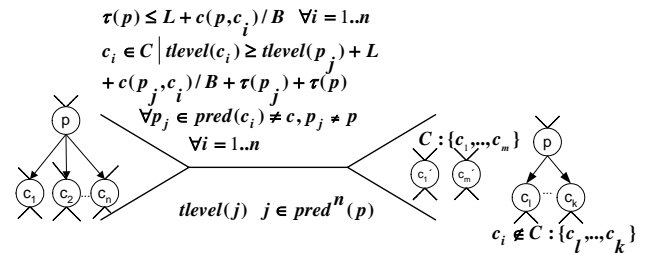


Figure 5. Replicating a parent and merging into each child task, called replicateparentmerge.

An unanswered question so far has been if the GRS is *confluent* or not. A confluent GRS gives the same resulting graph independently of the order of the applied rules. In earlier work, we investigated empirically whether the GRS was confluent, but now we have found a counter example that the rewrite rules are not confluent as they appear above. There are several alternatives to try to remedy this fact:

1. One could limit the order of matching of the patterns on the task graph. An idea of this is for instance to traverse the graph once in a top down fashion to prevent the confluence problem to occur. It is however not clear if this would work or not, without a more thorough investigation.

2. Another alternative is to instead use the simpler rewrite rules first presented in [2]. This approach might be taken for specific types of graphs, e.g. trees or forests, but in the general case, this is not sufficient. The simple rules did not succeed so well in reducing fine grained tasks graphs as produced by the task graph builder in ModPar.
3. A third, and the best practical alternative, is to give priorities to the rewrite rules. This means that a rewrite rule with a higher priority is always applied before other rules with lower priority. This will effectively prevent the GRS from being non-confluent, since only applications of transformations in priority order is allowed.

The priority order solution to the confluence problem was chosen in ModPar. The chosen priority is:

1. `singlechildmerge`
2. `replicateparentmerge`
3. `mergeallparents`

This means that the `singlechildmerge` rule has the highest priority and is always applied first. This rule is also the cheapest to apply since it does not have any condition, only a sub-graph pattern. Therefore, it makes sense to apply it with highest priority.

The second highest priority has the `replicateparentmerge` rule, thus giving the `mergeallparents` rule the lowest priority. The order between the last two rules is chosen so that rules limiting the amount of parallelism of the task graph are given lower priority. Since `mergeallparents` merges independent tasks (the successor of the parent), it reduces the amount of parallelism, which `replicateparentmerge` does not. Therefore, this order is chosen.

3 Application example

Lets consider a simple application example that can easily be scaled up using the `array of components` feature in Modelica. It uses the Modelica standard library and the one-dimensional `Rotational` package to create a flexible shaft. The shaft element is implemented as:

```

model ShaftElement "Element of a flexible
                    one dimensional shaft"
import Modelica.Mechanics.Rotational.*;1
        extends Interfaces.TwoFlanges;
        Inertia load;
        SpringDamper spring(c=500,d=5);

```

¹ Unqualified imports are not recommended to use. They are used here for space considerations.

```

equation
    connect(load.flange_b,
            spring.flange_a);
    connect(load.flange_a,flange_a);
    connect(spring.flange_b,flange_b);
end ShaftElement;

```

The `ShaftElement` model describes a one-dimensional shaft element with a spring and a damper. By instantiating this component as an array and connecting each array component to the next, we get a simple model of a flexible shaft.

```

model FlexibleShaft "model of a flexible
                    shaft"
import Modelica.Mechanics.Rotational.*;
        extends Interfaces.TwoFlanges;
        parameter Integer n(min=1) = 20 "number
        of shaft elements";
        ShaftElement shaft[n];
equation
    for i in 2:n loop
        connect(shaft[i-1].flange_b,
                shaft[i].flange_a);
    end for;
    connect(shaft[1].flange_a, flange_a);
    connect(shaft[n].flange_b, flange_b);
end FlexibleShaft;

```

Finally, we create a test model to test our shaft.

```

model ShaftTest
    FlexibleShaft shaft(n=20);
    Modelica.Mechanics.Rotational.Torque
    src;
    Modelica.Blocks.Sources.Step c;
equation
    connect(shaft.flange_a, src.flange_b);
    connect(c.outPort, src.inPort);
end ShaftTest;

```

The structural parameter `n` controls the number of element pieces of the model, i.e., the number of discretization points of the model. It is therefore directly proportional to the number of variables and equations of the model. Due to its simplicity and structure, it is suitable for parallelization.

4 Results

The confluence problem is successfully solved in this paper by introducing priorities on the task merging rules. These priorities makes the task merging GRS confluent, according to measurements made on a large set of random task graphs from the Standard Task Graph Set (STG)[10], as well as task graphs generated from the ModPar module.

The application example in section 3 can substantially be reduced in size but still reveal sufficient parallelism. When running the task-merging algorithm on the task graph produced from the example, it results in a set of independent tasks, which can then be allocated

to a set of processors in a simple load balancing manner, i.e., evenly distributing them among the processors. Thus, for this example, no scheduling is even required. This reduction is possible since the graph rewrite rules allow replication of tasks, such that dependencies between tasks of the task graph are completely removed.

Table 1 shows the increase of granularity² when applying the task merging for another Modelica example from the Thermofluid package. With realistic figures on bandwidth (B) and Latency (L), we see a substantial increase of granularity.

<i>Model</i>	<i>Granularity before merge</i>	<i>Granularity after merge</i>
PressureWave (B=1, L=100)	0.000990	0.106
PressureWave (B=1, L=1000)	0.0000990	0.0562

Table 1. **Granularity before and after Task Merging.**

The status of the parallelization tool is that we can generate C code with MPI calls for execution of parallel machines, such as the Linux cluster *monolith* at NSC (Swedish National Supercomputer Center). We have successfully executed smaller examples on this cluster computer but without obtaining any speedups. The application example in Section 3 can only be translated in reasonable time with about 9000 equations (using 1000 discretization points), which is a bit too small for obtaining sufficient speedups. In order to handle larger system of equations, the equation optimization and other parts of the compiler must be implemented in a more efficient way. In addition, the amount of work per state variable in the Flexible Shaft example is not so large, so in order to get better speedups, other applications must be considered.

5 Conclusions

We have proposed improvements on earlier work of merging tasks in a task graph using a graph rewrite system formalism. Earlier improvements made the task merging GRS non-confluent, thus giving different results depending of order of application. We proposed several alternative solutions to make the GRS confluent and have chosen and implemented the best-suited solution for our application area, parallelization of simulation code from Modelica models.

² The relation between communication and execution cost of the task graph.

The task merging technique is implemented in the ModPar module, a part of the OpenModelica compiler. It successfully reduces the number of tasks of task graphs built from Modelica simulation code to a suitable degree such that existing scheduling algorithms can succeed in producing parallel programs that give sufficient speedup.

6 Related Work

There is much work on scheduling of task graphs for multi-processors, like the DSC[11] algorithm, the TDS[4] algorithm or the Internalization algorithm[9], all working on unlimited number of processors, so called clustering algorithms. They all treat each task in the task graph as a non-preemptive atomic task, and do not consider merging of tasks. Therefore, they do not work well on very fine-grained task graphs.

There are other attempts to merge tasks, like the grain-packing algorithm[5]. The difference between this approach and ours is that our approach is iterative by nature and allows task replication.

Related work on parallelization of simulation code includes distributed simulation where the numerical solver is split into several parts, each handling a subset of the equations. The interaction between the subsystems is then delayed in time such that the subsystems becomes independent of each other in each time step. This division of the model equations into subsystems is implemented using a transmission line component in the system, giving the technique the name Transmission Line Modeling (TLM)[3].

Other related work on parallel simulation includes using parallel solvers, where the numerical solvers themselves are parallelized, like for instance Runge Kutta based solvers[8].

Acknowledgements

This work has been supported by the Swedish Foundation for Strategic Research (SSF), in the VISIMOD project and in the ECSEL graduate school, and by Vinnova in the GridModelica project.

References

- [1] P. Aronsson, P. Fritzson, Automatic Parallelization in OpenModelica, Proceedings of 5th EUROSIM Congress on Modeling and Simulation, Paris, France, 6-10 Sep 2004. ISBN 3-901608-28-1
- [2] P. Aronsson, P. Fritzson, Task Merging and Replication using Graph Rewriting, Tenth International Workshop on Compilers for Parallel Com-

- puters, Amsterdam, the Netherlands, Jan 8-10, 2003
- [3] Casella F. Maffezzoni C., The Transmission Line Modeling Method, EEE/OUP Series on Electromagnetic Wave Theory, 1995
 - [4] S. Darbha, D. P. Agrawal. Optimal Scheduling Algorithm for Distributed-Memory Machines. IEEE Transactions on Parallel and Distributed Systems, vol. 9(no. 1):87{94, January 1998.
 - [5] B. Kruatrachue. Static Task Scheduling and Grain Packing in Parallel Processor Systems. PhD thesis, Dept. of Electrical and Computer Engineering, Oregon State University, 1987.
 - [6] S.E. Mattsson, G. Söderlind, Index reduction in differential-algebraic equations using dummy derivative, Scientific Computing Vol. 14 , Issue 3 1993
 - [7] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.
 - [8] T. Rauber, G. Runger, Iterated Runge-Kutta Methods on Distributed Memory Multiprocessors. In Proceedings of First Aizu International Symposium on Parallel and Distributed Processing, pages 12-19. 1995.
 - [9] V. Sarkar. Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, Cambridge, MA, 1989.
 - [10] Standard Task Graph Set (STG), <http://www.kasahara.elec.waseda.ac.jp/schedule/>, accessed 2004-12-02.
 - [11] T. Yang, A. Gerasoulis. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. Transactions on Parallel and Distributed Systems, vol. 5(no. 9), 1994.

Session 2

Poster session

Some Results on Neutral Modelling of the Steel Continuous Casting Process

Dorel Aiordachioaie Mihai Munteanu Emil Ceanga
“Dunarea de Jos” University of Galati, Electrical Engineering Faculty,
Domneasca-47, Galati – 800008, Romania.

Emails: Dorel.Aiordachioaie@ugal.ro, mmunteanu@tcinf.ro, Emil.Ceanga@ugal.ro

Abstract

The modelling goal is to obtain a neutral representation of the process with enough information to generate qualitative information about the behaviour of the process. Such a representation must be able to change models on different levels among the user's world. The obtained models have physical structure and parameters, under declarative and neutral format, as the Modelica modelling language provides. The behaviour of the models is obtained by considering and properly modelling the basic phenomena running on different modelling levels. More sophisticated models could be obtained by adding more knowledge and information at the place where the model is used of.

Keywords: Process Modelling; Neutral modelling; Metamodeling; Continuous casting; Modelica.

1. Introduction

The considered process is steel continuous casting. Most previous advances in continuous casting modelling have been based on empirical knowledge gained from experimentation with the process. Such models are mainly equation based and describe only parts of the process. Meanwhile model exchange among different simulation environments is a strong and real need. No model can reveal all the phenomena running within a process.

The modelling goal is to obtain a neutral representation of the process. Such a model must have enough information to generate qualitative information about the behaviour of the process. A more sophisticated model could be obtained by adding more knowledge and information at the place where the model is used.

The model has a physical structure and is based on running phenomena, which have measurable parameters and physical meanings, e.g. temperature, pressure, volumetric flows. The modelling

framework is based on Modelica, which is a very promising standard in neutral modelling [1], [2], especial for very complex processes, like the considered process.

The physical process is described in section 2. Its description is made at the physical level and based on the involved phenomena. The methodology of modelling is presented using metamodeling concepts presented and described in section 3. Three basic sub-models are considered in this work, based on physical decomposition of the process: the ladle, the tundish and the cooling model. Each of these models is considered by describing and modelling separately for validation purposes. It is the scope of sections 4, 5 and 6. Finally, in the section 7, some simulation results are presented and discussed.

2. Description of the Process

In the continuous casting process, molten material (metal) is delivered from the bottom of a transfer vessel (the tundish) into a mold cavity. Here, the water-cooled walls of the mold extract heat to solidify a shell that contains the liquid pool. The shell is withdrawn from the bottom of mold at a “casting speed” that matches the inflow of metal, so that the process ideally operates at a steady state. Below the mold, water sprays extract heat from the surface, and the strand core eventually becomes fully solid when it reaches the “metallurgical length”.

Heat flow and solidification phenomena models are used for basic design and troubleshooting of this process. Heat transfer in the mold region is controlled mainly by heat conduction across the interface between the surface of the solidifying shell and the mold. In steel slab casting operations with mold flux, such models feature a detailed treatment of the interface, including heat, mass, and momentum balances on the flux in the gap and the effect of shell surface imperfections on heat flow [3]. Heat flow models which extend below the mold are needed for basic machine design to ensure that the

last support roll and torch cutter are positioned beyond the metallurgical length for the highest casting speed. Below the mold, air mist and water spray cooling maintain surface temperature of the strand, while the interior solidifies.

Continuous casting involves a staggering complexity of interacting phenomena at the mechanistic level. Some of the important phenomena include, [4], [5], [6], [7], [8], [9], [10], [11], and [12]:

- fully-turbulent, transient fluid motion in a complex geometry (inlet nozzle and strand liquid pool), affected by argon gas bubbles, thermal and solute buoyancies;
- thermodynamic reactions within and between the powder and steel phases;
- dynamic motion of the free liquid surfaces and interfaces, including the effects of surface tension, oscillation and gravity-induced waves, and flow in several phases;
- thermal, fluid, and mechanical interactions in the meniscus region between the solidifying meniscus, solid slag rim, infiltrating molten flux, liquid steel, powder layers, and inclusion particles;
- heat transport through the solidifying steel shell, the interface between shell and mold, (which contains powder layers and growing air gaps) and the copper mold;
- solidification of the steel shell, including the growth of grains and microstructures, phase transformations, precipitate formation, and microsegregation;

Because of this complexity, no model can include all of the phenomena at once. An essential aspect of successful model development is the selection of the key phenomena of interest to a particular modelling objective and by making of reasonable assumptions. The basic phenomena considered in this work are related to heat transfer among material's phases and flow of the processed material.

Mechanistic models are based on satisfying the laws of conservation of heat, mass, force and momentum in an appropriate domain with appropriate boundary conditions. In this work, each considered phenomenon is represented by term(s) in these governing equations, excepting the force and momentum.

Other phenomena can be ignored or incorporated using empirical constants, obtained through experimentation and model calibration.

3. The modelling methodology

Following the above hypothesis the phenomena from two domains were considered: thermal and fluid phenomena. In the thermal domain the considered phenomena are conduction and radiation. From the fluid domain, fluid flow is considered as effect of difference pressure. For each domain, ports (some time interfaces called) must be defined, in order to describe the quantitative behaviour of the process and to write mass and energy balances.

By metamodel is understood a model of the modelling methodology. From the methodology point of view two metamodels are presented in Fig.1 and Fig. 2. The first metamodel shows the highest point of view of the methodology. The process model is considered being an aggregation of physical model with one or more material models and one or more phenomena models. Material and phenomena models need properties models to compute the thermodynamic and transport properties. The physical model, in association with the properties model, generates constraints related to the behaviour of the model.

The second metamodel is closer to the structure of the model. The process model is composed of two basic models: a phenomena model and a mathematical model. The phenomena model describes the level of the modelling and the interaction among considered phenomena. The quantitative behaviour of the model needs a mathematical model, which depends on the considered phenomena. The mathematical model contains balance and constitutive equations. The interaction of phenomena and mathematical model is described by properly considering ports.

There are two types of ports: external (or static) to describe the input-output flow of the material and internal (or dynamic) to describe the phase transformations of the material during processing. There are two external models, in the sense they could be defined in an independent approach, possible in different (distributed) places and by different modellers. It is the material model and the geometry model.

The material model describes the thermodynamic properties and the interaction of phenomena running on different modeling levels. Based on the context, i.e. the composition of the devices that support the processed material, a model of constraints must be considered and defined.

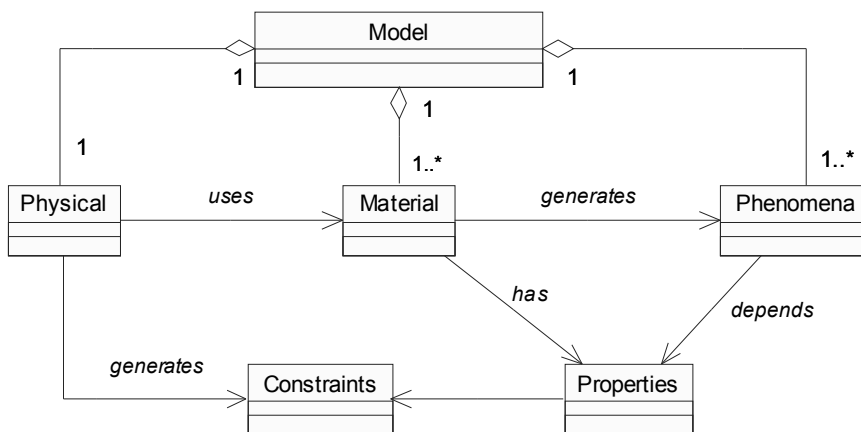


Figure 1: A *partial* metamodel of the process model

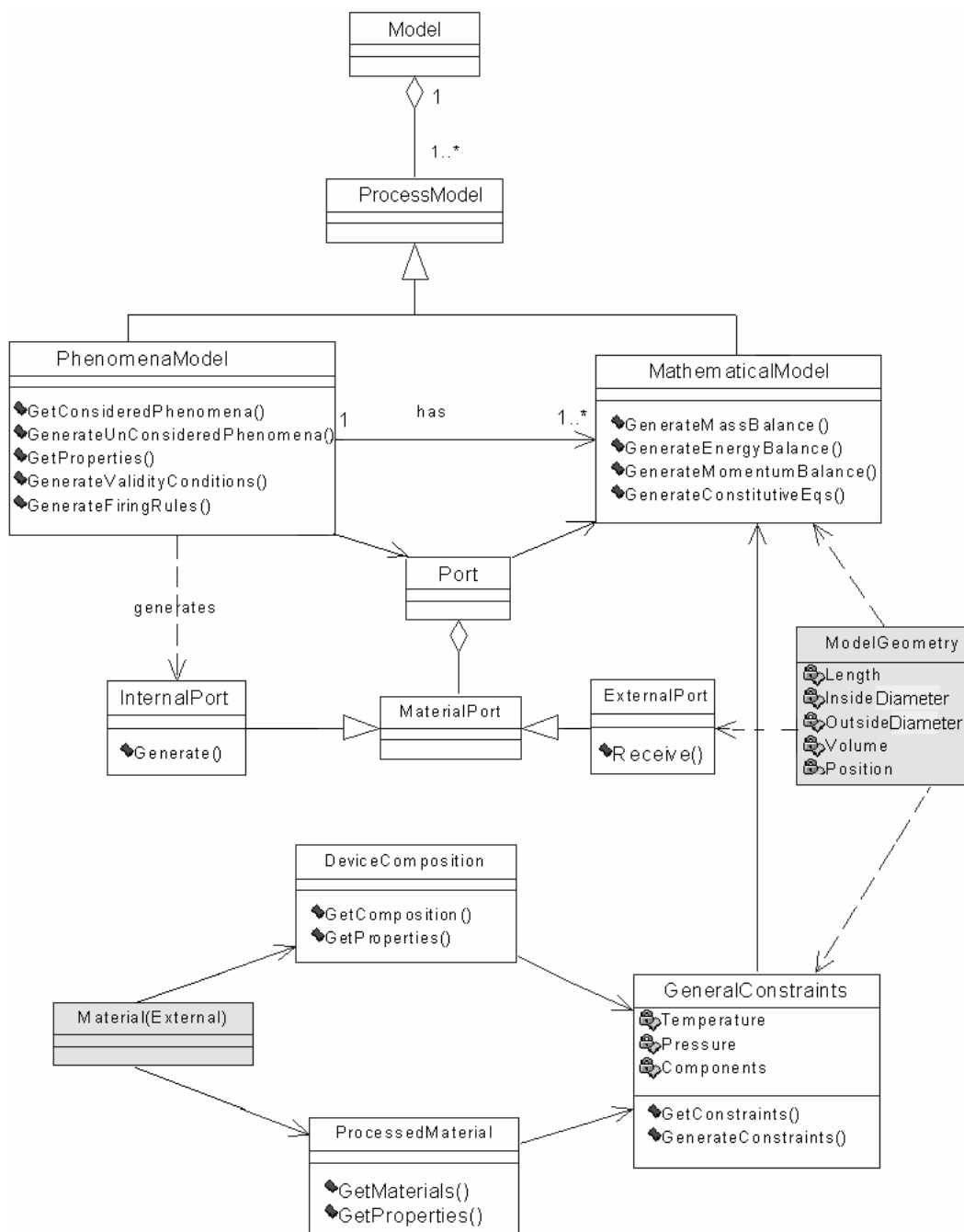


Figure 2: A more detailed metamodel

The constraints model is developed in correlation with the model of the geometry of the plant, where the processing of the material takes place.

Both metamodels are the base of the design a modelling tool, under computed aided modelling environment. Such a tool must assist the modeller to take in account the all the interactions and all necessary phenomena at the considered level of modelling.

4. The model of the ladle

The considered input variable in the ladle is the initial temperature T_0 of the processed material. The necessary phenomena to consider are related to time variation of the temperature inside and outside of the ladle and the variation of steel volume inside of the ladle, when the casting process starts. It is supposed that the steel is in liquid phase and no phase change is performed in the ladle.

Two sub-models cover the ladle: the processed material (steel) and the wall model. The transfer of thermal energy from ladle to outside of ladle is modelled by radiation phenomenon from the surface of the ladle to environment. A valve, conducted by a controller with proper drives, makes the control of the steel debit.

For the considered phenomena the following variables are necessary: temperature and heat flow rate for thermal domain; pressure, temperature and volume flow rate for fluid domain. The interfaces for such phenomena can be defined as:

```
connector PortFluid
  Pressure p;
  flow VolFlowRate qvol;
end PortFluid;
```

```
connector PortHeat
  CelsiusTemperature T;
  flow HeatFlowRate qheat;
end PortHeat;
```

```
connector PortThermoFluid
  Pressure p;
  CelsiusTemperature T;
  flow VolFlowRate qvol;
end PortThermoFluid;
```

The environment must be taken into consideration in order to show the behaviour of the high temperature

sources such as liquid steel. It is expected that the temperature of the environment to rise, near the space of the ladle. Such a model can be as

```
partial model Environment
  PortHeat ha;
  extends EnvironmentProperties;
  CelsiusTemperature T (start=30, min=30);
  parameter Volume vol(start=1000, min=0);
algorithm
  ha.T := T;
equation
  vol * rho * shcap * der(T) = ha.qheat;
end Environment;
```

In the EnvironmentProperties model the properties of the medium related to density, rho, and to specific heat capacity, shcap, are defined.

The phenomena from thermal domain are considered now. It is about of two main phenomena: radiation from the hot surface to another one with a lower temperature; the conduction of heat, which is specific to heat conduction in solid phases. The models can be as:

```
model Conduction
  PortHeat ha,hb;
  ThermalConductivity thermalcond (start = 1e-5);
  Thickness thick (start = 1);
  Area transfer_area (start=1);
  Real Rth(start=1, min=1E-6) "Thermal Res.";
algorithm
  Rth := thick / thermalcond / transfer_area;
  ha.qheat := (ha.T - hb.T) / Rth;
equation
  ha.qheat + hb.qheat = 0;
  // un-defined: thick, thermalcond, transfer_area;
end Conduction;
and
```

```
model Radiation
  PortHeat ha,hb;
  constant Real viewfactor = 0.1 "The view factor";
  constant Real sigma(final
unit="W/(m2.K4)")=5.6704e-8 "Stefan-Boltzman";
  Area transfer_area (start=1, min=0);
  Real Rth(start=1, min=1E-6);
algorithm
  Rth := 1 / sigma / transfer_area / viewfactor;
  ha.qheat := (ha.T^4 - hb.T^4) / Rth;
equation
  ha.qheat + hb.qheat = 0;
  // un-defined: transfer_area;
end Radiation;
```


The ladle model contains two sub-models: the wall and the steel. The wall model is composed mainly from a physical model, which defines the size and the composition of the wall, and the steel model. The wall model of the ladle can be as

```

model Wall
  PortHeat ha, hb;
  Conduction hcond;
  Volume vol (start=1);
  Density rho (start=1);
  Real shcap (start=1);
  CelsiusTemperature T (start=30, min=0)"In the
wall centre";
equation
  connect(ha, hcond.ha);
  connect(hb, hcond.hb);
  vol * rho * shcap * der(T) = abs(ha.qheat);
  //re-declare: vol, rho, shcap;
end Wall;

```

The model of the liquid steel is

```

model Steel
  PortHeat ha;
  PortThermoFluid tfb;
  extends SteelProperties;
  CelsiusTemperature T (start=1500);
  Volume vol(start=1, min=0);
  Area area (start=1, min=0.1);
  Real hout(start=1, min=0)"Enthalpy out-flow";
algorithm
  hout := tfb.qvol * shcap * rho * tfb.T;
  tfb.p := 1 + vol * rho * 9.8 / 101325 / area;
  tfb.T := T; ha.T := T;
equation
  der(vol) = tfb.qvol;
  rho * vol * shcap * der(T) = hout + ha.qheat;
end Steel;

```

Now, the ladle model can be defined as

```

model Ladle
  PortHeat ha;
  // interaction with env. near the steel surface;
  PortHeat hb;
  // interaction with tundish;
  PortThermoFluid tfc;
  // phenomena models:
  Wall wall;
  Steel steel;
  // inherits from:
  extends LadleGeometry;
  extends LadleMaterialProperties;
  Real h(start=1)"Steel height inside ladle";

```

```

algorithm
  h := steel.vol / ladle_area;
  wall.hcond.transfer_area := ladle_lateral_area;
  wall.hcond.thick := ladle_thick;
  wall.hcond.thermalcond := ladle_kL;
  wall.shcap := ladle_speccap;
  wall.vol := ladle_vol;
  wall.rho := ladle_rho;
  steel.area := ladle_area;
equation
  connect(wall.ha, ha);
  connect(wall.hb, steel.ha);
  connect(steel.ha, hb);
  connect(steel.tfb, tfc);
end Ladle;

```

The valve is considered as linear and with a very small resistance. The model is

```

model Valve "A model for valve"
  PortThermoFluid tfa, tfb;
  PortControl ca;
  parameter Real res(start=0.1, min=1e-6);
  Boolean off;
algorithm
  off = ca.u < 0;
equation
  tfa.qvol = if off then 0 else (tfa.p-tfb.p)/res;
  tfa.qvol + tfb.qvol = 0;   tfa.T = tfb.T;
end Valve;

```

5. The model of the tundish

The input variable in the tundish model is the volume flow and the steel height inside of the tundish is the controlled variable. The basic phenomena are related to the flow of materials from ladle to tundish. Because no phase transformations are taking place, for the material model is necessary to have only a model for the liquid phase. The interaction with the environment is made by the lateral surface of the tundish only.

The model and some submodels are presented in the Fig. 3. The test model is composed from the tandem ladle plus tundish, completed with material source, ("source"), valves for events ("valve1", "valve2"), and a load model ("load"). Inside of the dot lines polygons the sub-models of ladle and tundish were presented. In the following the models will be presented and discussed.

The ladle and the tundish have the same structure. Both use a material model (steel) and a wall model.

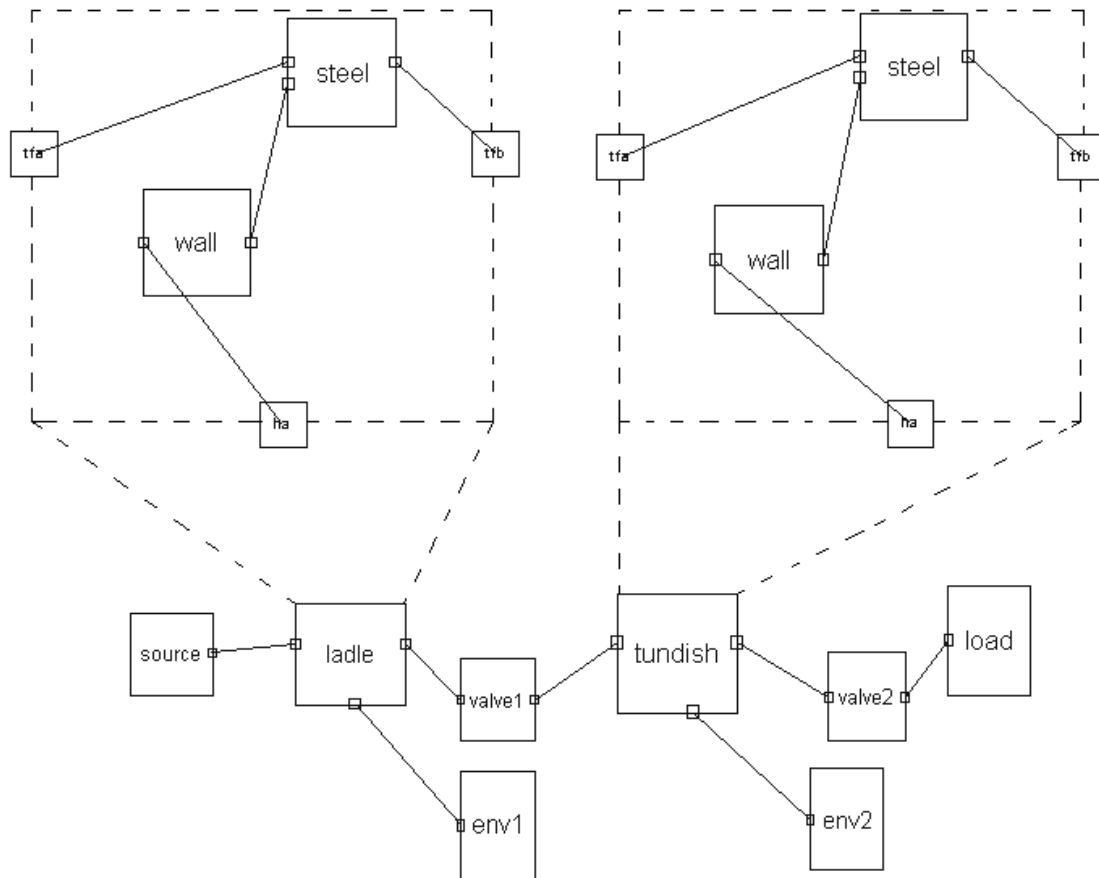


Figure 3: The test structure for the ladle-tundish tandem

The wall sustains the steel. The model of the processed material is called Steel. It has two thermo-fluid ports and one heat port. The first two are necessary to model the flow of the steel. The heat port is necessary to model the interaction with the wall.

The Steel model must be improved by adding a ThermoFluid port to model the input and the output flows of the material in tundish:

```

model Steel
  PortThermoFluid tfa,tfb;
  PortHeat ha;
  // parameters from steel properties model;
  parameter Density rho(start=1000);
  parameter SpecificHeatCapacity shcap(start=1e-5);
  CelsiusTemperature T (start=1500);
  Volume vol(start=1, min=0.1, max =10);
  Enthalpy hin(start=1, min=0)"Enthalpy in-flow";
  Enthalpy hout(start=1, min=0)"Enthalpy out-flow";
  Area press_area(start=1, min=0.1) "Area for grav.
pressure";
equation
  hin = tfa.qvol * shcap * rho * tfa.T;

```

```

hout = tfb.qvol * shcap * rho * tfb.T;
der(vol) = tfa.qvol + tfb.qvol;
vol * rho * shcap * der(T) = hout + hin + ha.qheat;
T = tfb.T; ha.T = T;
// un-defined: press_area, tfa.p, tfb.p; (context
dependency!);
end Steel;

```

The steel model is considered with volumic and material properties. The pressures on two ports will be defined later when the geometry of the vessel that sustains the steel is defined. It is about of context details. The model of the wall must define the geometry, the material properties and the heat conduction phenomena, as:

```

model Wall
  PortHeat ha, hb;
  // parameters from wall material properties;
  ThermalConductivity thermalcond (start=1e-5);
  // parameters from geometrical wall model;
  Thickness thick (start=1);
  Area hcond_transfer_area (start=1);
  Density rho(start=1);
  Volume vol (start=1);

```

```

SpecificHeatCapacity shcap(start=1);
Real Rth(start=1, min=1E-6) "Thermal resistance";
CelsiusTemperature T (start=100);
equation
  Rth = thick / thermalcond / hcond_transfer_area;
  ha.qheat = (ha.T - hb.T) / Rth;
  T = if ha.qheat > 0 then hb.T else ha.T;
  vol * rho * shcap * der(T) = ha.qheat + hb.qheat;
//un-defined: vol, rho, shcap, thermalcond, thick,
hcond_transfer_area;
end Wall;

```

The tundish model says that the tundish (object) an interaction between the wall and steel behaviour. The declarative model can be as

```

model Tundish
  PortThermoFluid tfa, tfb;
  PortHeat ha;
// parameters from tundish geometry model:
  parameter Height tundish_h (start=1);
  parameter Thickness tundish_thick (start=0.5);
  parameter Volume tundish_vol(start=5, min=0.1);
  parameter Diameter tundish_d (start=1, min=0.1);
  Area tundish_area (start=1, min=0.1)"Hor.Cross-
section area";
  Area tundish_lateral_area (start=1, min=0.1);
// parameters from tundish material properties:
  parameter ThermalConductivity tundish_kL;
  Real steelh(start=1)"Steel heigth inside tundish";
  Wall wall;
  Steel steel;
algorithm
  tundish_area := tundish_vol / tundish_h;
  tundish_lateral_area:=3.14*tundish_d*tundish_h;
  steelh := steel.vol / tundish_area;
  wall.hcond_transfer_area := 3.14 * tundish_d
* steelh;
  wall.thick := tundish_thick;
  wall.thermalcond := tundish_kL;
  wall.rho := 2000;
  wall.vol := 1;
  wall.shcap := 0.9;
  steel.press_area := tundish_area;
equation
// context definition:
  steel.tfb.p = steel.tfa.p + steel.vol * steel.rho * 9.8
/101325 / steel.press_area;
  steel.tfa.p =1;
  connect(wall.hb, ha);
  connect(wall.ha, steel.ha);
  connect(steel.tfa, tfa);
  connect(steel.tfb, tfb);
end Tundish;

```

6. The cooling model

The phenomena running after tundish can be considered as generated by a single type model: a cooling model of the liquid material. From a phenomenological point of view, it should model the transfer of energy from the liquid and the solid phase of the processed material to other material that acts as receptor or loads of the thermal energy.

The difficulty of the modelling problem is from the uncertainties generated by the material parameters, by the phenomena interactions during the casting process. More, all parameters are distributed and are temperature dependent. Moreover there are many material phase transformations such as:

- liquid-solid transformation of the processed material;
- liquid-solid-gaseous bidirectional transformations of the auxiliary materials, which allow the lubrication of the processed material in the primary cooling zone;
- liquid-gaseous phase transformations for the fluid materials that take the thermal energy of the solidified material and make the secondary cooling.

Taking in account such a complex set of phenomena, a simplified model will be considered based on balance energy. This approach is started also from the reality that in the real installation the information for control and monitoring purposes use global variables, e.g. volume flow rates and temperatures of the involved materials, and not local variables, like densities and viscosities.

In Fig. 4 the structure of the cooling models is presented. There are also represented the sources of the materials and materials loads. The structure of the cooling model is represented in the upper left side of the Fig. 4. Three models are considered: two of materials (Steel and Water) and a model for separation (Wall).

On the upper right side the structure of the processed material is presented, as interaction of two submodels: material in liquid phase (Liquid Steel) and material in solid phase (Solid Steel). For these two phases interfaces were defined: *tfla*, *tflb* for liquid phase and *tfsa*, *tfsb* for solid phase.

The processed materials have material interfaces (*tfa*, *tfb*, *tfc*) and interfaces for changing of the thermal energy (*ha*), as it is presented in the lower right side.

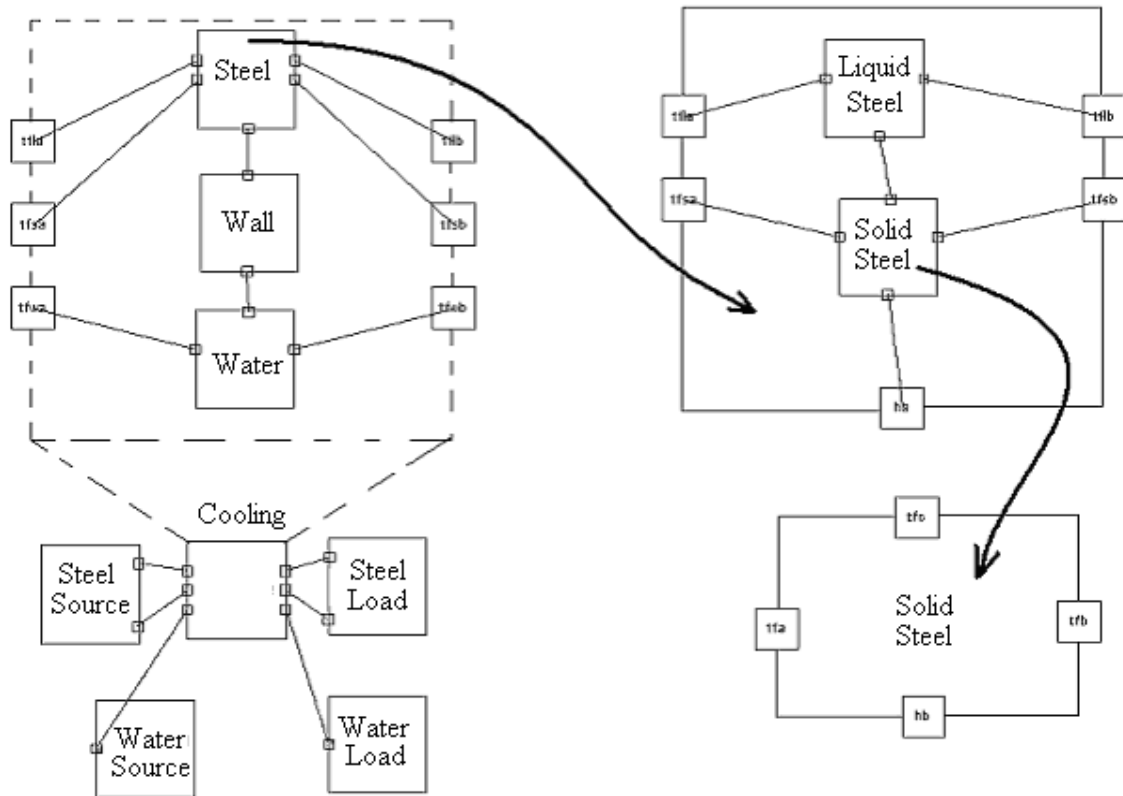


Figure 4: The structure of the cooling model for the continuous casting of the steel using water as cooling agent

The cooling model is designed with the same structure in order to be able to use in both sides of the cooling zones: primary and secondary. Setting up the numerical values of input-output variables, makes the selection of one of them. By example, for the primary cooling zone zeros values are necessary for the solid materials, because the material in received from tundish with liquid phase only. In the following the model of the processed material will be described, as interaction between solid and liquid phase. The model for the solid phase is

```

model SteelSolid "The SteelSolid model"
  PortThermoFluid tfa,tfb, tfc;
  PortHeat hb;
  extends SteelSolidProperties;
  CelsiusTemperature T (start=30);
  Volume vol(start=1, min=0.1);
  Enthalpy hin1(start=1)"In-flow from source side";
  Enthalpy hin2(start=1)"In-flow from liquid side";
  Enthalpy hout(start=1)"Out-flow";
equation
  hin1 = tfa.qvol * shcap * rho * tfa.T;
  hin2 = tfc.qvol * shcap * rho * tfc.T;
  hout = tfb.qvol * shcap * rho * tfb.T;
  vol*rho*shcap*der(T)=hout+hin1+hin2+hb.qheat;
  tfb.T = T;

```

```

hb.T = (tfa.T + tfb.T + tfc.T)/3;
// undefined: vol;
end SteelSolid;

```

The model for the liquid phase is

```

model SteelLiquid "The SteelLiquid model "
  PortThermoFluid tfa,tfb, tfc;
  Extends SteelLiquidProperties;
  Temperature T (start=1500);
  Volume vol(start=2, min=0.1);
  Enthalpy hin(start=1)"In-flow from source side";
  Enthalpy hout1(start=1)"Out-flow to next block";
  Enthalpy hout2(start=1)"Out-flow to solid steel";
  Enthalpy H(start=1) "Latent energy";
equation
  hin = tfa.qvol * shcap * rho * tfa.T;
  hout1 = tfb.qvol * shcap * rho * tfb.T;
  hout2 = tfc.qvol * shcap * rho * tfc.T;
  H = vol * rho * L;
  der(vol)=if vol>0 then tfa.qvol+tfb.qvol+tfc.qvol
  else 0;
  vol*rho*shcap*der(T)=hin+hout1+hout2- der(H);
  tfb.T = T;
end SteelLiquid;

```

The model for the processed (cooled) material is

```

model Steel "The Steel model"
  PortThermoFluid tfla, tflb;
  PortThermoFluid tfsa, tfbs;
  PortHeat ha;
  SteelLiquid sliquid;
  SteelSolid ssolid;
  Volume vol(start=3, min=0);
  parameter Real K (start=1e-8) "L2Solid speed";
algorithm
  ssolid.tfc.qvol := K * sliquid.H;
equation
  connect(tfla, sliquid.tfa);
  connect(sliquid.tfb, tflb);
  connect(ssolid.tfa, tfsa);
  connect(ssolid.tfb, tfbs);
  connect(sliquid.tfc, ssolid.tfc);
  connect(ssolid.hb, ha);
  sliquid.vol + ssolid.vol = vol;
  ssolid.tfc.T = if sliquid.vol > 0 then sliquid.Ts else
ssolid.T;
// un-defined: vol; context dependent;
end Steel;

```

Finally, the cooling model considers the interaction between the model of the processed material (steel) and the material that takes the energy in order to be able to transform the processed material from liquid to solid phase. The cooling model is as

```

model Cooling
  PortThermoFluid tfla, tflb "Steel Liquid";
  PortThermoFluid tfsa, tfbs "Steel Solid";
  PortThermoFluid tfwa, tfwb "Water";
// from geometry model;:
  parameter Real Rfluid (start=1) "Thermal res";
  parameter Area area (start=1) "Heat Transfer area";
  Water water; Steel steel; Wall wall;
equation
  connect(steel.tfla, tfla);
  connect(steel.tflb, tflb);
  connect(steel.tfsa, tfsa);
  connect(steel.tfsb, tfbs);
  connect(steel.ha, wall.ha);
  connect(wall.hb, water.ha);
  connect(water.tfa, tfwa);
  connect(water.tfb, tfwb);
  steel.vol = 3; // real volume must be defined;
  water.vol = 0.5; // real volume must be defined;
end Cooling;

```

7. The casting model

The casting model is composed from three main submodels or modules: the ladle, the tundish, and the

cooling model. In the simulation scenario two other models are necessary, i.e. the source of the steel, which impose the events in changing the liquid material on the input of the process, and, the second, the load models which is responsible for the reference of the casting speed. The declarative model can be as

```

model TestCastingProcess
  SourceSteel steel_S;
  Ladle ladle; Tundish tundish;
  Cooling cool;
  WaterSource water_S;
  WaterLoad water_L;
  LoadSteel load_S;
equation
  connect(steel_S.tfla, ladle.tfla);
  connect(ladle.tflb, tundish.tfla);
  connect(tundish.tflb, cool.tfla);
  connect(water_S.tfa, cool.tfwa);
  connect(cool.tfwb, water_L.tfa);
  connect(cool.tflb, load_S.tfla);
  connect(cool.tfsb, load_S.tfsa);
  connect(cool.tfwb, water_L.tfa);
end TestCastingProcess;

```

Fig.5 presents the evolution of the steel temperatures in ladle and tundish. The simulation scenario supposes that from time to time, when the temperature is increasing in steps, the ladle is filled up with new liquid material.

Conclusions

The main goal of the work was to obtain a neutral representation of the continuous casting process of the steel. Considering all aspects of the process is out of the scope and is quite difficult without a base library of materials and phase transformation under neutral format.

It was supposed that the neutral model is the start point in the development of more sophisticated and more accurate models and it is used as a first description of the process. Some aspects of the modelling methodology using Modelica language were also presented.

The models for ladle, tundish and cooling zone were presented. Simulation results are presented also in order to check the right qualitative behaviour of the obtained models, under an imposed scenario at the input of the casting process. The results are accurate regarding on the evolution of the temperature of the steel inside the tundish.

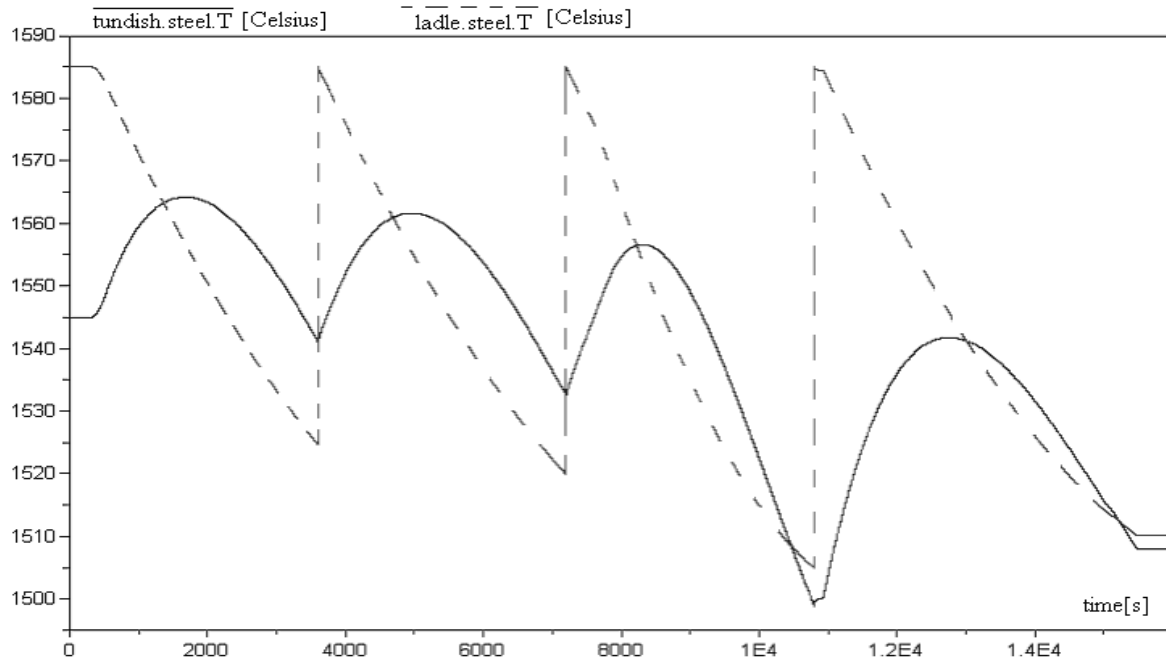


Figure 5: Temperature evolution of the steel in ladle and tundish

References

- [1].Modelica and Modelica Association, <http://www.modelica.org/>, 2004.
- [2].Michael Tiller, Introduction to Physical Modelling with Modelica, Kluwer Academic Publisher, 2001.
- [3].Thomas, B.G. "Continuous Casting: Modelling," The Encyclopedia of Advanced Materials, (J. Dantzig, A. Greenwell, J. Michalczyk, eds.) Pergamon Elsevier Science Ltd., Oxford, UK, Vol. 2, 2001, 8p., (Revision 3, Oct. 12, 1999).
- [4].Thomas, B. G., B. Ho, et al., Heat Flow Model of the Continuous Slab Casting Mold, Interface, and Shell. Alex McLean Symposium Proceedings, Toronto, Warrendale, PA, Iron and Steel Society: 177-193, 1998.
- [5].D.T.Creech, Computational modelling of multiphase turbulent fluid flow and heat transfer in the continuous slab casting mold, Master of Science in Mechanical engineering Thesis, University of Illinois at Urbana-Champaign, USA, 1997.
- [6].McDavid, R. and B. G. Thomas: "Flow and Thermal Behavior of the Top-Surface Flux/Powder Layers in Continuous Casting Molds." Metallurgical Trans. B **27B** (4): 672-685, 1996.
- [7].Thomas, B. G., X. Huang, et al., Simulation of Argon Gas Flow effects in a Continuous Slab Caster, Metallurgical Trans. **25B**(4): 527-547, 1994.
- [8].Thomas, B. G., A. Dennisov, et al., Behavior of Argon Bubbles during Continuous Casting of Steel. Steelmaking Conf. Proceedings, Chicago, IL, ISS, Warrendale, PA. : 375-384, 1997.
- [9]. Thomas, B.G and S.P. Vanka, Study of Transient Flow Structures in the Continuous Casting of Steel, University of Illinois at Urbana-Champaign, Mechanical and Industrial Engineering, Research Report,1999.
- [10].Thomas, B.G., Continuous Casting of Steel, Chap. 15. Modeling and Simulation for Casting and Solidification: Theory and Applications. Marcel Dekker, New York, 2000.
- [11].B. Kiflie and D. Alemu, Thermal Analysis of Continuous Casting Process, Faculty of Technology, Addis Ababa University, Ethiopia, ESME 5th Annual Conference on Manufacturing and Process Industry, September 2000.
- [12].B.Koza and J. Dzierzawski, Continuous Casting of Steel: Basic Principles, Research Article, American Iron and Steel Institute, Learning Center, <http://www.steel.org>, 2000.

Switched Capacitor Simulation with Modelica

Christoph Clauß¹⁾, Elisabeth Erler²⁾

1) Fraunhofer Institute Integrated Circuits, Branch Lab Design Automation
Zeunerstraße 38, D-01069 Dresden, Germany

2) Berufliches Schulzentrum, Otto-Dix-Straße 2, D-01705 Freital
clauss@eas.iis.fhg.de

Abstract

To simulate switched-capacitor circuits effectively special simulators use the charge-voltage system of equations instead of the current-voltage system. Furthermore, the set of devices is limited. In this paper possibilities are presented to follow this approach in Modelica. An example switched-capacitor library is implemented as well as example circuits.

1 Introduction

Switched-capacitor (SC) networks are often used for the realization of filters, comparators, or integrators. As a simple example of a switched-capacitor circuit c.f. **figure 1**. Depending on the switching frequency resistors with varying resistance can be created. Integrated SC circuits are often much cheaper than conventional IC's [1]. Since in such circuits the switching frequency and the signal frequency often differ considerably, long time simulations are necessary to investigate the circuit behaviour. This is time consuming because small switching intervals with high currents flowing cause small step sizes during the simulation.

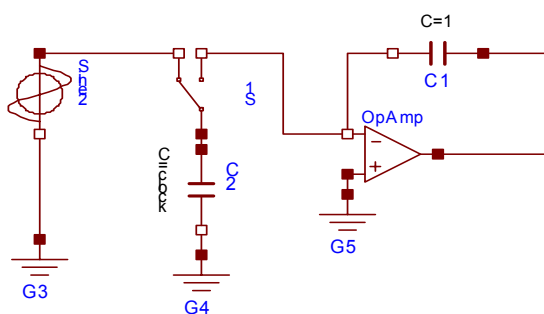


Figure 1: SC Integrator circuit

A simplification is possible if only the voltages at the ends of switching intervals are of interest. In such cases the calculation of the behaviour of currents can be avoided. Provided that the exchange of charges is finished within the switching interval it is sufficient to calculate the voltages for the total charge equalization only. If furthermore only devices are in the circuit

which do not need currents explicitly no differentiation of the charge is necessary. In this case only algebraic (linear or nonlinear) equations have to be solved with the stepsize of the switching intervals. Typical devices possible at this switched-capacitor simulation approach are ideal voltage sources, ideal switches, capacitors, voltage-controlled voltage sources, operational amplifiers. Some special simulators exploit this approach, e.g. TOSCA [2], SWITCAP [3], AWESwit [4].

In this paper a possibility is studied to perform the SC-simulation with Modelica.

2 Fundamentals

For the description and simulation of electrical circuits usually relations between currents (i) and voltages (v) are used. The combination of the equations of all devices in a circuit together with the KIRCHHOFF's law equations at nodes results in a current-voltage-system which consists of differential-algebraic equations (DAE).

This DAE has to be used if the switching behaviour is of interest. If from a more general point of view an abstraction from the switching behaviour is acceptable the switches can be modelled more ideally. In this case high current impulses occur which force the simulation of the DAE to small time steps, and therefore to a poor performance.

A way out is the change from the current-voltage-system to the charge-voltage-system. Provided that:

- the ideal switches are timed in an equidistant scheme (stepsize s)
- the devices are restricted to capacitors and such devices which can be described by algebraic relations between pin voltages only (e.g. ideal voltage sources, voltage controlled voltage sources, ideal operational amplifiers, ideal switches)
- capacitors are the only devices which combine pin currents and pin voltages
- no currents themselves are of interest

Then the current-voltage DAE can be integrated over each step interval of the size s . Since derivations with respect to time occur in capacitors only capacitors are outlined in detail:

The equation of a capacitor is:

$$i = \frac{d}{dt}C(v) \quad (2-1)$$

with i being the current and v the voltage over the capacitor. If $[ta, tn]$ with $s = tn - ta$ is the actual time interval the integration of (2-1) with respect to time results into

$$q = \frac{d}{dv}C(v)(v(tn) - v(ta)) \quad (2-2)$$

where q is the charge transported into the capacitor within the interval. If the capacitance is constant the formula is

$$q = C \cdot (v(tn) - v(ta)) \quad (2-3)$$

The device equations (2-2), (2-3) contain voltages and charges only. Since according to our assumption all other devices can be described by algebraic relations between pin voltages, and KIRCHHOFF's current equations can be trivially integrated using

$$q = \int_{ta}^{tn} i(t) dt \quad (2-4)$$

the resulting system of equations is a linear or nonlinear algebraic **charge-voltage** system.

During simulation it has to be solved once at each time interval at tn . Therefore a higher performance can be expected than solving the current-voltage-DAE. Further optimizations are possible if the network topology is exploited, which often changes between two cases only.

If the restricted amount of devices is left since e.g. a resistor is needed, then the current needs to be calculated by differentiating the charge variable q (2-4). In this case a differential-algebraic system is constructed with the loss of the above mentioned advantages.

There are some generalizations possible such as non-equidistant time grids, switching depending on voltage values and others, which are not yet considered in this paper. In the following the implementation of a switched-capacitor library is described which bases on the charge-voltage-system of circuit equations.

3 Implementation

In contrast to usual electrical modelling the connectors include the voltage, and the charge, which is transported via the pin in one switching interval. The charge is a flow value like the current in the current-voltage system since according to (2-4) the charge meets KIRCHHOFF's law. The connector definition is:

```
connector VoltageChargePin
  Modelica.SIunits.Voltage v
    "Potential at the pin";
  flow Modelica.SIunits.Charge q
    "Charge flowing into the pin";
end VoltageChargePin;
```

Using this connector models and partial models can be created like in the Modelica.Electrical.Analog package.

Basically, there are two groups of devices: Devices which depend on the switching interval, like switches and capacitances, and other devices which do not depend on the switching interval. Devices depending on the switching interval must be „informed“ about the events of switching. This could be achieved by further connector, which is connected with a clock generator, or a logic network. Since in this implementation the restriction is that each switch changes at equidistant timesteps, each device depending on switching intervals has a clock parameter with $clock = 2 \cdot s$. Via the sample function

```
algorithm
  when sample(0, clock/2.0) then ...
```

the calculations are controlled which have to be done at switching time points. The advantage of this approach is that no clock connections are necessary. Otherwise the user has to care about the correct clock parameters at each device. This is a disadvantage. In the examples a central clock parameter is introduced. Each device clock parameter is set equal to the central clock parameter by the user. The choice of $clock = 2 \cdot s$ instead of $clock = s$ seems to be practical: the clock parameter covers a complete on-off-interval.

Using this clock handling and equation (2-3), the implementation of the linear constant capacitor device is:

```
model Capacitor
  extends Interfaces.OnePort;
  parameter Modelica.SIunits.Capacitance C=1;
  parameter Real clock=1;
  Real vlast(start=0);
  Real tlast(start=-1);
algorithm
  when sample(0, clock/2) then
    if (time > tlast) then
      tlast := time;
```



```

    vlast := pre(v);
  end if;
end when;
equation
  q = C*(v - vlast);
end Capacitor;

```

In the algorithm section only once at a sample $tlast$ and $vlast$ (which correspond to ta and $v(ta)$ in equation (2-3)) are calculated. This is ensured by comparing the time with the variable $tlast$. The equation (2-3) itself is located in the equation section because it can be solved manifold at a switching point during iterations.

The switches are modelled similarly. At switching samples the state of the switch is changed. The pin relations are formulated in the equation section:

```

model OpeningSwitch
extends SwitchedCapacitor.Interfaces.OnePort;
parameter Real clock=1;
Real s;
Boolean control(start=false);
Real tswitch(start=-1);
algorithm
  when sample(0, clock/2.0) then
    if (time > tswitch) then
      tswitch := time;
      control := not control;
    end if;
  end when;
equation
  v = s*(if control then 1 else 0);
  q = s*(if control then 0 else 1);
end OpeningSwitch;

```

If for voltage inputs the electrical models shall be used a converter between the switched-capacitor and the usual electrical domain is necessary. Since signals in the switched-capacitor domain change at switching time points only it is useful to sample the input voltage. The converter model without any feedback into the current-voltage domain is:

```

model ElectricalToSwitchedCapacitorVoltage
parameter Real clock=1;
Interfaces.VoltageChargePin pinSC;
Modelica.Electrical.Analog.Interfaces.Pin pinElectrical;
algorithm
  when sample(0, clock/2.0) then
    pinSC.v := pinElectrical.v;
  end when;
equation
  pinElectrical.i = 0;
end ElectricalToSwitchedCapacitorVoltage;

```

Devices which do not depend on the switching intervals are modelled like the counterparts in the current-voltage-domain. Merely currents (i) are replaced by charges (q). As examples the voltage controlled voltage model and the ideal opamp model are cited:

```

model VCV
extends
  SwitchedCapacitor.Interfaces.TwoPort;
parameter Real gain=1;
equation
  v2 = v1*gain;
  q1 = 0;
end VCV;

model IdealOpAmp
SwitchedCapacitor.Interfaces.VoltageChargePin
  in_p, in_n, out;
equation
  in_p.v = in_n.v;
  in_p.q = 0;
  in_n.q = 0;
end IdealOpAmp;

```

The device models are combined to a SC-library for test and investigation purposes. It contains simple models only. An extension towards more complicated devices like operational amplifiers with parasitic capacitances and offset or nonlinear capacitance models is possible.

4 Examples

The models of the SC-library are successfully tested at a collection of about 20 circuits:

- simple resistors replaced by switched capacitors
- charging of one or more capacitances
- SC-integrators
- SC delay circuit
- Cauer-filter

In this section some of the examples are presented to demonstrate that the SC-simulation works correctly. All examples were simulated using the simulator Dymola5.3a.

4.1 Constant charge flow

In this example a constant charge flow circuit (Fig. 2) is simulated, which is compared with a constant current flow through a resistor.

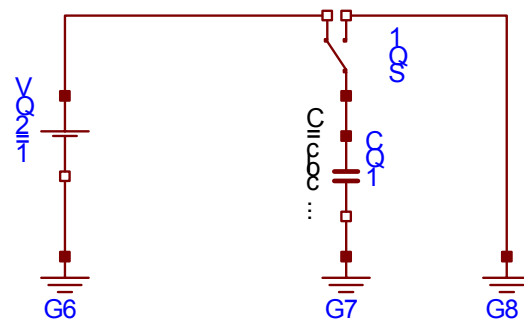


Figure 2: Constant charge flow circuit

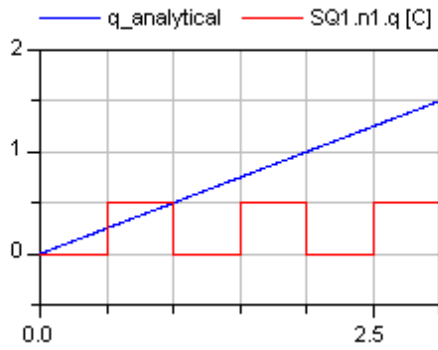


Figure 3: Constant charge flow result

In Fig. 3 the linearly growing line shows the analytical calculated charge of a circuit like in Fig. 2 where the switched capacitor with the value $clock/2$ is replaced by an equivalent resistor of value $R=2$ in the current-voltage domain. After each clock period ($=1$ second) which includes two switching periods ($s=0.5$ second) the charge flowing into the switch is equal to the charge of 0.5 Coulomb flown in the actual clock interval.

In the following Fig. 4 the current of the SC-Circuit like Fig. 2 is simulated in the current-voltage domain. The current peaks depend on the parasitic resistance value in the current-voltage switch model. A purely ideal simulation is not possible in the current-voltage domain.

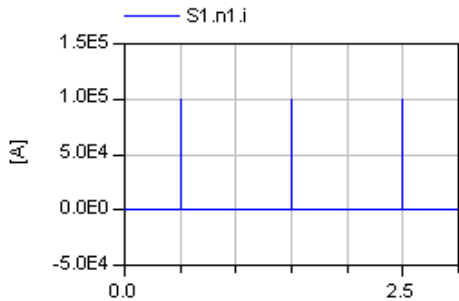


Figure 4: Current peaks in current-voltage simulation

4.2 Charging a capacitance

The following circuit (Fig. 5) is a simple charging up of a capacitance. In this example an electrical voltage source is used. The voltage is converted into the SC domain with the charge-voltage system.

The pictures in Fig. 6 show the voltages of the capacitances C , and C_s . Depending on the state of the switch the voltage $Cs.p.v$ of C_s is 1V, if C_s is connected with the voltage source, or it is equal to the voltage $C.p.v$ of C which is increasing with each charge equalization.

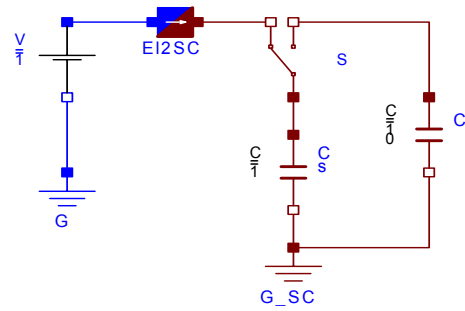


Figure 5: Charging a capacitance

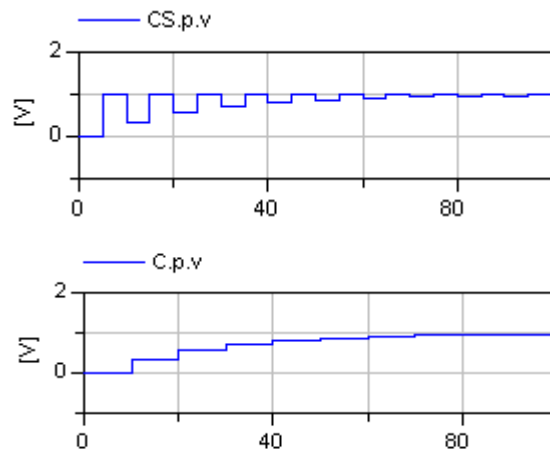


Figure 6: Charging a capacitance: voltages

Furthermore, Fig. 7 shows the charges flowing through the positive pins of both capacitances.

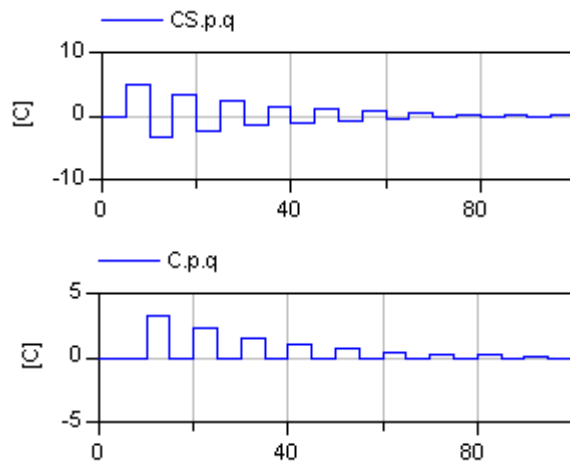


Figure 7: Charging a capacitance: charges

The charge flow through $Cs.p$ is positive if C_s is charged by the voltage source. Otherwise it is negative in the case of the charge equalization between both capacitances. The velocity of charging depends on the switching interval length $clock$.

4.3 SC-Integrators

The SC-Integrator [5] according to **Fig. 1** integrates the input voltage. The result which is inverted, can be seen in **Fig. 8**.

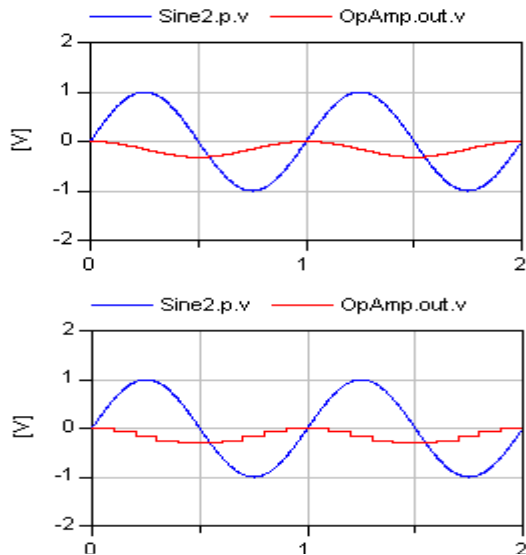


Figure 8: Inverting integration results with clock=0.01 (above) and clock=0.1 (below)

With a slightly changed topology [5] according to **Fig. 9** a noninverting integration of the input signal is possible. The results depend on the clock length. Already large clock switching intervals calculate sufficient results (**Fig. 10**).

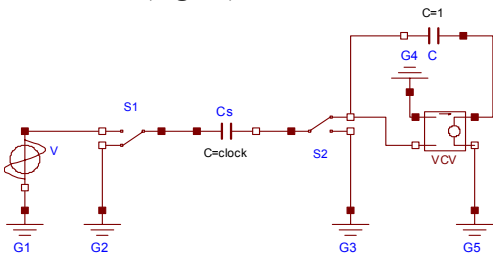


Figure 9: Noninverting SC-Integrator

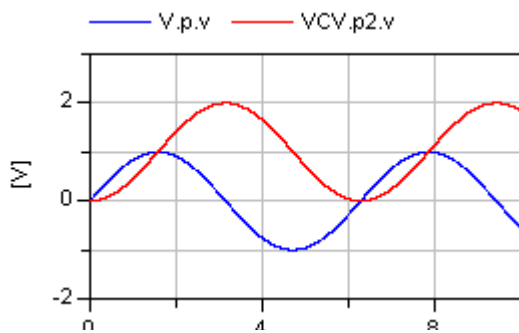


Figure 10: Noninverting integration results

4.4 Delay circuit

A clock-controlled delay example is the circuit in **Fig. 11** which combines two voltage amplifiers [6]. The result is delayed by one clock length. **Fig. 12** shows the input signal and the delayed output at a clock of 1.e-4.

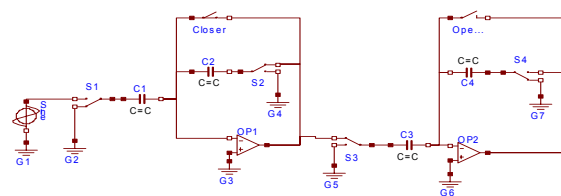


Figure 11: Clock delay circuit

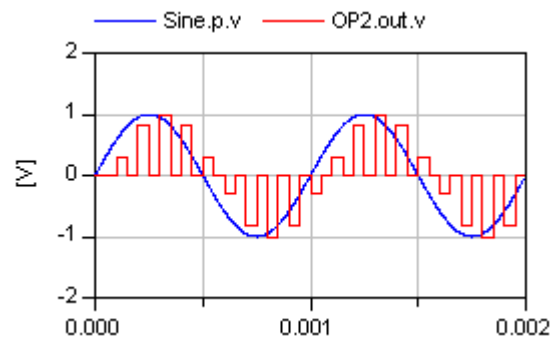


Figure 12: Sine input and delayed SC result

4.5 Cauer filter

As a final and more complex example which demonstrates the possibilities of the SC package, the 5th order cauer filter according to **Fig. 14** is modelled [7]. For purposes of test an unusual time scale is used. The pulse response results of some of the opamp outputs can be seen in **Fig. 13**.

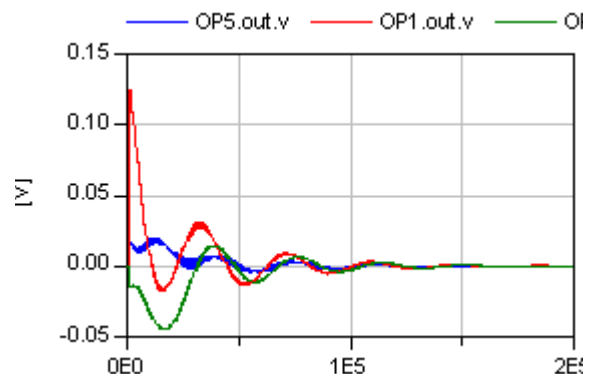


Figure 13: Sine input and delayed SC result

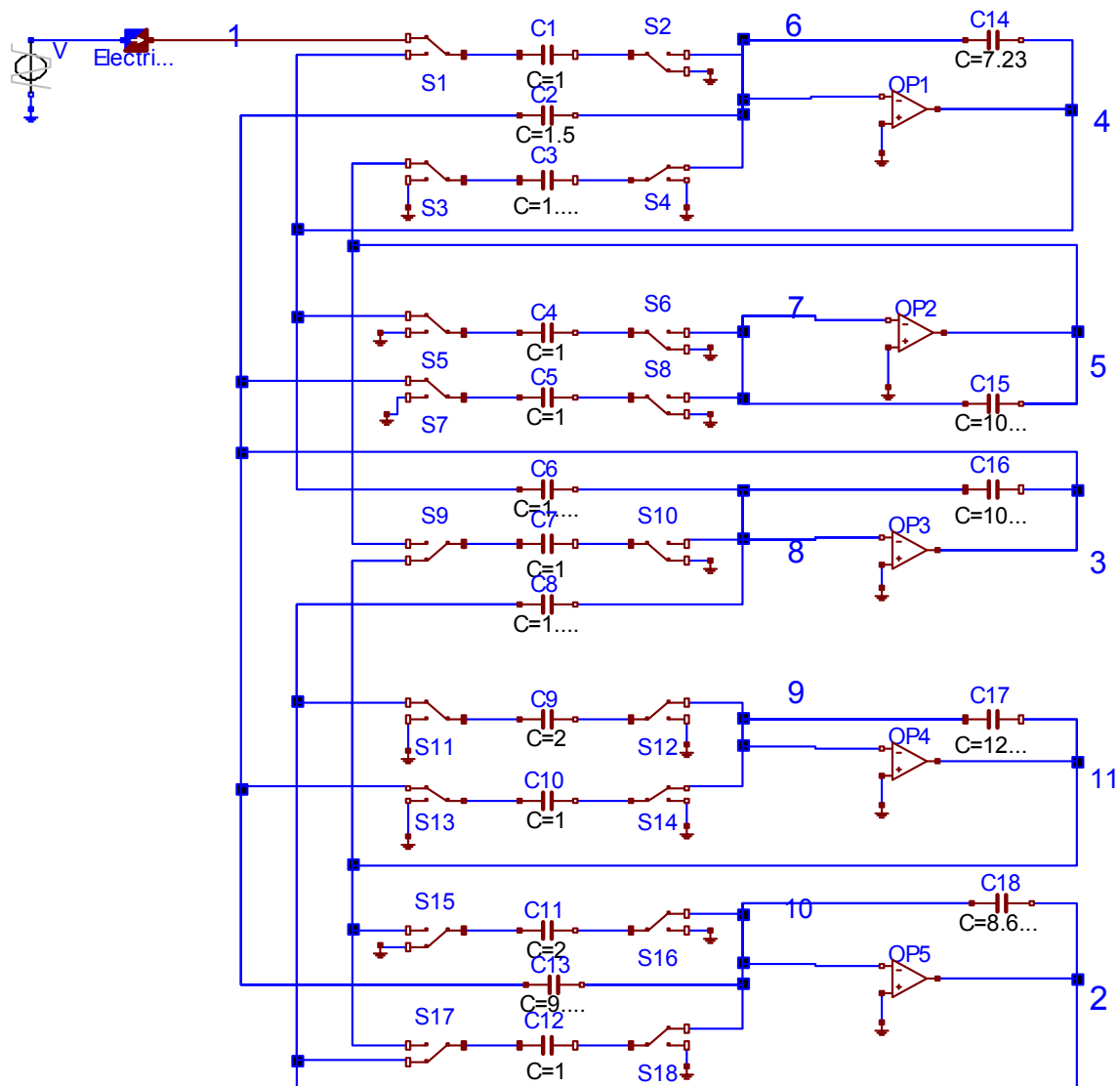


Figure 14: Cauer filter

4.6 Statistics

To compare the performance between simulations of the current-voltage system and the charge-voltage system of SC circuits the following test examples are used:

circuit	
Integ	Integrator according to Fig. 1
Integnon	noninverting integrator according to Fig. 9
Delay	Delay circuit according to Fig. 11
CauerOP	Cauer filter according to Fig. 14
CauerVC	like CauerOP, using voltage-controlled voltage sources instead of operational amplifiers

Each circuit is modelled both as current-voltage system using the Modelica.Electrical.Analog package (with default parameters of the switches) and as charge-voltage system using switched capacitor package. In the following the systems are abbreviated with CU (current-voltage system) and CH (charge-voltage system).

In the following table system related quantities calculated by Dymola are collected:

- unkn: unknown variables before translation
- diff: differentiated variables before translation
- tvar: time varying variables after translation
- state: continuous time states after translation

circuit	CU unkn	CU diff	CU tvar	CU state	CH unkn	CH diff	CH tvar	CH state
Integ	40	2	10	2	45	0	14	0
Integnon	61	2	18	2	65	0	24	0
Delay	111	4	35	4	123	0	53	0
CauerOP	382	18	135	14	410	0	209	0
CauerVC	412	18	177	14	440	0	219	0

Table 1: Translation related quantities

In the next table simulation related quantities are shown:

- steps: number of successful steps
- F: number of F-evaluations
- Jac: number of Jacobian-evaluations

circuit	CU steps	CU F	CU Jac	CH steps	CH F	CH Jac
Integ	22744	61244	10800	4700	8900	4200
Integnon	111387	295721	51654	20000	38000	18000
Delay	2613	9205	1173	640	1200	560
CauerOP	19912	241201	15222	5030	9657	4627
CauerVC	23273	331305	20891	5030	9657	4627

Table 2: Simulation related quantities

In the following table the CPU-time is compared:

- tstop: stop time
- CPU: CPU-time for integration in seconds

circuit	tstop	clock	CU CPU	CH CPU
Integ	2	0.01	1.4	0.7
Integinv	10	0.01	5.8	2.0
Delay	0.002	0.0001	1.1	0.95
CauerOP	200000	1000	18.7	1.68
CauerVC	200000	1000	49.2	1.71

Table 3: CPU-time

The simulations run on a 800 MHz PC with 128 MB RAM.

Although in the charge-voltage system the number of time-varying variables after translation is higher than in the current-voltage system (Table 1) the computational amount in the charge-voltage system is far less

than in the current-voltage system (Table 2, Table 3). The reason is that the charge-voltage system of equations is an algebraic one (Table 1).

If otherwise the at least necessary number of steps is calculated according to $tstop/(clock/2)$ in Table 3 the resulting number is less than the number of steps according to Table 2. That means that there are further possibilities of optimization within the simulation algorithm.

5 Conclusion

The main result of this investigation is that switched-capacitor simulation with Modelica and Dymola is possible. The switched-capacitor simulation using the charge-voltage system and the restricted set of devices is clearly faster than the simulation of the current-voltage system. The Cauer example shows that the package can be applied for the simulation of more complex examples than simple test cases. An extension of the package to devices including parasitic effects, nonlinearities etc. is desirable.

Tasks for future research are more flexible controlling of switches e.g. via logic networks and further optimization of the algorithm, especially in comparison with switched-capacitor special simulators.

6 References

- [1] Horowitz, P.; Hill, W.: The Art of Electronics. Cambridge University Press, 1989
- [2] Liberali, V. et al.: TOSCA: A simulator for switched-capacitor noise-shaping A/D converters. IEEE Tran. on Comp.-aided Design of Integr. Circuits and Syst. 12(1993)9 1376-1386
- [3] SWITCAP - Simulator for ideal switched capacitor networks. <http://www.cisl.columbia.edu/projects/switcap>
- [4] Trihy, R.; Rohrer, R. A.: A switched capacitor circuit simulator: AWESwit. IEEE Journal of Solid State Circuits 29(1994)3, 217-225
- [5] Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungselektronik. Berlin, Heidelberg, New York, Springer-Verlag, 1980
- [6] Civardi, L.; Gatti, U.; Torelli, G.: An AHDL-based methodology for computer simulation of switched-capacitor filters. Elsevier Science Microelectronics Journal, 27(1996)6, 485-497
- [7] Fehlauer, E.; Krauß, M.: Ein effektiver Algorithmus zur Zeitbereichsanalyse von SCOV-Schaltungen. Wiss. Z. Techn. Univers. Dresden 35(1986)H.4, 159-163

Hydrological modeling in Modelica

Karin Berg Kaj Nyström
 Linköping University
 Linköping, Sweden
 kb@lysator.liu.se kajny@ida.liu.se

Abstract

Hydrological modeling is an area where modeling has been used for a very long time. Applications range from forecasts for the hydro power industry, public safety, agriculture and environmental monitoring. Still, to the best of our knowledge, Modelica has been very little used in hydrological modeling so far. In this paper, we aim to show that the Modelica language is well suited for hydrological modeling and also to outline a possible future development of libraries in order to further facilitate hydrological modeling and coupling of hydrological models to other types of models in Modelica.

A Modelica implementation of the hydrological HBV model is compared with the original Fortran model. The main advantages of using Modelica as modeling language are more readable and re-usable code and better abstraction. The disadvantage is longer execution times compared to the Fortran model.

The HBV model is a quite simple model mathematically. It would be useful to investigate the behaviour of more complex hydrological models as well in order to see whether we can find the same advantages of using Modelica as modeling language in that respect as we have in the case with the HBV model.

Keywords: hydrology, modeling, HBV, Modelica, runoff simulation

1 Introduction

Even though specialised modeling languages have matured over the years, Modelica perhaps being one of the best examples, most hydrological models are still written in Fortran. This, we believe, hampers the development of hydrological modeling. Especially since considerable knowledge of computer science in general and Fortran programming in particular is not something that every hydrologist possess. Clearly, hydrological modeling needs better tools in order to facil-

itate future model development. The goal of this paper is to present Modelica as an alternative modeling language for hydrological applications and to investigate if a direct translation to Modelica of a hydrological model is actually easier to understand than the corresponding Fortran model.

Hydrological models used today have a wide range of applications including decision support for different business purposes, for example energy trading and farming, hydrological forecasts and warnings and other public safety applications. In addition, they are also often coupled with for example climatological, meteorological, chemical and/or biological models.

As an example of a hydrological model we have chosen a version of the HBV model [4]. HBV is a widely used model throughout the world primarily for hydrological forecasting and runoff simulation, for example as a tool when designing dams for the hydro power industry, but the HBV model has been applied to many other areas as well. Among recent applications are modeling and simulation of nutrient transport in large catchments and simulations of the potential effects of climate change on water flows, water quality, particle transport and biochemical processes in the water.

2 Model description

The HBV model was developed at the Swedish Meteorological and Hydrological Institute (SMHI) in the 1970's [4]. HBV/PULSE, which is used in this study, is a similar model developed from the HBV model in the 1980's as a consequence of the need to study acidification and substance transport. The two models have very similar structure but the HBV/PULSE model is slightly less complex when only hydrology is simulated. HBV and similar models are often described as semi-distributed conceptual models. They have some spatial resolution since they handle systems of catchments, lakes and rivers, but within catchments the spatial resolution is limited.

It is conceptual in the sense that it does not use detailed physical laws of nature in the calculations but rather simple equations which are consistent with the current hydrological knowledge. The model consists mainly of three parts: snow, soil moisture and runoff response functions. These steps are calculated for each land use type in every catchment and there is also possibility to divide the catchments further into sub-catchments. In the HBV model, but not in the HBV/PULSE model, it is also possible to divide the catchments and sub-catchments into altitude zones.

There are also some routines for weighting and correction of input data and evaluation of model performance. Output data from the forest, field and lake areas is weighted. The present model is based on an HBV/PULSE model [2] used for research purposes.

Driving variables in the model are daily temperature and precipitation measurements and monthly averages of potential evapotranspiration. The number of parameters in the model vary slightly depending on the current version and application area. The model used in this project has 34 parameters. Both driving variables and parameters are stored in text files.

3 Model implementation in Modelica

Existing libraries such as Fluid, Species, WasteWater [5], HylibLight (light version of HyLib [1]), QSSFluidFlow [3] were considered for the implementation of HBV, but we found them not very well suited for this application. Since the existing hydraulics libraries are developed for different engineering purposes they include for example pressure, conductance, geometry of pipes and vessels and other parameters which are not known and/or not meaningful to use on the scale on which the HBV model is operating.

Instead, we have created some very simple general components for water storage and transport where the only variable is the water volume itself. There are both some general base classes and HBV specific components and file reading functions.

The HBV model components can be divided into three levels as shown in figure 1. It was possible to transfer some parts of the model to equation form but the main part is written as algorithm statements due to special cases such as if-statements without any else-clause or with different number of assignments in different parts of the statement.

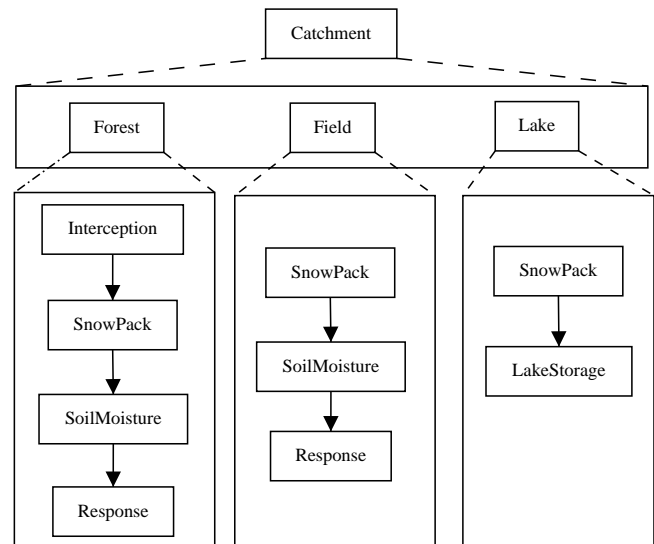


Figure 1: Component hierarchy in the Modelica implementation of the HBV model: Catchments consists of several land use components which in turn consists of different basic HBV components.

3.1 General components and interfaces

In hydrological applications, water volume is sometimes expressed in the unit mm. Water in the HBV model is entirely expressed in mm and conversion to other units is only performed when all calculations have been made. This may seem strange but is a consequence of precipitation measurements in dm^3/m^2 , which can be reduced to simply mm. Because of this, a HydrologyVol unit, connectors and storages for volumes expressed in mm have been created (figure 2). A snow storage was also created, which stores water in both frozen and liquid form. Connectors, icons and variables for stored volumes are declared in the general components.

Discrete variables have been used since the first aim was to make a quick translation of the HBV model from Fortran to Modelica and this was most easily accomplished using discrete variables rather than continuous.

Finally, there are base classes for sources and an infinite sink model. The sink model is not storing anything at the moment, but it can easily be modified to do so. Sources are available with outlets of one or two types. In this application they are used for sources with both snow and water outlets, providing input for the snow storages described above.


```

package Units
  type HydrologyVol =
    Real(final quantity="HydrologyVol",
         final unit="mm");
end Units;
package Interface
  connector HydrologyFlow
    import Hydrology;
    Hydrology.Units.HydrologyVol q;
  end HydrologyFlow;
  connector QoutD =
    discrete output HydrologyFlow;
  connector QinD =
    discrete input HydrologyFlow;
end Interface;
package Components
  partial model Storage
    import Hydrology.Units.HydrologyVol;
    import Hydrology.Interface.QinD;
    import Hydrology.Interface.QoutD;
    discrete HydrologyVol w(min=0);
    QinD qIn1;
    QoutD qOut1;
  end Storage;
  partial model SnowStorage
    import Hydrology.Units.HydrologyVol;
    import Hydrology.Interface.QinD;
    import Hydrology.Interface.QoutD;
    Interface.QoutD qOut1;
    Interface.QinD qIn1;
    Interface.QinD qIn2;
    discrete HydrologyVol s(min=0);
    discrete HydrologyVol w(min=0);
  end SnowStorage;
end Components;

```

Figure 2: Unit declaration and examples of connectors and storages using mm as volume unit

3.2 HBV basic components

Most of the HBV model equations are found in the basic HBV components Interception, SnowPack, SoilMoisture, Response and LakeStorage. One difference versus the Fortran model is that Interception and SnowPack are separated instead of treated together in a common snow routine. This makes the model structure more clear and it also made it possible to use fewer parameters. All basic components except Response are implemented using algorithm statements. The Response model is much smaller than the others and there were no problems with expressing it entirely in equation form (figure 3).

There are limitations on storage of water and snow intercepted in for example trees, since the trees only can

hold a limited amount of water or snow. There is also a limitation on relative water content in the snow, but SoilMoisture and Response have no such limits.

```

model Response
  import Hydrology;
  extends Hydrology.Components.Storage;
  outer parameter Real k;
  outer parameter Real alfa;
equation
  when sample(0,1) then
    w = pre(w) + qIn1.q - qOut1.q;
    qOut1.q = 0.001*k*w^(1 + alfa);
  end when;
end Response;

```

Figure 3: Modelica code for the HBV response function

3.3 HBV land use components

Features common for the three types of land use class in HBV - the icon type and a snow pack variable which need to be accessed from several basic components - are declared in the partial class LandUse. Forest, Field and Lake are extensions of LandUse. Forest is the land use model which contains most parts and equations. It consists of four model parts: Interception, SnowPack, SoilMoisture and Response, and also one sink for containing evaporated water. Field has the same structure as Forest but without Interception since field is defined in the HBV model as land area with no interception. Lake models consists of only two parts - SnowPack and LakeStorage. Contributions from the different land use types are added and given a weight proportional to their area. Calculations for the land use components are only performed if the area is greater than zero.

3.4 HBV catchment component

The largest component in the HBV model is the catchment. Each Catchment consists of one precipitation source P and the three land use components described above. As mentioned in the previous section, there is also a component for adding and weighting the outflow from the three land use components. Since flows are weighted based on the relative area of the land use class, this type of catchment component can be used even if the catchment for example has no lake or consists only of one big lake.

Reading of driving variables and parameters is accomplished using external C functions which are called

from Catchment. Parameters in the model can also be altered by the user between simulations. In order to manage the parameters in an efficient way, all parameters are stored as inner parameters in Catchment. All models inside Catchment consequently have the parameters which are common for the whole catchment, which is almost all parameters, declared as outer.

Daily temperature and monthly maximum evapotranspiration are calculated in the Catchment component and accessed as outer parameters from the catchment parts. Precipitation has its own component P, a source with three snow outlets and three rain outlets which provides input to the land use components. Catchment also keeps track of which day and month it is in order to assure that the right data is delivered to the other model components.

4 Simulation

When setting up an HBV model for a specific area, Catchments are declared with the appropriate parameter files, PTQW files (driving variables and measured flow and lake water levels) and initial values. They are connected to each other directly or through AddFlows components depending on the geography of the area. The test simulation setup in this study consisted of two catchments connected as shown in figure 4. Since the test areas are small (0.44 and 0.43 km² respectively), the AddFlows component adds the two outflows without any weighting or distribution of the flows over time. The model was run for 1475 time steps, which corresponds to a time period of a little more than four years. An Euler solver with fixed step was used since calculations only need to be performed once every time step.

A rough estimation of execution times was made in order to make sure that the Modelica model performed reasonably well compared to the Fortran version. Without any optimisations, the Modelica model has approximately five to ten times longer execution time than the Fortran model. The model described in this paper does not have very long execution time in either case, so in this application it is not the most important factor to consider when choosing modeling language.

The added outflow from the two catchments, Qout (figure 5), was compared with the corresponding output from the Fortran model. Qout is the runoff response to precipitation which is used for hydrological forecasts and warnings. Some other important model variables, for example the soil moisture storage which

is used for estimating the risk of forest fires, were also compared with Fortran results. The comparisons show that the Modelica model produces the same results as the Fortran version.

5 Experiences and conclusions

The translation from sequential Fortran code to an object oriented approach was rather easy but it was difficult to also translate the code to equations. The greatest use for Modelica is probably in designing new models and making additions to existing models. Some restructuring of the model was done in this project but more needs to be done in order to take full advantage of the Modelica language.

The main benefit of translating the model into Modelica is more readable and reusable code, which facilitates future model development. Modelica has proven quite easy to work with for a non computer scientist with some background in programming. Debugging, though, could be improved since many of the errors that can be encountered are radically different from those encountered in 'normal' imperative programming in for example C.

The test example in this study was quite small, with only two catchments. In most cases the model is run over a larger number of catchment. It would therefore be useful to test the HBV implementation on a larger scale as well.

Since hydrological modeling is performed with many different techniques, further development of general hydrological base classes and components is needed including components for for example physically based modeling. It would also be useful to investigate the behaviour of more complex hydrological models as well in order to see whether we can find the same advantages of using Modelica as modeling language in that respect as we have in the case with the HBV model.

```

model HBV
  import Hydrology;
  Hydrology.HBV.Components.SimpleCatchment SimpleCatchment1
    ( Forest1 (
      SnowPack1 (s (start = 0.0), w (start = 0.0)),
      SoilMoisture1 (w (start = 100.0)),
      Response1 (w (start = 10.0))),
    Field1 (
      SnowPack1 (s (start = 0.0), w (start = 0.0)),
      SoilMoisture1 (w (start = 100.0)),
      Response1 (w (start = 10.0))));
  Hydrology.HBV.Components.StartCatchment StartCatchment1
    ( Forest1 (
      SnowPack1 (s (start = 0.0), w (start = 0.0)),
      SoilMoisture1 (w (start = 55.0)),
      Response1 (w (start = 10.0))),
    Field1 (
      SnowPack1 (s (start = 0.0), w (start = 0.0)),
      SoilMoisture1 (w (start = 55.0)),
      Response1 (w (start = 10.0))));
  Hydrology.Components.TestSource TestSource1;
  Hydrology.Components.Sink Sink1;
  Hydrology.HBV.Components.Add2Flows Add2Flows1;
equation
  connect(TestSource1.qOut1, SimpleCatchment1.qIn1);
  connect(Add2Flows1.qOut1, Sink1.qIn1);
  connect(SimpleCatchment1.qOut1, Add2Flows1.qIn2);
  connect(StartCatchment1.qOut1, Add2Flows1.qIn1);
end HBV;

```

Figure 4: Modelica code for declaration and connection of the HBV model parts

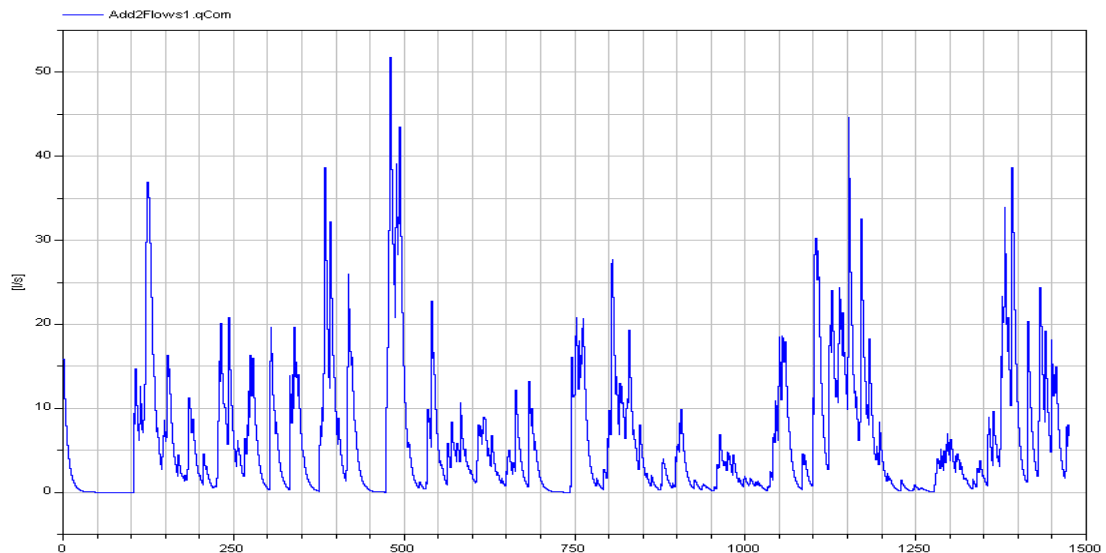


Figure 5: HBV runoff response simulated in Modelica

References

- [1] P. Beater. Modeling and digital simulation of hydraulic systems in design and engineering. education using modelica and hylib. In M. Otter, editor, *Modelica workshop 2000 proceedings*, pages 33–44. The Modelica Association, 2000.

- [2] B. Carlsson, S. Bergström, M. Brandt, and G. Lindström. PULS-modellen - struktur och tillämpningar (the PULSE model - structure and applications, in Swedish), 1987. SMHI Reports Hydrology No.8.
- [3] S.M.O. Fabricius and E. Badreddin. Modelica library for hybrid simulation of mass flow in process plants. In The Modelica Association, editor, *2nd International Modelica Conference proceedings*, pages 225–234. The Modelica Association, 2002.
- [4] G. Lindström, B. Johansson, M. Persson, M. Gardelin, and S. Bergström. Development and test of the distributed hbv-96 hydrological model. *Journal of Hydrology*, 201:272–288, 1997.
- [5] G. Reichl. Wastewater a library for modelling and simulation of wastewater treatment plants in Modelica. In P. Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, pages 171–178. The Modelica Association, 2003.

Visualisation of Model Transformation Algorithms for a Modelica Translator

Peter Harman, Ricardo UK Ltd., Peter.Harman@ricardo.com

Abstract

A software component has been developed to visualise the bipartite graph representing the model structure of a Modelica model. The visualisation is a window on the underlying graph, and therefore the graph transformations appear animated.

This work forms part of research being carried out into improved strategies for simulation of highly discontinuous systems. The primary use is to allow the structure of an equation system to be studied with the aim of categorising the equations according to the type of discontinuity.

This tool has other applications such as debugging or clustering tools. Different layouts and animation methods have been used to maximise the clarity of the visualisation, however large models produce graphs which are too large to view.

1 Introduction

The Modelica specification [1] and existing Modelica tools such as Dymola [2] and OpenSourceModelica [3], make use of graph-theoretical model transformations. The mixture of differential equations and constraint equations lead to a Differential-Algebraic-Equation (DAE) system of unknown index.

Current model transformations [4] have the following objectives:

- Exploit the sparsity of the system of equations and sort the equations into small systems of linear or non-linear equations, this is done in current tools using the Tarjan algorithm [5]
- Ensure the system is of DAE-index 0 or 1, which can be solved by a standard solver, this is done in current tools using the Pantelides algorithm [6,7]
- Aim to reduce the size of any systems of linear or non-linear equations by use of tearing [8] or relaxing [9]

Central to the model transformation algorithms is the concept of representing the system of equations and unknowns as a bipartite graph, with vertices

representing each equation or unknown, and edges to show relationships between them. The tool described here extends this concept to include the hybrid features of a Modelica model, adding vertices to represent conditional expressions and edges to relate these to variables and equations. All variables, including constants, parameters, discrete and continuous states and derivatives, are represented as vertices. This allows all actions to be taken, such as the evaluation of constants, to be represented as transformations of the underlying graph.

Edges in the graph can be undirected or directed. Undirected edges relate a variable with an equation, where the variable can be either on the left-hand-side or the right-hand-side of the equation. Directed edges are used to show a variable that is required by, or is the result of, an algorithm, a function-call or a conditional expression.

2 Visualisation of a Model

Figure 1 shows the model `Modelica.Mechanics.Rotational.Examples.CoupledClutches` in Dymola.

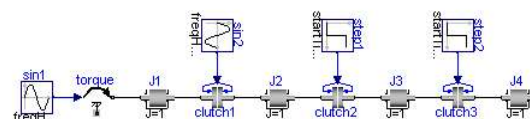


Figure 1: CoupledClutches model schematic

The system of variables, equations and conditional expressions is shown in Figure 2. This seemingly small model has 100 equations and hence the graph is large. In the left-hand column each vertex represents a variable, with the colour representing the variability of the variable. Constants are shown magenta, Parameters blue and time-varying variables are red. In the middle column each vertex represents an equation. In the right-hand column each vertex represents a condition expression, such as `time < sin2.startTime`. The vertices are joined by edges, those with arrows are directed edges with the arrow showing the direction.

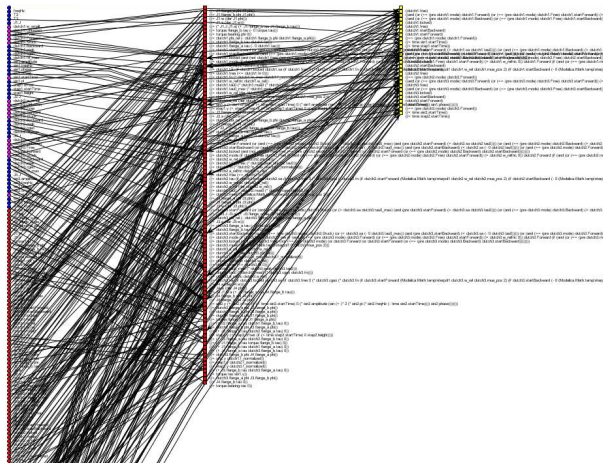


Figure 2: CoupledClutches model equation system

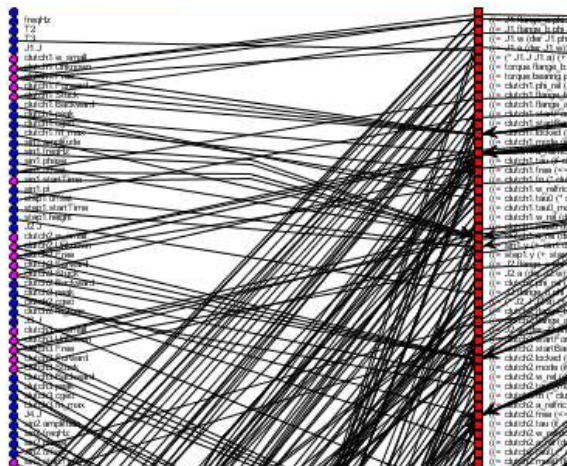


Figure 3: Zoomed-in view of Figure 2

3 Categorisation of Discontinuous Equations

The main application of this work is the development of improved strategies for simulating highly discontinuous systems. It has been shown [4] that an efficient method of simulating a hybrid system is to use a multi-step DAE solver for the smooth continuous parts of the simulation, and to stop and restart the simulation at each discontinuity or event. However, for large highly discontinuous models this is not necessarily efficient. A concept developed is that of *event-density*, the frequency at which discontinuities occur in the model. Event-density rises exponentially with the size of the model. As the time between discontinuities reduces towards the time-step of the simulation, the simulation performance is seriously compromised. Strategies are being developed to allow the simulation to handle events differently depending on the source and type of event. To achieve this aim discontinuous equations must firstly be categorised.

Visualisation of the structure of the system is being used to study interconnection between equations,

conditional expressions and the variables on which the conditions depend. This allows the categorisation of discontinuities into *Local* and *Global*. *Local* discontinuities affect only a small part of the system when they occur, whereas *Global* discontinuities affect the entire system. This categorisation is performed by calculating a measure of the size of the system that is directly connected to the variable in which the discontinuity occurs, i.e. the number of equations and conditional expressions which depend on the variable.

Further categorisation is performed determining whether the discontinuity occurs to a state variable or one of its derivatives.

4 Other Applications

Although this tool has been developed to study the relationships between discontinuous equations and the variables on which they depend, there are other applications during the translation process for which visualisation of the equation structure would be advantageous.

4.1 Model Debugging

Recent work into debugging for equation-based modelling systems [10] has made use of visualisations of graphs to show which part of the model is over or under constrained. A visualisation tool such as this could be used as part of an interactive debugging tool. Many models are too large to visualise in this manner, however a subsection of the model can be shown.

4.2 Assessment of Model Transformation Algorithm Efficiency

By visualising the graph, aspects of a particular model can be shown, such as algebraic loops. Figure 4 shows a visualisation of a small model with a clearly identifiable algebraic loop. In this visualisation, the dark vertices represent equations and the light coloured vertices represent variables. The efficiency of tearing algorithms can be interpreted from the resulting visualisation, as the loop becomes a 'tree' with branches, each branch representing a system of equations to be solved numerically. This is analogous to the sizes of blocks shown in a sparsity plot of the model Jacobian.

The effect of the technique of 'inline integration' [11] on the model structure can also be visualised. This technique inserts equations for integration

algorithms into the equation set before the equations are sorted.

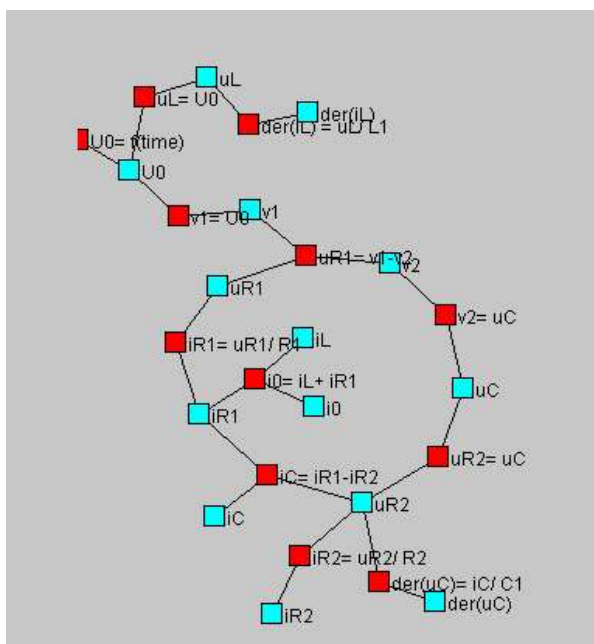


Figure 4: Small model with algebraic loop

4.3 Clustering

Clustering algorithms have been used [12] to divide the Modelica model up into subsystems which can be executed on separate processors in a cluster. This is done after the model equations have been sorted, and requires analysis of the graph to select the subsystems.

An interactive visualization tool could be used at this stage, allowing the user to influence the output by selecting the edges which become points of communication between the processors in the cluster.

4.4 Visualisation of Virtual Connection Graph for Debugging Overdetermined Systems

The technique for translating overdetermined DAE's used by the Modelica MultiBody library, introduced in [13], makes use of a *Virtual Connection Graph*. Each overdetermined type in the model is a vertex in the graph. This vertex can be a root of a spanning tree, a potential root which the system will decide whether it is a root or not, or an ordinary node. There are two types of edge in the graph, breakable or non-breakable, which are defined by `connect` or `Connections.branch` statements.

The translator aims to break the virtual connection graph into spanning trees each with one root. This may not be possible, due to the number of root vertices or non-breakable edges. Visualisation of the

virtual connection graph and the resulting spanning trees would allow the user to identify and correct the source of any problem, such as altering the priority of potential root nodes to control which are selected during translation.

5 ModeliCode

This visualisation tool forms part of an object-oriented framework created for the development of a Modelica to simulation code translator, called ModeliCode. This is written in Java [14]. The graph manipulation and visualisation package uses the Java Universal Network and Graph (JUNG) library [15]. ModeliCode also includes a symbolic computation library to rearrange and differentiate equations, developed using JScheme [16], which allows the mixing of Scheme code and Java classes.

A flexible template-based code-generator is included which allows code to be output in a number of languages.

Currently ModeliCode only translates from flattened models. These can be read from `.mof` files output from Dymola, or can be read via a Corba interface with the OpenModelica Modeq program.

6 Development Issues

6.1 Layout

The layout defines the locations of each vertex. JUNG provides a number of classes for defining the layout. Initial studies used a class called `SpringLayout`, which is analogous to having a spring acting between each vertex. Figure 4 was generated using this layout, showing how it is very good at illustrating an algebraic loop for a very small model. However, as the model size increases this layout makes the graph very hard to read. A new layout class was written to place the vertices in columns according to the object they represent. This matches graphs shown in similar work [10].

6.2 Animation

Clarity is improved by animating the graph. As vertices are removed, such as during the evaluation of constants and parameters, or the removal of alias equations, the vertices in the same column are moved up to fill their spaces. As equations and variables are sorted, the vertices are moved into their new order.

7 Conclusions

By visualizing the model in this way, understanding can be gained of the internal structure of the model. This understanding can be used to develop model transformation algorithms and assess their efficiency, to find errors within the model, or to apply specialized algorithms such as clustering.

A relatively small model can produce graphs that are very large and difficult to read, therefore simple layouts and features such as animation must be used in the viewer to improve clarity.

8 Acknowledgements

Thanks to Professor Seamus Garvey and Dr Atanas Popov at Nottingham University for supervising this project, and to Michael Tiller at Ford Motor Company; Peter Bunus, Kaj Nyström, Håkan Lundvall and Peter Aronsson at Linköping University for useful discussions.

In memory of Dr Pete Lockett, Coventry University.

References

1. Modelica specification, <http://www.modelica.org/documents/ModelicaS pec21.pdf>
2. Dynasim Dymola, <http://www.dynasim.se>
3. Fritzson P., Aronsson P., Bunus P., Engelson V., Saldamli L., Johansson H., Karström A., (2002), "The Open Source Modelica Project", Proceedings Modelica 2002
4. Mattsson SE, Otter M, Elmqvist H, (1999), "Modelica Hybrid Modelling and Efficient Simulation", 38th IEEE Conference on Decision and Control
5. Duff I.S., Erismann A.M., and Reid J.K. (1986), "Direct Methods for Sparse Matrices", Oxford Science Publications
6. Cellier F.E., and Elmqvist H. (1993), "Automated formula manipulation supports object-oriented continuous-system modelling", IEEE Control System Magazine, 13(2)
7. Mattsson S.E., and Söderlind G. (1993), "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives", SIAM Journal on Scientific Computing. Vol. 14
8. Elmqvist H., and Otter M. (1994), "Methods for Tearing Systems of Equations in Object-Oriented Modelling", Proceedings ESM'94 European Simulation Multiconference
9. Otter M., Elmqvist H., and Cellier F.E. (1996), "Relaxing - A Symbolic Sparse Matrix Method Exploiting the Model Structure in Generating Efficient Simulation Code", Keynote Address, CESA'96 IMACS Multiconference, Symposium on Modelling, Analysis and Simulation
10. Bunus P, (2002), "Debugging and Structural Analysis of Declarative Equation-Based Languages" (Licenciate thesis), Department of Computer and Information Science, Linköping University
11. Elmqvist H., Otter M., and Cellier F.E. (1995), "Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems", Keynote Address, Proceedings ESM'95, European Simulation Multiconference
12. Aronsson P., Fritzson P., (2002), "Multiprocessor Scheduling of Simulation Code from Modelica Models", Proceedings Modelica 2002
13. Otter M., Elmqvist H., Mattsson S.E., (2003), "The New Modelica MultiBody Library", Proceedings Modelica 2003
14. JUNG, <http://jung.sf.net>
15. Java, <http://java.sun.com>
16. JScheme, <http://jscheme.sf.net>

Modeling of Interactive Virtual Laboratories with Modelica

Carla Martin Alfonso Urquia Sebastian Dormido

Departamento de Informatica y Automatica, E.T.S. de Ingenieria Informatica, UNED

Juan del Rosal 16, 28040 Madrid, Spain

Abstract

The implementation of virtual-labs supporting runtime and batch interactivity is discussed and it is illustrated by means of several case studies. The virtual-lab *models* have been programmed using Modelica language and translated using Dymola. The virtual-lab *views* (i.e., the user-to-model interfaces) have been implemented using Ejs and Sysquake. This software combination approach allows us to take advantage of the best features of each tool. Ejs and Sysquake capability for building interactive user interfaces composed of graphical elements, whose properties are linked to the model variables. Modelica capability for physical modeling and Dymola capability for simulating hybrid-DAE models.

In order to implement this approach, the following tasks have been completed: (1) a novel modeling methodology, adequate for runtime interactive simulation using Ejs, Simulink and Modelica/Dymola, has been proposed; and (2) a Sysquake to Dymosim interface has been programmed: a set of functions in LME, intended to be used by the Sysquake applications.

1 Introduction

A virtual-lab is a distributed environment of simulation and animation tools, intended to perform the interactive simulation of a mathematical model. Virtual-labs provide a flexible and user-friendly method to define the experiments performed on the model. In particular, interactive virtual-labs are effective pedagogical resources, well suited for web-based and distance education [1].

Typically, the virtual-lab definition includes the following two parts: the *model* and the *view*. The *view* is the user-to-model interface. It is intended to provide a visual representation of the model dynamic behavior and to facilitate the user's interactive actions on the model. The graphical properties of the *view* elements are linked to the *model* variables, producing a bidirectional

flow of information between the *view* and the *model*. Any change of a model variable value is automatically displayed by the view. Reciprocally, any user interaction with the view automatically modifies the value of the corresponding model variable.

Two alternative types of interactivity can be implemented:

- *Runtime interactivity*. The user is allowed to perform actions on the model during the simulation run. He can change the value of the model inputs, parameters and state variables, perceiving instantly how these changes affect to the model dynamic. An arbitrary number of actions can be made on the model during a given simulation run.
- *Batch interactivity*. The user's action triggers the start of the simulation, which is run to completion. During the simulation run, the user is not allowed to interact with the model. Once the simulation run is finished, the results are displayed and a new user's action on the model is allowed.

1.1 Contributions of this paper

The implementation of interactive virtual-labs is discussed in this manuscript. Runtime and batch interactivity are considered. In both cases, the *models* are programmed using Modelica language and translated using Dymola [2]. The *view* of the virtual-labs supporting *runtime interactivity* has been implemented using Easy Java Simulations [3] (abbreviated: Ejs. <http://fem.um.es/Ejs/>). The *view* of the virtual-labs supporting *batch interactivity* has been programmed using Sysquake (<http://www.calerga.com/>).

This software combination approach allow us to take advantage of the best features of each tool. Ejs and Sysquake capability for building interactive user-interfaces composed of graphical elements, whose properties are linked to the model variables. Modelica capability for physical modeling, and finally Dymola capability for simulating hybrid-DAE models.

The tasks completed to successfully implement this approach are discussed. In particular:

- **Runtime interactive simulation.** The communication between the virtual-lab view (programmed using Ejs) and the virtual-lab model (C-code generated by Dymola) is accomplished by using the Ejs-Simulink and the Dymola-Simulink interfaces. The C-code generated by Dymola for the Modelica model can be embedded within a Simulink block [2]. On the other hand, Ejs allows the model to be partially or completely developed using Simulink block diagrams. As a consequence, virtual-labs supporting runtime interactivity can be implemented by combining the use of Ejs, Matlab/Simulink and Modelica/Dymola.

The Modelica model needs to be adequately formulated in order to be: (1) useful as a Simulink block; (2) able to accept information from the virtual-lab view; and (3) able to return information to the virtual-lab view. As a consequence, a modeling methodology has been proposed. It states how a Modelica model can be formulated to suit *runtime interactive simulation*. This methodology has been successfully applied to program a set of virtual-labs for chemical process control. One of them is discussed in this manuscript: the virtual-lab of a double-pipe heat exchanger. Other virtual-labs are discussed in [4, 5, 6].

- **Batch interactive simulation.** A set of Sysquake functions has been programmed to facilitate data exchange between the view and the model of the virtual-lab. These functions synchronize the execution of the *dymosim.exe* file (generated by Dymola) and the Sysquake application. The combined use of Sysquake and Modelica/Dymola for virtual-lab programming is illustrated by means of two case studies.

2 Runtime interactive simulation, by combining the use of Ejs, Simulink and Modelica/Dymola

Easy Java Simulations (Ejs) is an open source, Java-based software tool intended to implement virtual-labs. It can be freely downloaded from the website <http://fem.um.es/Ejs/>. Ejs guides the user in the process of creating the *model* and the *view*, generates

the Java source code of the virtual-lab program, compiles the program, packs the resulting object files into a compressed file, and generates HTML pages containing the virtual-lab as an applet. Then, the user can readily run the virtual-lab and/or publish it on the Internet.

The *view* definition is a strong point of Ejs. Ejs includes a set of ready-to-use visual elements, that the modeller can use to compose a sophisticated view in a simple, drag-and-drop way. The properties of the view elements can be linked to the model variables.

On the contrary, the *model* definition and simulation is a weak point of Ejs. Ejs provides its own procedure to define the model, which must be formulated by the user as a sorted sequence of algorithm clauses (i.e., assignment statements). Ejs implements some standard ODE solvers. However, it implements neither algorithms for symbolic formula manipulation nor algebraic-loop solvers.

Ejs version 3.3 (release 2004) provides a Ejs to Matlab/Simulink interface. Therefore, Ejs 3.3 supports the option of describing and simulating the model using Matlab/Simulink: (1) Matlab code and calls to any Matlab function can be used at any point in the Ejs model; and (2) the Ejs model can be partially or completely developed using Simulink block diagrams. This significantly improves the Ejs capabilities for model description and numerical solution. However, Simulink modeling paradigm (i.e., graphical block-diagram modeling) exhibits some limitations [7]. It requires explicit state models (ODE) and that the blocks have a unidirectional data flow from inputs to outputs. These restrictions strongly condition the modeling task, which requires a considerable effort from the modeller.

The use of Modelica language is an attractive alternative to Simulink, because it reduces considerably the modeling effort and permits better reuse of the models. The combined application of Modelica/Dymola and Ejs to the implementation of virtual-labs is discussed next.

2.1 Combined use of Ejs, Matlab/Simulink and Modelica/Dymola

Dymola 5.0 interface to Simulink 3.0 can be found in Simulink's library browser: DymolaBlock block [2]. This block is an interface to the C-code generated by Dymola for the Modelica code. DymolaBlock block can be connected to other Simulink blocks, and also to other DymolaBlocks blocks, in the Simulink's workspace window. Simulink synchronizes the nu-

merical solution of the complete model, performing the numerical integration of the DymolaBlock blocks together with the other blocks.

In order to make the Modelica model useful as a DymolaBlock block, the computational causality of the Modelica model interface needs to be explicitly set [2]. The input variables are supposed to be calculated from other Simulink blocks, while the output variables are calculated from the Modelica model.

Ejs 3.3 supports the option of describing and simulating the virtual-lab model using Simulink. In this case, the data exchange between the virtual-lab view (composed using Ejs) and the model (Simulink block diagram) is accomplished through the Matlab workspace. The properties of the Ejs' view elements are linked to variables of the Matlab workspace, which can be written and read from the Simulink block diagram.

The Modelica model needs to be built to allow the communication with the virtual-lab view. It needs to support the discontinuous changes in the value of its state variables, parameters and input variables which are the result of the user interaction. In some cases, several choices of the state variables need to be supported simultaneously in the model, in order to provide the user with alternative ways of describing the state changes. A design methodology for the Modelica model is described in Section 2.2. Further details can be found in [4, 6].

2.2 Modeling methodology

The model of a perfect gas is shown in Figure 1. The input flow of gas (F), of heat (Q) and the input temperature (T_{in}) are input variables. The gas volume (V) and the heat capacities (C_p, C_v) are time-independent properties of the physical system.

In general, different choices of the model state-variables are possible. Possible choices in the model shown in Figure 1 include: $e_1 = \{p, T\}$, $e_2 = \{n, T\}$ and $e_3 = \{n, p\}$; where e_i represents one particular choice of the state variables. If the user wants to change interactively p and T , the appropriate choice is $e_1 = \{p, T\}$. This is also the right choice if the user wants to change p and to keep constant T , or if he wants to change T and to keep constant p . Likewise, the appropriate choice is e_2 if the user wants: (1) to modify interactively n and T ; or (2) to modify n and to maintain constant T ; or (3) to modify T and to maintain constant n . An analogous reasoning is applied to e_3 . In general, an interactive model is required to support state changes that correspond with different choices of the state variables.

In addition, interactive changes of the model parameters can have different effects depending on the state variable choice. Consider an instantaneous change in the gas volume (V) of the model shown in Figure 1. If the state variables are $e_1 = \{p, T\}$, then the change in V produces an instantaneous change in the number of moles (n), while the pressure (p) and the temperature (T) remain constant. On the contrary, if the state variables are $e_2 = \{n, T\}$, then the change of volume produces a change of pressure. In this case, the number of moles (n) and the temperature remain constant. As a consequence, the interactive model needs to support different choices of the state variables simultaneously.

An approach to implement this capability is the following. Building the interactive model as composed of several instantiations of the physical model, each one with a different choice of the state variables. When describing an interactive action on the model, the user selects the adequate state-variable choice according to his preference. This information is transmitted from the virtual-lab *view* to the *model*. Then, the interactive model uses the adequate physical-model instantiation (that with the chosen state selection) for executing the instantaneous change in the parameters and state variables, and for solving the re-start problem. Finally, these calculated values are used to re-initialize the other physical-model instantiations. This action guarantees that all physical-model instantiations describe the same trajectory.

Modelica capability for state-selection control allows easy implementation of this approach [8]. Three instantiations of the perfect-gas model (i.e., *perfectGas*) have been defined (see Figure 2): (1) *perfectGasSS1*, with $e = \{p, T\}$; (2) *perfectGasSS2*, with $e = \{n, T\}$; and (3) *perfectGasSS3*, with $e = \{n, p\}$. The Appendix A provides the Modelica code for the perfect-gas model.

Two input variables to the DymolaBlock block are used to carry out the interactive changes in the state: *Istate[:]* and *CKstate[:]* (see Figure 2).

The array *Istate[:]* contains the values used to re-initialize the model state. In the perfect-gas model: $Istate[:] = \{n, p, T\}$.

The array *CKstate[:]* is used to trigger the state re-initialization events, which are performed using the Modelica operator *reinit*. Each variable of the array *CKstate[:]* is used to trigger the events in a different instantiation of the physical model. The perfect-gas model contains three instantiations of the physical-model: *perfectGasSS1*, *perfectGasSS2* and *perfectGasSS3*. Consequently, the array *CKstate[:]* has three

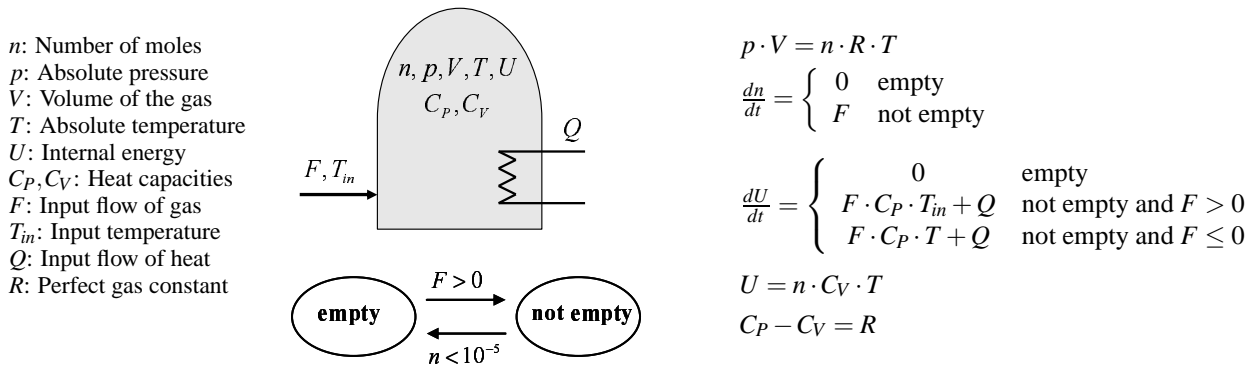


Figure 1: Model of a perfect gas

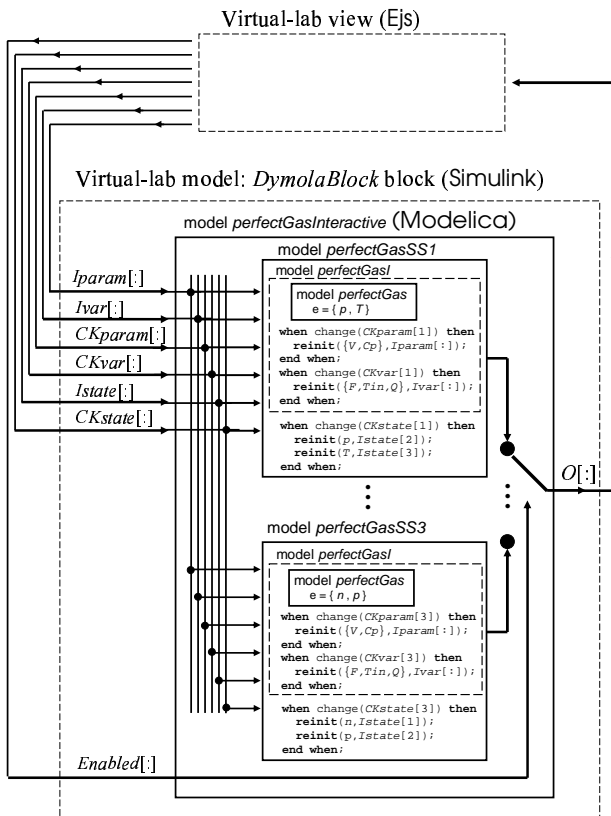


Figure 2: Schematic description of the perfect-gas virtual-lab

components. $CKstate[1]$ triggers the change in the state-variables of *perfectGasSS1*. $CKstate[2]$ and $CKstate[3]$ trigger the change in the state-variables of *perfectGasSS2* and *perfectGasSS3* respectively (see Figure 2).

The interactive parameters (V , C_p) and the input variables (F , T_{in} , Q) are defined as constant state-variables (i.e., with zero time-derivative) in the physical model [4]. Their values are changed by using the *reinit* operator. Four input variables to the DymolaBlock block are used (see Figure 2): two arrays

($Iparam[:]$, $Ivar[:]$) containing the new values, and two arrays ($CKparam[:]$, $CKvar[:]$) for triggering the re-initialization events.

The output-variable array of the DymolaBlock block, $O[:]$ (see Figure 2), contains the variables linked to the properties of the virtual-lab *view*. Ejs uses the value of this output array ($O[:]$) to refresh the simulation view. The value of the input array $Enabled[:]$ is set by Ejs, and it selects which output is connected to the output signal $O[:]$. The output array in the perfect-gas model is the following: $O[:] = \{n, p, T, V, C_p, T_{in}, F, Q\}$.

The Simulink model of the perfect-gas is shown in Figure 3a. The Modelica model (*perfectGasInteractive*) is embedded within the DymolaBlock block. The blocks connected to the DymolaBlock inputs (“MATLAB Fcn” blocks) transmit the value of the input variables from the Matlab workspace to the Simulink block-diagram window. The blocks connected to the DymolaBlock outputs (“To Workspace” blocks) transmit the value of the output variables from the Simulink block-diagram window to the Matlab workspace. Ejs reads the value of these output variables from the Matlab workspace and writes the value of the input variables in the Matlab workspace.

The view of the virtual-lab is shown in Figure 3b. The main window (on the left side) contains the schematic diagram of the process (above) and the control buttons (below). Both of them allow the user to experiment with the model. The vessel volume, represented in the schematic diagram, is linked to the V variable. Its value can be interactively changed by clicking on the hand picture and dragging the mouse. Three radio buttons allow choosing the state variables ($\{p, T\}$, $\{n, T\}$ or $\{n, p\}$). Text fields allow the user set the value of the state variables (n , p , T), the input variables (F , T_{in} , Q) and the parameters (V , C_p). The window placed on the right side of the virtual-lab view contains graphic plots of the model variables.

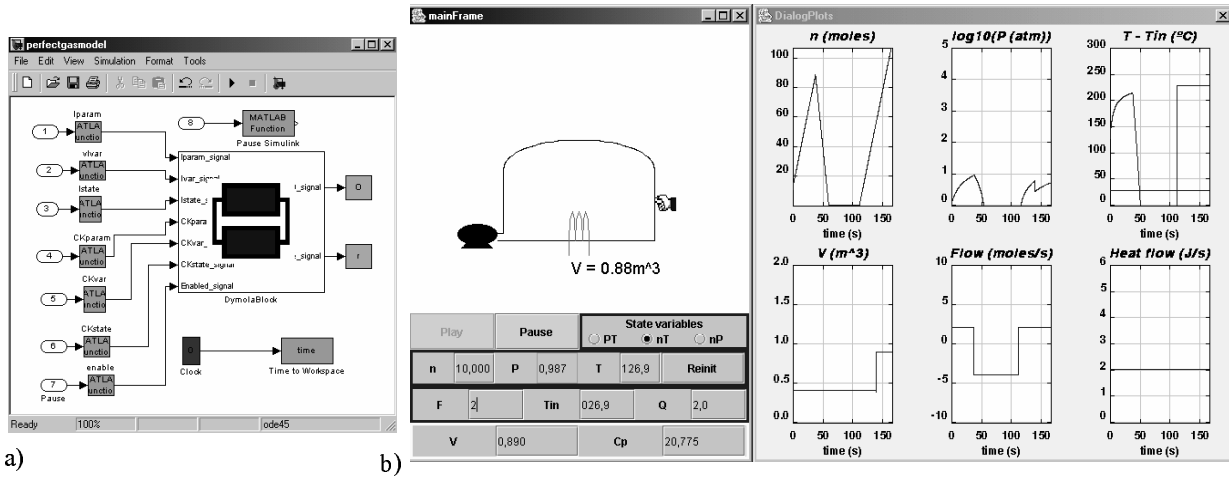


Figure 3: Perfect-gas virtual-lab: a) Simulink model; b) View

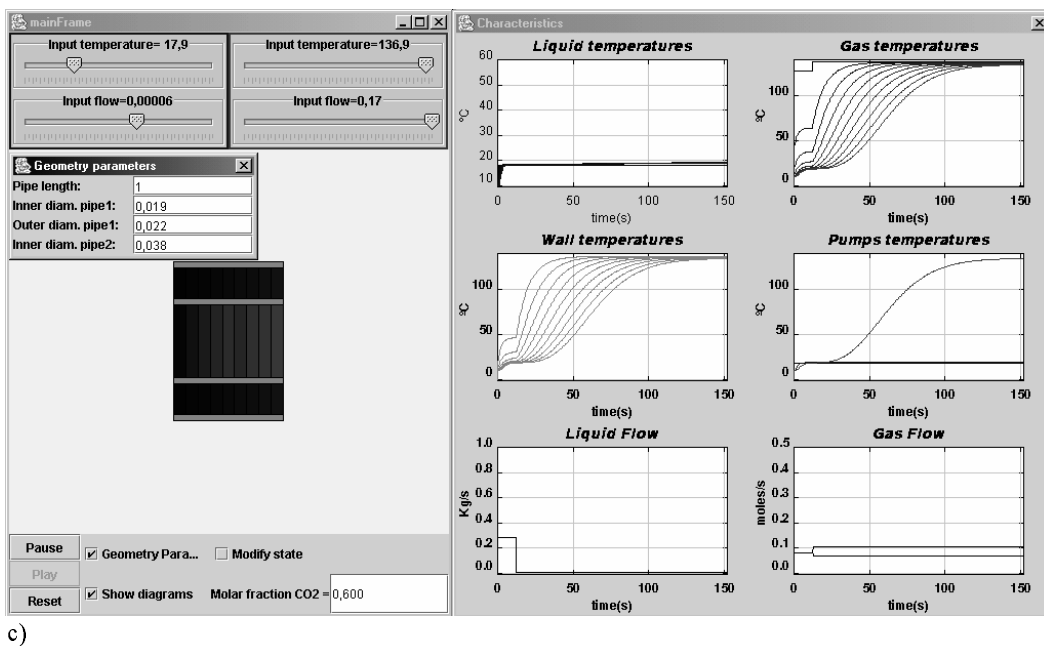
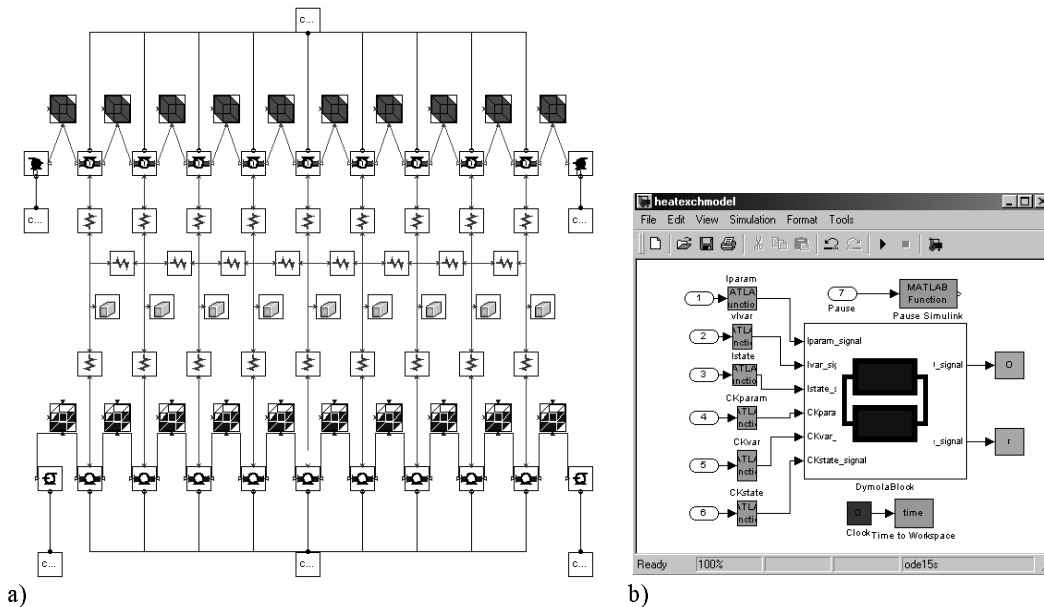


Figure 4: Heat exchanger virtual-lab: a) Physical model; b) Simulink model; c) View

2.3 Case study I: heat exchanger

The interactive simulation of a heat exchanger has been implemented, by the combined use of Ejs, Matlab/Simulink and Modelica/Dymola. A mixture of carbon dioxide and sulfur dioxide is cooled by water in a double-pipe heat exchanger [9]. Two modes of operation are allowed: cocurrent or parallel flow and countercurrent flow. The convective heat transfer on both the tube and shell sides are calculated from the Dittus-Boelter correlation [9]. The center heat exchanger tube is made of copper with a constant thermal conductivity, and the exterior of the steel pipe shell is very well insulated.

The physical model of the heat exchanger has been composed using JARA. The model diagram is shown in Figure 4a. JARA is a set of libraries of some fundamental physical-chemical principles. JARA was originally written in Dymola language [10, 11]. Later on, it was translated into Modelica language. The methodology discussed in Section 2.2 was applied in order to make JARA useful for interactive simulation [5].

JARA is composed of seven model libraries, including models of:

- *Control volumes* containing: (1) an ideal mixture of an arbitrary number of semi-perfect gases; or (2) a homogeneous liquid mixture composed of an arbitrary number of components; or a homogeneous solid. The liquid and gaseous control volumes are considered open systems (i.e., they can exchange mass and heat with their environment) and chemical reactions can take place inside them. The solid control volumes are considered closed systems (i.e., they only exchange energy, not mass, with their environment).
- *Mass transport* due to the pressure and concentration gradient, the gravitational acceleration, chemical reactions, liquid-vapor phase changes, etc.
- *Heat transport* by conduction and convection.

The Simulink model is shown in Figure 4b. The interactive model of the heat exchanger, written in Modelica language, has been embedded within the DymolaBlock block. Observe that the structure of this Simulink model is completely analogous to the perfect-gas model, shown in Figure 3a.

The view of the virtual-lab is shown in Figure 4c. The main window (on the left side) contains: (1) a diagram of the heat exchanger; (2) buttons to control the simulation run (i.e., pause, reset and play); (3) sliders and a

text field to modify the input variables (i.e., liquid and gas flows, liquid and gas input temperatures, and molar fraction of CO_2 and SO_2 in the gas mixture); and (4) checkboxes to show and hide three secondary windows: “*Geometry Parameters*”, “*Modify State*” and “*Characteristics*”.

The “*Geometry Parameters*” window contains text fields that can be used to modify the pipe length and diameters. The controls placed in the “*Modify State*” window allow changing the temperature of the medium inside each control volume (i.e., the cooling liquid, the gas mixture or the metal wall). Finally, “*Characteristics*” is a window with several plots of the model variables.

3 Batch interactive simulation, by combining the use of Sysquake and Modelica/Dymola

Sysquake is a commercial tool intended to develop interactive applications [12]. It is based on LME, an interpreter specialized for numerical computation. LME is mostly compatible with the language of MATLAB(R) 4.x and it includes many features of MATLAB 5 to 7. It implements graphic functions specific to dynamic systems (such as step responses and frequency responses) and general purpose functions used for displaying any kind of data.

Typically, a Sysquake application contains several interactive graphics, which are displayed simultaneously. These graphics contain elements that can be manipulated using the mouse. While one of these elements is being manipulated, the other graphics are automatically updated to reflect this change. The content represented by each graphic, and its dependence with respect to the content of the other graphics, is programmed using LME.

The main goal of Sysquake is the interactive manipulation of graphics. The user can define functions, called *handlers*, intended to perform different tasks managed by Sysquake. These tasks include the model initialization, manipulation of figures and selection of menus.

As input and output, the *handlers* use variables as well as values managed directly by Sysquake, such as the position of the mouse. Therefore, only the code necessary for displaying the figures and processing manipulations from the user is required. This results in small scripts, developed quickly and easy to maintain.

LME can be extended by libraries, composed of related functions written in LME, or by extensions de-

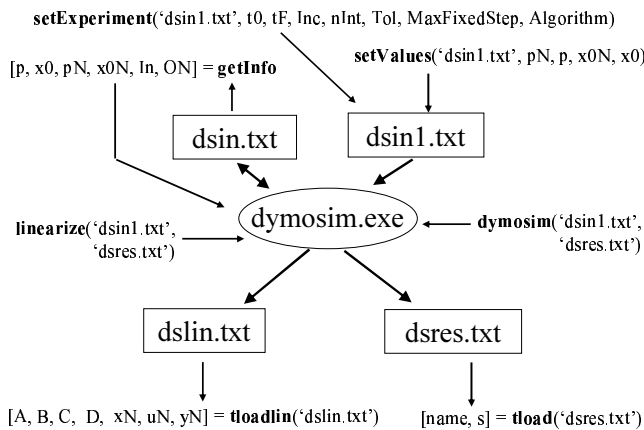


Figure 5: Sysquake-Dymosim interface functions

veloped with standard compilers.

3.1 Combined use of Sysquake and Modelica/Dymola

A Sysquake interface to Dymosim (i.e., the executable file generated by Dymola [2]) has been programmed. This interface is a set of functions in LME, intended to be used by the Sysquake applications. These functions perform the following tasks:

- The *setExperiment* and *setValues* functions write the experiment description to a text file. This text file is intended to be the input file for *dymosim.exe*.
- The *dymosim* and *linearize* functions execute the *dymosim.exe* file in order to simulate and linearize the Modelica model respectively.
- The *tload* and *tloadlin* functions: (1) read the output file generated by *dymosim.exe* after a model simulation or linearization respectively; and (2) save these results as variables to the Sysquake workspace. These variables can be used by Sysquake applications.

Next, a brief description of each function is provided (see Figure 5):

- *setExperiment(txtFile, StartTime, StopTime, Increment, nInterval, Tolerance, MaxFixedStep, Algorithm)*. It writes to the *txtFile* text file (default file name: *dsin1.txt*) the simulation parameters.
- $[p, x0, pN, x0N, InputN, outputN] = getInfo$. This function executes the *dymosim.exe* file (command *dymosim -i*) in order to generate the Dymosim input file (*dsin.txt*). In addition, this function reads

the names of the model variables (i.e., inputs, outputs, parameters, states) and their default values from *dsin.txt* file, and saves them as variables to the Sysquake workspace.

- *SetValues(txtFile, pN, p, x0N, x0)*. The name and the value of the model parameters and state variables are written to the *txtFile* text file (*dsin1.txt* by default).
- *dymosim(iFile, oFile)*. This function executes the following command: *dymosim -d dsin.txt iFile oFile*. The default file name for *iFile* and *oFile* is *dsin1.txt* and *dsres.txt* respectively.
- *linearize(iFile, oFile)*. This function obtains the linearized model by executing the command: *dymosim -l iFile oFile*. The default file name for *iFile* and *oFile* is *dsin1.txt* and *dslin.txt* respectively.
- $[N, s] = tload(oFile)$. This function reads the result file, *oFile* (default file name: *dsres.txt*), and stores the signal names and the simulation results into *N* (text matrix) and *s* (numeric matrix) respectively.
- $[A, B, C, D, xN, uN, yN] = tloadlin(txtfile)$. It loads the linear model generated by dymosim from the *txtfile* result file (default file name: *dslin.txt*) into the Sysquake workspace.

Next, two case studies are provided to illustrate the use of this Sysquake-Dymosim interface.

3.2 Case study II: control loop

The interactive simulation of the control loop shown in Figure 6 is implemented by combining the use of Sysquake and Modelica/Dymola. The constitutive relation of the hysteresis-based controller is shown in Figure 7. The setpoint is the composition of two signals: a piecewise linear function and a sine function. The model of the control loop has been programmed using Modelica language and translated using Dymola. The execution of the *dymosim.exe* file generated by Dymola is controlled by the Sysquake application (i.e., the virtual-lab *view*).

The *view* of the virtual-lab is the Sysquake application shown in Figure 8. It is composed of four graphics. Three of them are interactive:

- “Constitutive relation” plot (graphic on the upper left). The position of the $\{a, b, c, d, e, f\}$ points of

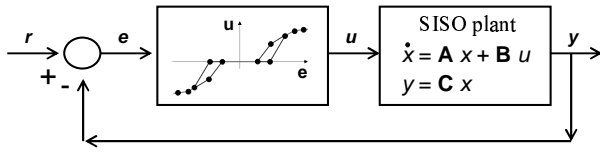


Figure 6: Control loop

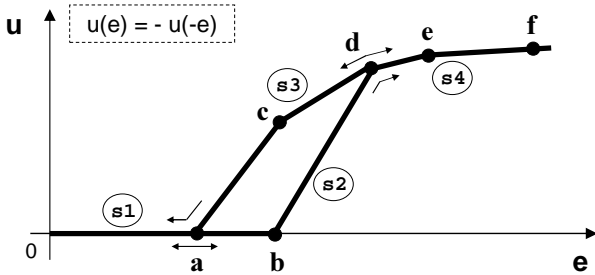


Figure 7: Constitutive relation of the controller

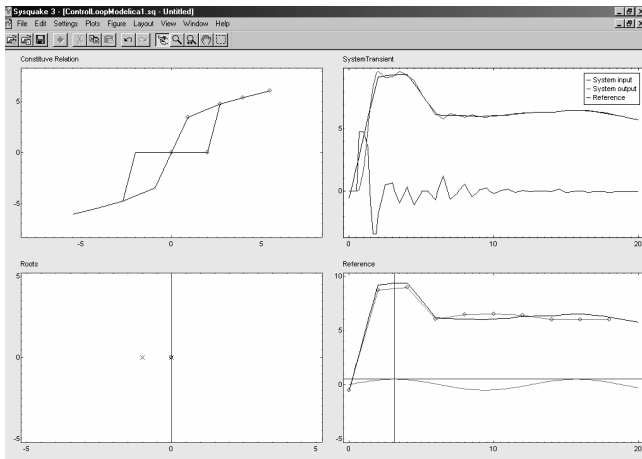


Figure 8: View of the control loop virtual-lab

the controller constitutive relation can be changed by dragging the mouse.

- “Roots” plot (graphic on the lower left). The plant zeros and poles can be changed by clicking on the circles and crosses and dragging the mouse.
- “Reference” plot (graphic on the lower right). The shape of the piecewise linear function and the amplitude and frequency of the sine function can be modified by clicking on the lines and circles that appear in the graphic and dragging the mouse.

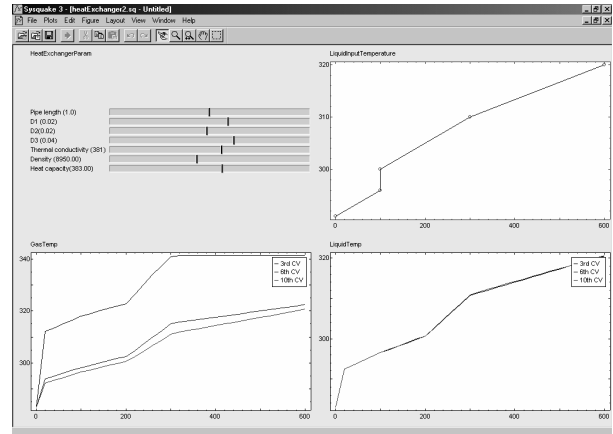


Figure 9: View of the heat exchanger virtual-lab

3.3 Case study III: heat exchanger

The heat exchanger virtual-lab described in Section 2.3 supports runtime interactivity. It was implemented using Ejs, Simulink and Modelica/Dymola. In this section, the heat exchanger model is revisited, and a virtual-lab supporting batch interactivity is programmed by combining the use of Sysquake and Modelica/Dymola.

The *view* of the virtual-lab is the Sysquake application shown in Figure 9. The sliders placed on the upper left side allow modifying some model parameters: the pipe length and diameters, and the thermal parameters of the center heat-exchanger tube.

The graphic on the upper right corner is interactive. It represents the time-evolution of the inlet temperature of the water. The shape of this curve can be changed by clicking on one of the points and dragging the mouse.

The graphics on the lower side of Figure 9 show the time-evolution of the temperature at certain positions of the tube and the shell.

4 Conclusions

The feasibility of combining Modelica/Dymola with Ejs and Sysquake, for implementing runtime and batch interactive simulations respectively, has been demonstrated. Ejs and Sysquake are software tools intended to develop interactive applications. Their strong point is the programming of the virtual-lab *view*. Working together with Modelica/Dymola significantly improves the Ejs and Sysquake capabilities for *model* description and simulation. The use of Modelica language reduces considerably the modeling effort.

In order to implement this software combination ap-

proach, a modeling methodology has been proposed and a Sysquake-Dymosim interface has been programmed. Several case studies of virtual-labs supporting runtime and batch interactivity have been discussed.

Acknowledgements

The authors wish to thank Dr. Yves Piguet (Calerga Sarl, Lausanne, CH) for his constructive comments. This work has been supported by the Spanish CICYT, under DPI2001-1012 and DPI2004-1804 grants.

References

- [1] Dormido S. Control learning: Present and Future. In: Annual Reviews in Control, vol. 28, pp. 115–136, 2004.
- [2] Dynasim AB. Dymola. User's Manual. Version 5.0a. Dynasim AB. Lund, Sweden.
- [3] Esquembre F. Easy Java Simulations: a Software Tool to Create Scientific Simulations in Java. In: Computer Physics Communications, vol. 156, pp. 199–204, 2004.
- [4] Martin C, Urquia A, Sanchez J, Dormido S, Esquembre F, Guzman J.L, Berenguel M. Interactive Simulation of Object-Oriented Hybrid Models, by Combined use of Ejs, Matlab/Simulink and Modelica/Dymola. In: Proc. 18th European Simulation Multiconference, pp. 210–215, 2004.
- [5] Martin C, Urquia A, Dormido S. JARA 2i - A Modelica Library for Interactive Simulation of Physical-Chemical Processes. In: Proc. European Simulation and Modelling Conference, pp. 128–132, 2004.
- [6] Martin C, Urquia A, Dormido S. Object-Oriented Modeling of Virtual Laboratories for Control Education. 16th IFAC World Congress, Praha, Czech Republic, July 2005. Accepted.
- [7] Astrom K.J, Elmqvist H, Mattsson S.E. Evolution of Continuous-Time Modeling and Simulation. In: Proc. of the 12th European Simulation Multiconference, Manchester, UK, 1998.
- [8] Otter M, Olsson H. New features in Modelica 2.0. In: Proc. 2nd International Modelica Conference, pp. 7.1–7.12, 2002.
- [9] Cutlip M.B, Shacham M. Problem Solving in Chemical Engineering with Numerical Methods. Prentice-Hall, 1999.
- [10] Urquia A. Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el Ámbito del Control de Procesos Químicos. PhD. Thesis. Dept. Informática y Automática, UNED, Madrid, Spain, 2000.
- [11] Urquia A, Dormido S. Object-Oriented Design of Reusable Model Libraries of Hybrid Dynamic Systems. Mathematical and Computer Modelling of Dynamical Systems, 9(1), pp. 65–118, 2003.
- [12] Calerga Sarl. Sysquake 3. User's Manual. Calerga Sarl. Lausanne, Switzerland.

APPENDIX A: Modelica code for the perfect-gas model

```

model perfectGas
  parameter Boolean nIsState, pIsState, TIsState;
  Real n (unit="mol", start=20,
          stateSelect = if nIsState
                        then StateSelect.always
                        else StateSelect.default);
  Real p (unit="N.m-2", start=1e5,
          stateSelect = if pIsState
                        then StateSelect.always
                        else StateSelect.default);
  Real T (unit="K", start=300,
          stateSelect = if TIsState
                        then StateSelect.always
                        else StateSelect.default);

  Real V (unit="m3", start=1);
  Real Cp (unit="J/(Kg.K)", start=5*R/2);
  Real Cv (unit="J/(Kg.K)");
  Real F (unit="mol.s-1");
  Real Tin (unit="K");
  Real Q (unit="J.s-1");
  parameter Real R (unit="J/(mol.K)") = 8.31;
protected
  Real U (unit="J", stateSelect = StateSelect.never);
  Boolean empty (start=false);
equation
  // Interactive parameters
  der(V) = 0;
  der(Cp) = 0;
  // Input variables
  der(F) = 0;
  der(Tin) = 0;
  der(Q) = 0;
  // State equation
  p * V = n * R * T;
  // Mol balance
  der(n) = if empty then 0 else F;
  // Energy balance
  der(U) = if empty then 0
            else if F>0 then F*Cp*Tin+Q else F*Cp*T+Q;
  // Internal energy
  U = n * Cv * T;
  // Mayer law
  Cp - Cv = R;
  // Empty-vessel condition

```

```

when F > 0 and pre(empty) or
  n < 1e-5 and not pre(empty) then
  empty = not pre(empty);
end when;
end perfectGas;

model perfectGasI
  extends perfectGas;
  Modelica.Blocks.Interfaces.InPort Iparam (n=2);
  Modelica.Blocks.Interfaces.InPort Ivar (n=3);
  Modelica.Blocks.Interfaces.InPort Istate (n=3);
  Real CKparam;
  Real CKvar;
  Real CKstate;
  Modelica.Blocks.Interfaces.OutPort O (n=8);
protected
  Boolean CKparamIs0 (start = true, fixed=true);
  Boolean CKvarIs0 (start = true, fixed=true);
  Boolean CKstateIs0 (start = true, fixed=true);
equation
  // Interactive change of the parameters
  when CKparam>0.5 and pre(CKparamIs0) or
    CKparam<0.5 and not pre(CKparamIs0) then
    CKparamIs0 = CKparam < 0.5;
    reinit(V, Iparam.signal[1]);
    reinit(Cp, Iparam.signal[2]);
  end when;
  // Interactive change of the input variables
  when CKvar>0.5 and pre(CKvarIs0) or
    CKvar<0.5 and not pre(CKvarIs0) then
    CKvarIs0 = CKvar < 0.5;
    reinit(F, Ivar.signal[1]);
    reinit(Tin, Ivar.signal[2]);
    reinit(Q, Ivar.signal[3]);
  end when;
  // Output signal
  O.signal = { n, p, T, V, Cp, Tin, F, Q };
end perfectGasI;

model perfectGasSS1
  extends perfectGasI (nIsState=false,
    pIsState=true,
    TIsState=true);
equation
  // Interactive change of the state variables
  when CKstate>0.5 and pre(CKstateIs0) or
    CKstate<0.5 and not pre(CKstateIs0) then
    CKstateIs0 = CKstate < 0.5;
    reinit(p, Istate.signal[2]);
    reinit(T, Istate.signal[3]);
  end when;
end perfectGasSS1;

model perfectGasSS2
  extends perfectGasI (nIsState=true,
    pIsState=false,
    TIsState=true);
equation
  // Interactive change of the state variables
  when CKstate>0.5 and pre(CKstateIs0) or
    CKstate<0.5 and not pre(CKstateIs0) then
    CKstateIs0 = CKstate < 0.5;
    reinit(n, Istate.signal[1]);
    reinit(T, Istate.signal[3]);
  end when;
end perfectGasSS2;

model perfectGasSS3
  extends perfectGasI (nIsState=true,
    pIsState=true,
    TIsState=false);
equation
  // Interactive change of the state variables
  when CKstate>0.5 and pre(CKstateIs0) or
    CKstate<0.5 and not pre(CKstateIs0) then
    CKstateIs0 = CKstate < 0.5;
    reinit(n, Istate.signal[1]);
    reinit(p, Istate.signal[2]);
  end when;
end perfectGasSS3;

model perfectGasInteractive
  Modelica.Blocks.Interfaces.InPort Iparam (n=2);
  Modelica.Blocks.Interfaces.InPort Ivar (n=3);
  Modelica.Blocks.Interfaces.InPort Istate (n=3);
  Modelica.Blocks.Interfaces.InPort CKparam (n=3);
  Modelica.Blocks.Interfaces.InPort CKvar (n=3);
  Modelica.Blocks.Interfaces.InPort CKstate (n=3);
  Modelica.Blocks.Interfaces.InPort Enabled (n=3);
  Modelica.Blocks.Interfaces.OutPort O (n=8);
  Modelica.Blocks.Interfaces.OutPort Release(n=1);
  perfectGasSS1 SS1 (CKparam = CKparam.signal[1],
    CKvar = CKvar.signal[1],
    CKstate = CKstate.signal[1]);
  perfectGasSS2 SS2 (CKparam = CKparam.signal[2],
    CKvar = CKvar.signal[2],
    CKstate = CKstate.signal[2]);
  perfectGasSS3 SS3 (CKparam = CKparam.signal[3],
    CKvar = CKvar.signal[3],
    CKstate = CKstate.signal[3]);
equation
  connect(Iparam, SS1.Iparam);
  connect(Istate, SS1.Istate);
  connect(Ivar, SS1.Ivar);
  connect(Iparam, SS2.Iparam);
  connect(Istate, SS2.Istate);
  connect(Ivar, SS2.Ivar);
  connect(Iparam, SS3.Iparam);
  connect(Istate, SS3.Istate);
  connect(Ivar, SS3.Ivar);
  Release.signal = {4,0};
  O.signal = if Enabled.signal[1] > 0.5
    then SS1.O.signal
    else if Enabled.signal[2] > 0.5
    then SS2.O.signal
    else if Enabled.signal[3] > 0.5
    then SS3.O.signal
    else zeros(size(O.signal, 1));
end perfectGasInteractive;

```

Parallelization in Modelica

Kaj Nyström Peter Aronsson Peter Fritzson
 Linköping University, Sweden
 {kajny,petar,petfr}@ida.liu.se

Abstract

The better the computer, the larger and more precise simulations can be carried out, and the more beneficent modeling can be. It is well known that faster computers enable more precise and computationally expensive simulations to be carried out, which allow more pre-cise mathematical models. This paper gives an overview of certain methods for expanding the limits of what can be done in the area of simulation by parallelizing simulations based on Modelica [18, 16] models. This is an efficient and less expensive way of achieving better simulation performance.

In the following, we will restrict ourselves to describing various ways of parallelizing a simulation in Modelica, ranging from coarse grained high level parallelization to fine grained task merging at a very low level. It is very difficult to say which approach is the most successful or promising since little research has been done in most of the subareas of parallelizing Modelica models. Task merging seems to be the most developed approach and does give significant performance increases [1] but the other areas are largely unexplored. We can therefore only guess that based on parallelization research in other areas, there is little to gain for a normal user in parallelizing a small simulation. Larger, more complex simulations on the other hand can benefit greatly from parallelization, especially if it can be done automatically.

Keywords: Modelica, parallelization, task merging, transmission line modeling, weak connectors

1 Introduction

Since the advent of computers, there has always existed a need for more computational power. In fact, the peak performance of a computer system effectively sets a limit to what the user can actually do. For a Modelica user, the amount of computational power available at simulation time is what determines what can be simulated. Obviously, if we can simulate more

complex applications, the use of modeling and simulation is promoted.

There are four different ways which a user can go about in order to faster be able to simulate a more complex physical structure:

1. Buy a faster computer
2. Optimize the model for faster simulation
3. Optimize the compiler
4. Parallelize the model, distributing simulation across many processors.

While option one can sometimes be a viable alternative, it is so only up to a certain point. There is a limit to how fast computers are available, even if you have the financial resources to update your computer every month.

Option two might very well be the issue for a whole series of articles all by itself. Still, experience show that there is a limit to the performance gain which can be obtained also from model optimization.

Option three is an interesting item as it can potentially boost performance of every model compiled with that compiler. The amount of performance gain which can be obtained by compiler optimization is however also quite limited.

This paper will focus on item four, parallelization of the model. The model is assumed to be written in Modelica and to be of a size and complexity which makes it nontrivial to simulate within reasonable time on a single powerful computer. Apart from this, we make no assumptions whatsoever on the model structure, composition or domain.

2 Parallelization in General

Parallelization of computational problems has been an issue for as long as there has been computers. Regardless of how fast current computers are, there has always been applications where this is simply not

enough. Parallelization of the problem is then the only viable solution. The problem is to get good performance while distributing the computation across several CPUs. Communication between jobs can, even if carefully implemented cause severe delays in the computation because the necessary data may not be available when it is needed. Also, simple program branches such as if-statements which decide which statement should be executed and not must be treated with great care so that the right program path is taken. These, and several other issues must always be taken into account when trying to parallelize an application. This is even more important when trying to parallelize a general class of applications, such as Modelica models.

3 Parallelization in Modelica

We will in this paper try to summarize past, present and future research advances within the area of parallelization of models expressed in the Modelica language. We have divided the subject into three parts or levels, depending on where the parallelization is applied. The high level parallelization tries to partition the model on the Modelica source code level. The medium level deals with the numerical solvers while the low level parallelization deals with the solver generated code.

Before we begin, we would like to make one observation which makes the task of parallelizing Modelica even more challenging than with other languages, for example C or Fortran. Modelica developers are normally not experts in parallel programming. In fact, they are usually not computer scientists at all, but instead domain experts within one or many specific fields using physical modeling. This demands a lot of the parallelization framework since the user (that is, the Modelica modeler) can not really take an active part in the parallelization. This means that the parallelization must be automatically performed, without user interaction to a great extent. If this is not possible then at least the user interaction must be minimized and formulated in a way that makes sense even to a user with no experience whatsoever in parallel computing.

4 High Level Parallelization

High level parallelization, as stated previously deals with the problem of parallelizing models at the Modelica source code level. In general, this means that the Modelica language itself is extended or modified

in some way in order to allow the user to provide the compiler with directions on how to parallelize the simulation. In comparison with other high level languages, the Modelica language has some interesting properties which can be used to our advantage when trying to parallelize Modelica simulations.

The most interesting property is probably the connection construct. A Modelica model almost always consists of a multitude of components with connections between them. The connections define an explicit interface between components which is quite useful when considering how to best partition the model. Indeed, both of the two high level parallelization methods we know about use connections in one way or another.

4.1 The Transmission Line Modeling Method

The transmission line modeling (TLM) method [6] is derived from two ideas. First, that many models can be viewed as a continuous transmission line which propagates information. Second, that the information being propagated in time step $t - 1$ in many cases does not differ much from the information propagated in step time step t . This means that we can reuse information received in time step $t - 1$ in the calculations for time t , thus reducing the amount of communication needed between partitions in the model. While we do introduce an error in the model by reusing values from earlier time step, this error is mathematically decidable and it is possible to reduce the amount of value reuse and thus reduce the error introduced. The transmission line modeling method is not yet implemented in any Modelica implementation that we know of although an implementation of the TLM method has been planned in the GridModelica project [5] for 2005.

4.2 The Weak Connectors Method

Introduced by [7] this somewhat less explored method also deals with connections. By introducing the concept of weak connections, the model can be partitioned in two or more parts. The idea is to separate fast subsystems from slow so that different solver step size, or even different solvers can be used when solving the system. The difficult part here is to find good places to insert the weak connection, instead of a normal connection. Such places frequently occur between domain boundaries and while these could quite easily be identified by a domain expert, it is not so easy to find them automatically, which is of course the desirable method. One way of doing this could be to exploit the package

structure of Modelica which roughly divides components into different domains.

4.3 Other high level parallelization methods

There are a some high level parallelization techniques in traditional parallel programming that could be adapted to Modelica. One such technique is matrix operation partitioning. Matrixes and vectors represent large data chunks upon which operations are executed. One example operation could be to add one to each element in the matrix. Such an operation could quite easily be distributed across several CPUs as the individual operations of adding one to element $m[i][j]$ is independent from the operation of adding one to element $m[k][l]$.

In the same way, parts of normal loop parallelization techniques could probably be employed to achieve parallelization in Modelica. For example High Performance Fortran (HPF)[11] has the *forall*, *pure* and *independent* keywords which gives the compiler directions on how to parallelize loops in the program. Even though these constructs could quite easily be introduced in the Modelica language, it is unsure whether they will provide the same performance boost as they do in HPF due to Modelicas radically different execution model.

5 Medium level parallelization

The next level of parallelization is at the equation system and numerical solver level. Parallel solvers have in the past had problems with numerical stability in comparison with other state-of-the-art solvers. Thus, limiting the usage of such solvers to specific domains where the requirement on the numerical stability of the solver is not too demanding. Parallelizing numerical solvers is in itself a very complex task and while an interesting way to achieve additional parallelism, for example with algorithms such as [12, 13] it is not really Modelica specific. We shall therefore in this paper concentrate on other ways of equation parallelization. Another interesting solver related technique is the mixed mode integration technique presented in [4]. It is a compromise between explicit and implicit integration, done by splitting fast and slow subsystems in a model and to apply implicit discretization only to the fast part. Results presented in [4] indicates performance increases ranging from four to sixteen times. One task that could be parallelized without too much effort be parallelized is the Jacobian calculation. Ja-

cobian calculation is sometimes necessary when using an implicit ODE solver and its calculation is side effect free which makes the amount of interCPU communication small. Related to this, it is possible to achieve some degree of parallelism in the calculation of the states in an ODE or a DAE, meaning function f in the ODE system 1 and functions f, g in the DAE system 2.

$$\dot{X} = f(X, t) \quad (1)$$

$$f(\dot{X}, X, Y, t) = 0, g(X, Y, Z) = 0 \quad (2)$$

It might also be possible to conduct parallelize solving of equation system in some cases, as done in [14]. Even though it is common that subcomponents in an equation system depend upon each other in a linear fashion, it does not have to be so. What has to be done is to build a task dependency graph and determine if subsystems can be solved simultaneously and pass this to a task scheduler which then distributes the tasks. Scheduling and partitioning algorithms as described in [2] also belongs on this level. In that paper only static scheduling algorithms are described and while these work very well for continuous systems, they will not work with hybrid models, meaning models that contain both continuous and discrete parts. In such hybrid systems, discrete events can radically change the behavior of the system so in that case, we need to use dynamic scheduling instead.

6 Low level parallelization

While the difference between medium and low level parallelization might be hard to define, we have in this paper drawn the line at the data level at which the parallelization algorithms work. With low level parallelization, the object is to parallelize the compiler generated simulation code. We will refrain from describing low level parallelization here since it is already thoroughly described in [3].

7 Discussion

Parallelization in Modelica is still very much underdeveloped, with the possible exceptions of the low level parallelization and solver integration. This is perhaps somewhat surprising since physical modeling and simulation is one of the areas with the strongest demand for more computational power. While there exist some parameter study applications [10, 9, 8], real

parallelization of Modelica models is still to a great extent an open issue to be explored.

While the above mentioned techniques can probably be applied separately with good results, even better results can be expected if they are combined together. For instance, parallelization of the model with the TLM method can be combined with task merging in the lower layer to achieve a coarse grained parallelization at Modelica source level while achieving a more fine grained parallelization at lower level.

To conclude, we think that while a modeling language perhaps does not live or die with its parallelization abilities, it is still important to develop parallelization in order to make the Modelica language a serious competitor to Fortran, C and C++ also when it comes to simulation of computationally demanding models.

8 Acknowledgement

This work was sponsored by Vinnova[19] via the GridModelica[5] project.

References

- [1] Aronsson P., Fritzson P. , Task Merging and Replication using Graph Rewriting, Tenth International Workshop on Compilers for Parallel Computers, Amsterdam, the Netherlands, Jan 8-10, 2003
- [2] Aronsson P., Fritzson P. *Clustering and Scheduling of Simulation Code Generated from Equation Based Simulation Languages* 9th Workshop on Compilers for Parallel Computers, CPC 2001, June 27-29, Edinburgh, Scotland, UK
- [3] Aronsson P. Fritzson P. *A Task Merging Technique for Parallelization of Modelica Models* Modelica conference 2005, Hamburg, Germany 2005.
- [4] Schiela A. Olsson H. *Mixed-mode Integration for Real-time Simulation* Proceedings of the Modelica Workshop 2000, pp 69-75.
- [5] The GridModelica project, <http://www.ida.liu.se/labs/pelab/modelica/GridModelica.html>, accessed 2004-12-06.
- [6] Christopoulos C. , *The Transmission Line Modeling Method*, EEE/OUP Series on Electromagnetic Wave Theory, 1995
- [7] Casella F. , Maffezzoni C. : "Exploiting Weak Interactions in Object Oriented Modeling", EUROSIM Simulation News Europe, Mar. 1998, pp. 8-10.
- [8] Nyström K, Aronsson P, Fritzson P. *GridModelica - A modeling and simulation framework for the grid* Proceedings of SIMS conference, Copenhagen, Denmark 2004.
- [9] Engelson V, Fritzson P. *A Distributed Simulation Environment* Proceedings of SIMS 2002 conference.
- [10] Joos H-D, Looye G, Moormann D. *Design of Robust Inversion Control Laws using Multi-Objective Optimization* AIAA Guidance and Control Conference, Montreal, Canada, 2001.
- [11] High Performance Fortran Forum *High Performance Fortran Language Specification, version 1.0* Houston, Texas, 1993
- [12] Yi-Ling F. Chiang, Ji-Suing Ma, Kuo-Lin Hu and Chia-Yo Chang *Parallel multischeme computation* Journal of scientific computing, 3(3):289-306, 1988
- [13] Gilbert C. Sih and Edward A. Lee *Declustering: A new multiprocessor scheduling technique* IEEE Transactions on Parallel and Distributed Systems, 4(6):625-637, June 1993
- [14] Andersson Niklas *Compilation of Mathematical Models to Parallel code* Licentiate thesis, Linköping University, 1996
- [15] *The OpenModelica project.* <http://www.ida.liu.se/pelab/modelica/OpenModelica.html>
- [16] *The Modelica Association.* <http://www.modelica.org>.
- [17] Fritzson P, Aronsson O, Bunus P, Engelson V, Johansson H, Karström A, Saldamli L. *The Open Source Modelica Project.* In Proceedings of the 2nd International Modelica Conference, Munich Germany, 18-19 2002.
- [18] Fritzson P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, 2004.
- [19] Vinnova <http://www.vinnova.se>

A Simulation Management Environment for Dymola

Sven-Erik Pohl* Jörg Ungethüm†

German Aerospace Center (DLR), Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart

Abstract

Building Modelica libraries is a steady process of adding and refining models. There is rarely a final version of a library. This leads to the fact that simulation results are difficult to reproduce due to changes in sub-models. However, reproducible simulation results have to be provided for solid project work and scientific research. Using Matlab as a platform a simulation management environment, called SiME, was developed. This environment includes general simulation handling, as well as tools for pre- and postprocessing. A HTML-based simulation history containing parameters and results is included.

Keywords: Modelica tools, simulation management, Matlab, CVS

1 Introduction

Developing Modelica libraries while at the same time maintaining several complex models can make backward compatibility difficult. Moreover, keeping specific library versions is especially important if simulation results were published in research reports or scientific publications. A common practice in software engineering, a version control system appears to be an adequate solution to manage the code evolution. However, if several libraries are involved in the simulation keeping track of the files becomes tedious. Additionally, linking the versions with the results has to be done manually.

2 Objectives

A simulation management environment was outlined to overcome this shortages. The central idea is to automate and standardize the versioning process. It is

necessary to apply versioning not only to the Modelica libraries but also to any auxiliary files like pre- and postprocessing scripts, which are necessary to run the simulation. The ability for recovering the model and rerun the simulation is sensible against the completeness and version-correctness of these auxiliary files. Multiple simulation projects must be supported, even if libraries or scripts are used and developed concurrently. Another objective is a clear and informative simulation report which includes version information as well as simulation results.

3 Design

The simulation management environment tends to become a very complex system, as various different components are needed to reproduce simulation results. Besides the core model code, parameter lists, measured data, experiment scripts, documentation and other auxiliary files must be stored and recreated. Format, size or number of these auxiliary files is not known in advance. Nonetheless, reliability is major concern for the simulation management environment. However, a substantial ambition in the design process was a straight and simple realization. Therefore, the Concurrent Versions System (CVS) [1] was employed as base layer. CVS uses a client/server architecture, which makes it easy to install and maintain. SiME itself consists only of client side scripts and does not require any additional server software. Therefore SiME can be used with any CVS server. Due to the use of a standard version control system, any file within SiME is accessible using standard tools. Furthermore, concurrent access using SiME and standard CVS clients is seamlessly possible. The following code fragment shows the Matlab call of the CVS client.

```
syscmd=[cvsbin_name,' ',cvsopt_str,' ',...
        cvscmd, ' ',cmdopt_str,' '];
for i = 1:length(cmdargv)
    cmdarg_str = deblank(cmdargv{i});
```

*sven.pohl@dlr.de

†joerg.ungethuem@dlr.de

release	comment	date	time	path	mainscript	results
fk1g0001	a first test run	2004-03-16	13:13:56	.	fk1g_dymola.m	fk1g0001.html
fk1g0002	Comparison of four gas springs to investigate impact of heat transfer and blowby	2004-03-30	10:46:59	.	fk1g_dymola.m	fk1g0002.html
fk1g0003	Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-26	14:31:41	.	fk1g_dymola.m	fk1g0003.html
fk1g0005	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-28	13:19:14	.	fk1g_dymola.m	fk1g0005.html
fk1g0006	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring. (Using new release of plotmaster)	2004-06-01	13:05:22	.	fk1g_dymola.m	fk1g0006.html

Figure 1: Automatically created simulation overview html-page

```

cmdlen=length(syscmd)+length(cmdarg_str);
[rcc,msg]=system([syscmd,cmdarg_str]);
end

```

The CVS command part is defined by `syscmd` while the files to processed are bundled in `cmdargv`.

The core of SiME consists of a few scripts written in Matlab. Dymola's simulation results are easily accessible within Matlab. Matlab's numerical capabilities are outstanding and within SiME all of its features, e.g. the data visualization tools are applicable. However, Matlab is not optimal for string processing and system command execution. Dedicated scripting languages like python and Perl are much more comfortable in this context.

4 Features

The basic concept of SiME is to organize arbitrary simulation tasks in projects. Each project consists of a history of completely reproducible simulation runs. For example, a project named "hybridcar" is a Modelica library development of a hybrid electric vehicle. The developer uses SiME to protocol the developing process and to document the evolution steps. It's not only possible to directly compare the results but also to retrieve the complete simulation code, to make changes if necessary and to re-run the simulation.

The Simulation Management Environment splits each simulation process into four steps. The initialization part sets up the simulation run and calls the CVS routines. Preprocessing, simulation and postprocessing mainly contain code to handle the simulation application (e.g. Dymola) and its results. In a possible fifth

step the complete simulation run can be repeated simply using the simulation ID.

4.1 Initialization

During initialization a unique simulation ID is generated and a complete list of all files which are relevant for the simulation is built. Matlab script dependencies are collected automatically. However, Modelica and auxiliary files must be added manually. SiME forces any file on the list under version control if this was not done before:

```

for element=1:counter
[err,errmsg]= fkcvsadd('',...
cmdopt,notinrepository(counter));
end;

```

Afterwards the files are checked in and tagged using the simulation ID. In that way all files involved in the simulation process are marked with the simulation ID and can be retrieved securely. This code fragment illustrates the process of committing and tagging:

```

for i=1:filenum
% commit files
cmdopt.m = ['automated commit'];
[err.commit(i),errmsg.commit{i,1}]=...
fkcvscommit('',cmdopt, ...
remainder(startpos(i):endpos(i)));
% tag files
[err.tag(i),errmsg.tag{i,1}]=...
fkcvsstag(tag,','', ...
remainder(startpos(i):endpos(i)));
end;

```

4.2 Preprocessing

The initialization routine calls the main simulation script. From this Matlab script the external simulation

General	
project :	fklg
projectrem :	Simulation of components and system of a free piston linear engine
path :	d:\spohl\simulation\project\fklg
mainscript :	fklg_dymola.m
overviewfile :	fklg.txt
overviewfilehtml :	fklg.html
template_parameter :	template_parameter.html
filelist :	1 (1,1) fklg_sim.m 2 (1,2) fklg_dymola.m
externalfiles :	1 (1,1) fklg.mo
date :	2004-07-28
time :	10:32:35
path_current :	D:\spohl\Simulation\dymola
comment :	Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring. Error-plot for pressure added.
tag :	fklg0015
resultpath :	d:\spohl\simulation\project\fklg\fklg0015

Figure 2: Automatically created simulation report html-page

application is started. Generally, any application using the Dynamic Data Exchange (DDE) interface can be called remotely. In this case Dymola is used. In the preprocessing part the Dymola model and the model parameters are defined. For parameter studies an array of values for each parameter can be provided. A parameter matrix is then built from the parameter arrays.

4.3 Simulation

Dynasim provides Matlab functions to start Dymola and execute commands via DDE interface. These functions are used to establish an interaction between Matlab and Dymola. The self explanatory code is shown below:

```
% Set up experiment
ex=dymoexperiment; % default values ex
ex.StopTime = 0.1; % set StopTime

% Start Dymola
res.start = dymostart(sim.dymolabinpath);
% Load Package
res.load = dymoload(sim.package);
% Check Model
res.check = dymocheck(sim.model);
% Translate Model
res.translate = dymotranslate(sim.model);
for num = 1:sim.parmatrixsize
    % Set parameter(s)
    dymosetpar(sim,num);
    % Simulate Model
    res.simulate = dymosimulate(...
```

```
sim.model,ex,sim.modelname);
end;
% Close Dymola
dymoexit;
```

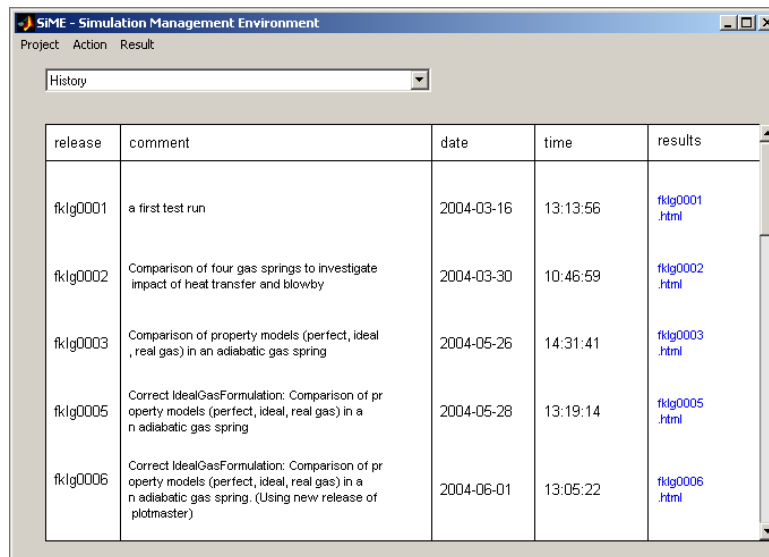
4.4 Postprocessing

Subsequent to the simulation process arbitrary Matlab scripts can be run to further process the results, e.g. to generate plots.

A standardized protocol in HTML is generated including the history of simulation tasks. An example of this simulation overview is shown in Figure 1. For every simulation task a link to a HTML report is given. The HTML report (Figure 2) includes in detail all the parameters used within initialization, preprocessing, simulation process and hyperlinks to the saved plots. Additionally, a summary of all error messages and comments occurred during the runtime is given.

4.5 Re-Run Simulations

To retrieve the complete set of simulation files only the simulation ID is needed. A MATLAB function will retrieve the files from the repository. The files are now ready for manipulation and a new simulation run can be started.



The screenshot shows the SiME - Simulation Management Environment window. It features a menu bar with 'Project', 'Action', and 'Result'. Below the menu is a 'History' dropdown menu. The main area contains a table with the following data:

release	comment	date	time	results
fk1g0001	a first test run	2004-03-16	13:13:56	fk1g0001.html
fk1g0002	Comparison of four gas springs to investigate impact of heat transfer and blowby	2004-03-30	10:46:59	fk1g0002.html
fk1g0003	Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-26	14:31:41	fk1g0003.html
fk1g0005	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-28	13:19:14	fk1g0005.html
fk1g0006	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring. (Using new release of plotmaster)	2004-06-01	13:05:22	fk1g0006.html

Figure 3: The SiME graphical user interface

5 Graphical User Interface

To coordinate the projects and their history a graphical user interface is added. Figure 3 shows the main interface window. Projects can be loaded, created and managed. General information, e.g. directories and project details, can be edited. The simulation history of projects can be browsed and simulation results displayed. New simulation runs can be tested or launched. Dymola result files can be browsed using the Matlab GUI provided by Dynasim.

6 Limitations

SiME inherits the limitations of the CVS system. E.g. the handling of binary files like pictures is not optimal and reordering of directory structures is difficult and error prone.

Most of the CVS limitations are overcome by its successor, the Subversion [2] system. Subversion reached release 1.0 in March 2004. Currently little practical experiences with Subversion are present. However, as Subversion is downwards compatible to CVS, switching to Subversion should be possible.

Absolute directory path references might be included in the model code, like script references in Dymola's annotations. These path references become staled links in the recreated files which cannot be fixed easily. The Matlab–Dymola communication uses Dymola as DDE server which is available on MS Windows only. This prevents the SiME client currently from working on any other operation system.

Some additional Dymola scripting language commands would be desirable, e.g. retrieving a list of all Modelica files which are referenced by a model or generating a screen-shot of the current diagram window.

7 Conclusions

A simulation management environment (SiME) was developed to provide easy and efficient access to earlier simulation runs. SiME ensures the reproducibility of simulation results, even if the models involved are still in development. This facilitates the documentation, avoids redundant work and is an important contribution for quality assurance.

SiME uses Matlab as scripting language, since Matlab is used frequently already for pre- and postprocessing. As backbone server for SiME the Concurrent Versions System (CVS) was selected, since it is freely available and extremely reliable. The use of the models outside of the simulation management is not restricted, so that the normal, CVS supported model development is not disturbed.

References

- [1] Version Management with CVS. Per Cederqvist. <https://www.cvshome.org/docs/manual/>
- [2] Version Control with Subversion. Ben Collins-Sussmann, Brian W. Fitzpatrick, C. Michael Pilato. <http://svnbook.red-bean.com/en/1.1/svnbook.pdf>

Meta-modelling of Mechanical Systems with Transmission Line Joints in Modelica

Alexander Siemers Iakov Nakhimovski Dag Fritzson
 Linköping University, Linköping, Sweden
 SKF, Göteborg, Sweden

Abstract

A framework for meta-modelling with *Transmission Line* (TLM) joints is presented. The framework is intended to support transient simulations of mechanical systems using co-simulation of different tools. The expressive power of the Modelica language is used to describe the meta-model in an easy to understand, object oriented way. A ModelicaXML based translator is used to convert Modelica code to an XML document which is accepted as input by the co-simulation engine. The framework prototype for SKF's BEAST and MSC.ADAMS is presented here. It is designed to be general, so that support for other simulation tools can be easily added. The main focus is on modelling of co-simulation Meta-Models taking advantage of Modelica's graphical and object-oriented modelling capabilities.

Keywords: simulation; co-simulation; meta-modelling; multibody; TLM; XML

1 Motivation

In the area of modelling and simulation of mechanical systems one can identify many different classes of models and corresponding tools. The specialization leads to different focus for different tools. One might say that every tool is optimized for a certain kind of problems. In terms of meta-modelling every tool can be seen as a black-box handling a particular *component*. A component is a model defined in some specific language together with some modelling and simulation tool that can perform a transient simulation of it. The examples of such components are equation-based multi-physics Modelica models, general multibody models in MSC.ADAMS, models with detailed contact definitions in SKF's BEAST, flexible components as modelled in FE tools, etc. .

In reality the different components are dependent on each other. Two components that are in physical in-

teraction form boundary conditions for each other and some interface can often be defined.

Unfortunately it is often the case that the different classes of tools are used independently. Every class of tools is using approximations of the components it has interface with, that is, simplified models of the boundary conditions. Several time consuming iterations are often necessary to make the components converge to similar values on the common interface. The limitations on the modelling accuracy are thus fundamental.

The need to bring different components into a complete more tightly coupled simulation is therefore justified. This allows higher accuracy and preserve the investments in the components.

Different co-simulation systems have appeared on the market during the last years. Most of them are focused on co-simulation of control systems and corresponding mechanical component. The coupled simulations this paper is focusing on are different. All components in our framework are mechanical and they have forces and motion in the interfaces. What is more important from numerical point of view, the sub-models are likely to use different differential equation solvers with variable time step. Numerical stability, which is not an issue for discrete time simulations, becomes an important consideration.

One method that was earlier used to enable closer interaction between such sub-models in a coupled simulation is *transmission lines modelling* (TLM). The TLM uses physically motivated time delays to separate the components in time and enable efficient co-simulation. The technique has proven to be stable and was implemented for coupling of hydraulical and mechanical sub-systems [1], [2].

However, no attempt to design a general coupled simulations framework was done. In this paper a general approach to meta-modelling of mechanical systems using TLM is presented. Modelica language is used to make such models easy to manage and the frame-

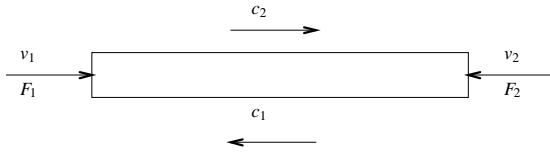


Figure 1: Delay line with the passing wave variables c_1 and c_2 and velocity variables v_1 and v_2 .

work is designed to enable simple extension with new simulation tools.

2 Transmission Line Modelling

TLM method, also called Bilateral Delay Line Method, exploits the fact that all physical interactions in nature have finite propagation speed. The properties of the delay lines were studied in [7]. The method is briefly described below.

A basic one-dimensional transmission line is shown in Figure 1. For the mechanical case the line is basically a long spring with force waves c_1 and c_2 going between it ends. The input disturbances are velocities v_1 and v_2 and the reaction forces from the transmission line F_1 and F_2 .

Note that the spring in our implementation is assumed to be iso-elastic. That is no cross-term waves are generated when working in 2D and 3D. See [2] for further discussions.

If the line delay is set to T and its impedance to Zc then the governing equations are:

$$\begin{aligned} c_1(t) &= F_2(t - T) + Zc v_2(t - T) \\ c_2(t) &= F_1(t - T) + Zc v_1(t - T) \\ F_1(t) &= Zc v_1(t) + c_1(t) \\ F_2(t) &= Zc v_2(t) + c_2(t) \end{aligned} \quad (1)$$

The equations show that the two simulation systems are decoupled with the delay time T . Simulation framework can utilize this decoupling to enable efficient communications during co-simulation.

The transmission line introduces a parasitic mass $m_{tlm} = Zc T$ and stiffness $k_{tlm} = Zc/T$. Since it is often necessary to have a relatively large delay time (to enable larger communication intervals) while keeping the stiffness value, the user needs to be aware of the large parasitic mass.

3 Simulation Framework

The design goals for the simulation part of the framework were portability, simplicity to incorporate new simulation tools, computational efficiency. The design goals were realized by defining following concepts and interfaces:

TLM interface. A named point on a mechanical object where position and velocity can be evaluated and reaction force applied.

TLM manager. The central simulation engine. It is a stand alone program that reads in a XML definition of the coupled simulation. It then starts *Simulation components* and provides the communication bridge between the running simulations. That is the components only communicate with the TLM manager which acts as a broker marshalling information between the components as required by TLM theory. TLM manager sees every simulation component as a black box having one or several TLM interfaces. The information is then forwarded between TLM interfaces belonging different components.

TLM plug-in. A small C++ library having a single abstract class representing TLM interface for a specific simulation tool. The TLM plug-in can be seen by a simulation component as an external force that depends on position, velocity and time. The implementation of the plug-in handles the necessary communications with TLM manager.

Simulation component. Any simulation program that has incorporated TLM plug-in as a part of its model. A small script that takes the general parameters as input and starts the specific component is an additional requirement. This intermediate step is necessary since TLM manager needs a common way to start all the components and each tool might have some specific start procedures.

4 Modelica as Meta-Model Language

Simulations of complete systems where components are modeled and simulated in different simulation packages are called co-simulations. The model of a co-simulation including all system components and its inter connections we call a meta-model.

The extended markup language (XML) has its strength in textual data representation and conversion. It is often the language of choice for communicating information between different tools. Those were the reasons behind the decision to use it as the input to the simulation engine.

Readability and edit-ability, on the other hand, are not the strengths of XML. Design of co-simulation meta-models requires thus a more powerful modelling language or graphical modelling environment. The following requirements were defined for meta-model definitions:

- Meta-Models should be based on a standard language.
- A graphical model editor should be available for ease of use.

Modelica with its object oriented modelling capabilities and its standardized graphical notations is thus perfectly suited. The fact that the Modelica standard defines graphical notations results in the availability of graphical model editors, i.e., MathModelica [5] and Dymola [4]. These editors typically allow easy connection modelling and user interface driven class design.

It should be mentioned that only Modelica's modelling capabilities are of interest here. Meaning that there is no need for Modelica based simulations. The use of Modelica as meta-modelling language might as well simplify the integration of Modelica simulations into meta-model based co-simulations. This, however, is not within the scope of this work.

4.1 Meta-Model Class Library

A meta-model Modelica package for component and TLM connection modelling, using Modelica's object oriented features, has been designed.

Three packages plus a base model class were defined:

The Components package contains classes for the different simulation components. These are currently BEAST and MSC.Adams components.

The Connections package contains the TLM connection or joint. Different TLM specific parameters can be specified for each connection.

The Interfaces package contains the corresponding TLM interface. Each TLM component contains at least one TLM interface.

The BaseMetaModel class is the base class for each Meta-Model. It contains Meta-Model specific parameters.

Different TLM components are defined in the components package which are inherited from the simulation tool specific components, see also Figure 2.

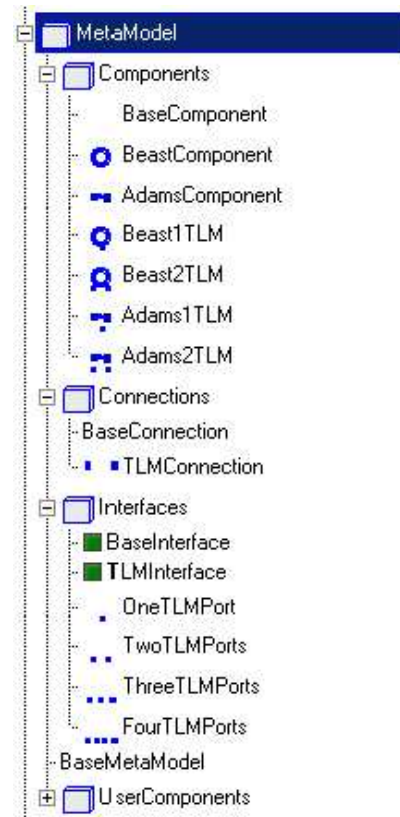


Figure 2: The basic TLM Meta-Model class library.

They add a certain number of TLM interfaces to each component. TLM connections define data exchange and synchronization between these components during co-simulation. Connections are created between two TLM interfaces of two TLM components. TLM interfaces are therefore defined as connectors. Several base classes define common model parameters needed by the TLM manager or for correct XML translation. Specialized child classes modify these parameters to their needs. BEAST Components for example modify the start-method as follows:

```

model BaseComponent
  parameter String Description;
  parameter String SimulationFiles;
  parameter String StartMethod;
  .
  .
end BaseComponent;

model BeastComponent
  extends BaseComponent
  (StartMethod="beast --serial");
  .
  .
end BeastComponent;

```

Both component and interface classes contain a type specifier which is TLM for TLM components and TLM

interfaces. This allows for additional type checking during model translation and guarantees that TLM interfaces are connected with TLM connections. But is also useful for future extensions with new connection types.

4.2 Component Modelling

Component modelling is divided into two steps:

- Component modelling in the specialized environment. Each component of the multi-scale simulation is modeled in its specific environment. Users define the TLM interfaces to the model.
- Component modelling in the multi-scale environment. The component needs to be integrated into the multi-scale environment. Startup methods, interfaces, and communication parameters must be specified.

Co-simulation components are modelled in the modelling environment of the specific simulation tool. To participate in a TLM co-simulation each simulation program needs to integrate an TLM plug-in and a way to model TLM interfaces. In MSC.ADAMS for example external forces are connected to a TLM interface, and BEAST defines so called TLM-ties. The TLM interfaces are thus part of the simulation model expressed in the modelling language of the specific program.

Simulation components are integrated into the meta-model by selecting a matching component from the Modelica Meta-model library. Component type and number of TLM interfaces have to match.

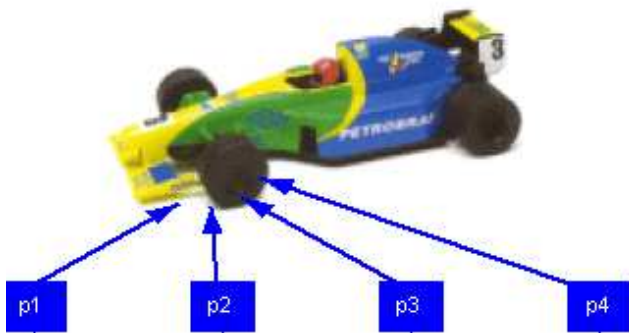


Figure 3: A MSC.ADAMS car simulation component in the Modelica environment with four TLM interfaces at the front tire.

Alternatively can the base components, i.e., *BeastComponent* and *AdamsComponent*, be extended (inherited) and a certain number of TLM interfaces be

added. This allows for other extensions as well, e.g., selecting appropriate component icons for more intuitive modelling, see Figure 3. New components should be added to the *UserComponents* package in the Meta-Model library. This is needed for the XML translator to work properly.

4.3 Meta Modelling

Meta-Models are created using a graphical Modelica editor, see Figure 4, where components are dragged into the model. Every Meta-Model must extend the *BaseMetaModel* class that contains Meta-Model and co-simulation specific parameters. TLM components and connections are added to the model and connections are drawn between the TLM interfaces.

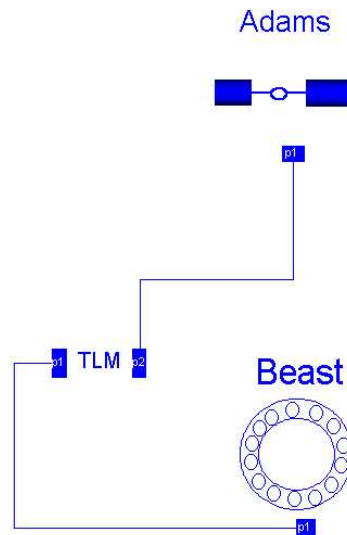


Figure 4: A simple BEAST-MSC.ADAMS Meta-Model.

Several parameters need to be specified for the different parts of the model. They are needed by the TLM manager for correct simulation execution. BEAST and MSC.ADAMS components, for example, need a simulation file to be specified, see Figure 5, and TLM connections require correct TLM parameters.

The meta-model description is kept general and works with any simulation tool that supports TLM connections. New components can be created by extending the *BaseComponent* class or any of the predefined component classes. Only the start-method for the simulation tool needs to be specified for new components. Predefined components can be extended if more TLM interfaces are required. The number of required TLM interfaces is application and simulation-model dependent.

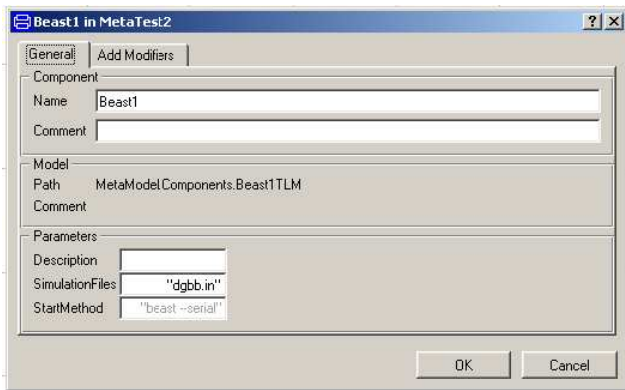


Figure 5: The BEAST component parameter dialog.

4.4 Meta-Model Translation

The meta-model is translated into XML code to run in the co-simulation framework. A Modelica to XML translator has been designed for this purpose. The translator makes use of ModelicaXML [3] plus some co-simulation specific translations. The translation is done in two steps:

1. Translation from Modelica to ModelicaXML
2. Translation from ModelicaXML to the Meta-Model XML representation

To simplify parsing of the Modelica Meta-Model it is first translated into a Modelica-XML representation using the ModelicaXML [3] translator. This representation can be parsed and translated with a standard XML-parser. The libXML2 [6] standard library has been used to convert the ModelicaXML Meta-Model into the XML representation required by the TLM manager.

4.5 Meta-Model Example

An typical example of a BEAST-MSC.ADAMS Meta-Model is shown in Figure 6. A front wheel bearing hub-unit is connected to the race-car with four flanges each of which is modelled as a TLM connection. The components have to be prepared in BEAST and MSC.ADAMS to contain the TLM interfaces. Afterwards they are integrated into the meta-model environment by creating component classes with appropriate icons and TLM interfaces in the Modelica package. Each Meta-Model needs to extend the *BaseMetaModel* Modelica class to inherit the global co-simulation parameters. TLM connections are added between the TLM interfaces according to the hub-unit flanges. The complete Modelica model looks like this:

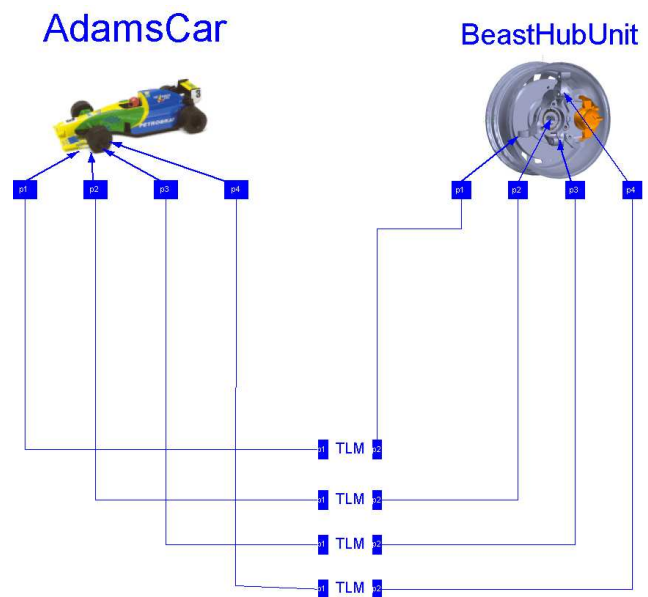


Figure 6: Modelica Meta-Model detailed BEAST hub unit integrated into a MSC.ADAMS racing-car model.

```

model BeastHubInAdamsCar
  extends MetaModel.BaseMetaModel;
  MetaModel.UserComponents.BeastCarCorner
    BeastHubUnit(
      Description=
        "A complete Beast hub-unit",
      SimulationFiles="CarCorner.in",
      StartMethod="start-beast");
  MetaModel.UserComponents.AdamsCarModel
    AdamsCar(
      Description="A MSC.ADAMS car model",
      SimulationFiles="racing_car.cmd");
  MetaModel.Connections.TLMConnection TLM1;
  MetaModel.Connections.TLMConnection TLM2;
  MetaModel.Connections.TLMConnection TLM3;
  MetaModel.Connections.TLMConnection TLM4;
equation
  connect(AdamsCar.p1,
    TLMConnection1.p1);
  connect(AdamsCar.p2,
    TLMConnection2.p1);
  connect(AdamsCar.p3,
    TLMConnection3.p1);
  connect(TLMConnection1.p2,
    BeastHubUnit.p1);
  connect(TLMConnection2.p2,
    BeastHubUnit.p2);
  connect(TLMConnection3.p2,
    BeastHubUnit.p3);
  connect(TLMConnection4.p2,
    BeastHubUnit.p4);
  connect(AdamsCar.p4,
    TLMConnection4.p1);
end BeastHubInAdamsCar;
    
```

Finally the model is converted into the XML representation required by the TLM manager by first converting it into ModelicaXML and then into the XML Meta-Model representation. The XML model looks like this:

```
<?xml version="1.0"?>
<Model name="BeastHubInAdamsCar"
  StartTime="0"
  EndTime="1"
  TLMDelay="0.001">
  <SubModels>
    <SubModel Name="BeastHubUnit"
      Description=
        "A complete Beast hub-unit"
      SimulationFiles="CarCorner.in"
      StartMethod="start-beast">
      <InterfacePoint Name="p1"
        iType="TLM"/>
      <InterfacePoint Name="p2"
        iType="TLM"/>
      <InterfacePoint Name="p3"
        iType="TLM"/>
      <InterfacePoint Name="p4"
        iType="TLM"/>
    </SubModel>
    <SubModel Name="AdamsCar"
      Description=
        "A MSC.ADAMS car model"
      SimulationFiles="racing_car.cmd"
      StartMethod="start-adams">
      <InterfacePoint Name="p1"
        iType="TLM"/>
      <InterfacePoint Name="p2"
        iType="TLM"/>
      <InterfacePoint Name="p3"
        iType="TLM"/>
      <InterfacePoint Name="p4"
        iType="TLM"/>
    </SubModel>
  </SubModels>

  <Connections>
    <Connection From="AdamsCar.p1"
      To="BeastHubUnit.p1"
      iType="TLM" alpha="0" zf="0"/>
    <Connection From="AdamsCar.p2"
      To="BeastHubUnit.p2"
      iType="TLM" alpha="0" zf="0"/>
    <Connection From="AdamsCar.p3"
      To="BeastHubUnit.p3"
      iType="TLM" alpha="0" zf="0"/>
    <Connection From="BeastHubUnit.p4"
      To="AdamsCar.p4"
      iType="TLM" alpha="0" zf="0"/>
  </Connections>
</Model>
```

5 Conclusion

A framework for meta-modelling and simulation of mechanical systems using transmission lines for coupling components was presented. The main features of the framework are:

- General object-oriented meta-modelling utilizing the strengths of Modelica
- Stability by applying minimalist approach to the program design resulting in small portable code
- Extensibility of the framework thanks to the portable and easy to incorporate software plug-in.

The framework currently targets SKF's BEAST simulation tool and MSC.ADAMS.

References

- [1] Krus P., Jansson A. Distributed Simulation of Hydromechanical Systems 'Third Bath International Fluid Power Workshop', Bath, UK 1990
- [2] Krus P. Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints. Transactions of ASME, Journal of Dynamic Systems Measurement and Control. Dec 1999
- [3] Pop A., Fritzson P. ModelicaXML: A Modelica XML Representation with Applications, Modelica 2003 Conference
- [4] Dymola, <http://www.dymola.com>, Dynasim AB
- [5] MathModelica, <http://www.mathcore.com/products/mathmodelica> Mathcore AB
- [6] The XML C parser and toolkit of Gnome, <http://www.xmlsoft.org/>
- [7] Larsson, J. Interoperability in Modelling and Simulation, PhD thesis, Linköping University, Linköping, Sweden, 2003

Session 3a

Automotive Simulation I

AirConditioning – a Modelica Library for Dynamic Simulation of AC Systems

Hubertus Tummescheit[†] Jonas Eborn[†] Katrin Pröbß[‡]

[†] Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden

[‡] TU Hamburg–Harburg, Department of Thermodynamics, Denickestr. 17, D-21073 Hamburg, Germany

Abstract

The AirConditioning library is a new, commercial Modelica library for the steady-state and transient simulation of air conditioning systems using modern, compact heat exchangers that use microchannel tubes instead of the bulkier fin-and-tube type heat exchangers. Currently it is mostly used by automotive OEMs and suppliers that need high-accuracy system level models to ensure both passenger comfort and energy efficiency of systems developed under the pressure of reduced design cycle times. The AirConditioning library contains basic correlations for heat and mass transfer and pressure drop, components for control volumes and flow resistances and advanced ready-to-use models for all relevant components of automotive air conditioning systems like condenser, evaporator, compressor, expansion devices and accumulator.

1 Introduction

The AirConditioning library has been derived from the Modelica library ThermoFluid [1, 2] and the ACLib library [11], with considerable enhancements in particular of the useability and robustness. Most of the fundamental design ideas outlined in [1, 2] are still valid, but a number of useability-oriented design improvements have been made also with respect to the specializations for AC-cycles described in [11]. Compared to ThermoFluid, also simplifications of the library structure have been made due to the reduced spectrum of applications. The most important differences are:

Steady-state capabilities Traditionally, AC system level models are only used as steady-state models, with the exception of very simplistic, often linear models for control design. ThermoFluid provided accurate dynamic models, but could not be used for steady-state tasks. AirConditioning bridges that gap and is suited both for

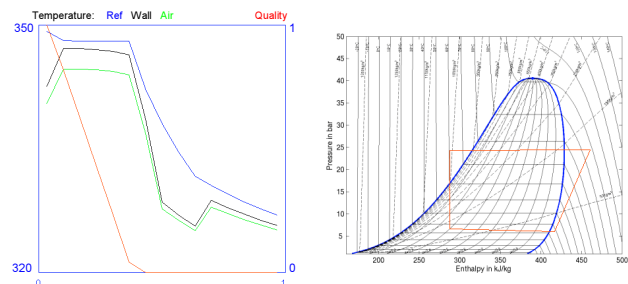


Figure 1: Examples of dynamic diagrams: spatial plot of condenser temperatures and ph-diagram for a R134a cycle.

dynamic and steady-state design computations, eliminating the need for multiple platforms and models. Significant improvements to the steady-state solvers in Dymola combined with model improvements have resulted in reliable steady-state initializations that can be used for design optimization.

Re-designed user interface The user interface improvements make full use of recent Dymola features: structured dialogs using hierarchy, tabs and groups where appropriate, illustrations linked into the dialogs for explanation of the parameter meaning and use of interactive elements for direct user feedback during simulation runs.

Dynamic process diagrams The UserInteraction library by Dynasim has been used to create dynamic interface elements for AC applications: spatial plots of temperatures or heat transfer coefficients and instantaneous corner points of the refrigerant cycle ph-diagram, as shown in Figure 1.

New two-phase dynamic state model The *integrated mean-density model* has been introduced for two-phase flow and greatly reduces the risk of discretization-triggered flow oscillations.

Apart from the robustness benefit it is also a reduced-order dynamic model that doesn't sacrifice accuracy, but rather allows the same or better accuracy with fewer dynamic states.

Many AC component models A number of new, specialized AC-component models have been added, e.g. internal heat exchanger, condenser with integrated receiver, short orifice tube, and many more.

Compressor models Two formulations for the compressor efficiencies have been developed: one is for the case of full-load measurement data only, reported in [4], the other computes also efficiencies for varying swash-plate angle inputs [3].

Optional model encryption Dynasim has developed a novel approach to model encryption that makes use of symbolic pre-processing of the model code before the actual encryption, called "scrambling". Most critical data is irretrievably removed from the model code even before proper encryption by evaluating all given parameters. The new method allows to selectively hide or reveal model features, giving the user full control over available model parameters and outputs. The symbolic evaluation of parameter expressions before code generation masks geometry information in a way that it is impossible to retrieve it even from the generated C-code.

1.1 Standard for model exchange

Dymola and the AirConditioning library was chosen by a group of German OEMs after a benchmark comparing it with other potential tools. During 2004, the tool was tested by the OEMs [9] and many of their suppliers, and then chosen as a common tool for model exchange between suppliers and OEMs. The benchmark and testing process has contributed to continued improvements of the library regarding the component-oriented requirements from suppliers and the system-oriented requirements of OEMs.

2 Heat exchanger models

In automotive refrigeration cycles heat is absorbed at the low temperature level of the cabin air or at ambient temperature and rejected at the discharge level of the ambient. For heat transfer between air and working fluid a condenser/gas cooler on the high pressure

level and an evaporator on the low pressure level are used, exploiting the low heat transfer resistance of the two-phase refrigerant. In some, mainly R744, applications, an internal refrigerant-to-refrigerant heat exchanger which transfers heat from one pressure level to the other enhances the performance of the cycle. Most heat exchanger types currently used in automotive air conditioning systems are represented by the library models or they can be developed from subcomponents.

2.1 Refrigerant side models

The fluid flow on the refrigerant side is based on dynamic control volume models that are different than the standard finite volume model found in ThermoFluid [1, 2]. The AirConditioning library uses from version 1.1 a new control volume that is similar to the one used in the ThermoPower library [5]. The main difference is that it is based on the computation of the *mean density*, $\bar{\rho}$, found by integrating over enthalpy along the flow, assuming constant pressure and taking into account the location of the phase boundaries (h_{pb}),

$$\bar{\rho} = \int_{h_1}^{h_2} \rho(p, h) dh = \int_{h_1}^{h_{pb}} \rho(h) dh + \int_{h_{pb}}^{h_2} \rho(h) dh \quad (1)$$

With different inlet and outlet conditions and over the two boundaries, h_{liq} and h_{vap} , the integral splits up into 9 different cases, for which the analytic solution can be derived. In the one-phase region a regular mean value is used. Within the two-phase region the integral is rewritten using the expressions for quality x and volumity, $v = 1/\rho$, which are linear in enthalpy.

$$x = \frac{h - h_{liq}}{h_{vap} - h_{liq}} \quad v = x \cdot v_{vap} + (1 - x)v_{liq}$$

$$\int_{h_1}^{h_2} \rho(h) dh = \frac{h_{vap} - h_{liq}}{v_{vap} - v_{liq}} \int_{v_1}^{v_2} \frac{1}{v} dv = \frac{h_{vap} - h_{liq}}{v_{vap} - v_{liq}} \ln \frac{v_2}{v_1}$$

The expressions are such that they are continuously differentiable even across the phase boundaries. The analytic derivatives of the mean density w.r.t. the inputs of the fluid property calculation have also been derived and validated using the new automatic differentiation feature of Dymola [6].

Due to the magnitudes of temperature gradients and pressure drops, a different parameterization than chosen by [5] has to be implemented for air conditioning systems: pressure drops have a larger influence on the driving temperature difference and can not be neglected. Another important feature of the refrigerant side models is to fully make use of the fact that

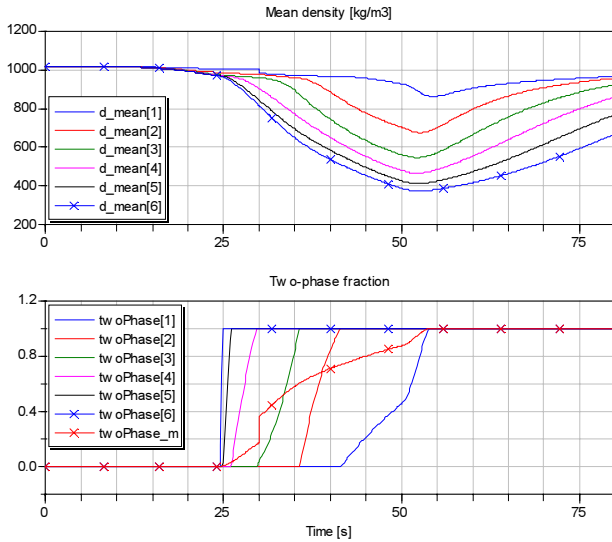


Figure 2: Mean density and two-phase fraction in a six segment pipe with R134a during a rapid transient starting in all liquid phase. Both properties are smooth throughout the simulation, with the exception of $d_{mean}[1]$ and the overall two-phase fraction, $twoPhase_m$, that jump when the inlet enthalpy changes at $t=30s$.

the phase boundary location is resolved continuously within each finite volume and not just discretely for each volume. Using the two-phase length fraction for interpolation of all phase dependent correlations and properties improves calculation accuracy vastly. An added benefit is that the interpolation also makes variables such as heat transfer coefficient change continuously with time when the phase boundary moves from one finite volume to the next.

The smooth results of the mean density model is illustrated in Figure 2 where the calculated mean density and two-phase fraction of a refrigerant pipe is shown. Heat transfer properties are interpolated with the individual two-phase fraction of each volume, while the pressure loss model can use the overall two-phase fraction of the pipe. The pipe model with $n = 6$ will then only have one dynamic pressure state but six enthalpy states. This model is normally used for the refrigerant side of a heat exchanger, where a six pass evaporator with $n = 3$ will have six pressure states and eighteen enthalpy states.

2.2 Air side models

Air side models in compact heat exchangers of air conditioning systems are characterized by three features:

- sharp gradients along short flow paths,

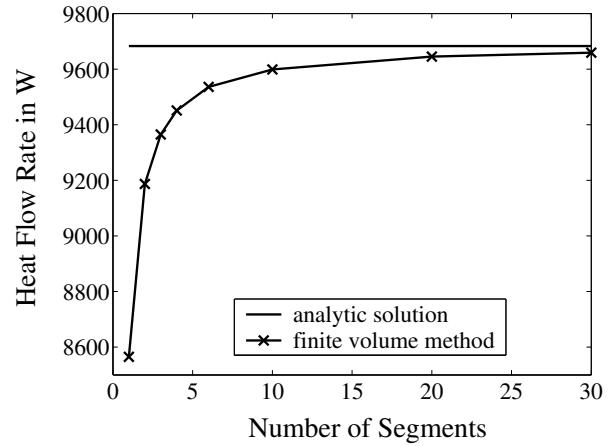


Figure 3: Influence of air side discretization on total transferred heat in a refrigerant condenser

- discontinuous phenomena depending on condensing/ non-condensing conditions on cold surfaces and
- very short residence times of an air particle inside the heat exchanger.

To accommodate for these features, two different models have been developed for the air side of compact automotive heat exchangers. One is a discretized finite volume based model with steady-state mass- and energy balances for each volume, the other is a symbolic solution of the outflow temperature found by applying constant medium properties along the flow path. In both cases the correlation for convective heat transfer is used,

$$\dot{Q} = \alpha A (T_{wall} - T)$$

where \dot{Q} is the heat flow rate, α is the coefficient of heat transfer, A is the surface area, T and T_{wall} are the temperature in the bulk flow and at the wall surface, respectively. The heat connector variables \dot{Q} and T_{wall} provide an interface to wall models. The computational burden of dynamic balances with an increased number of dynamic states is avoided by using steady-state balances, which is justified by the short residence time of the fluid.

Due to the sharp gradients and/or discontinuities on the air side, the finite volume method requires a relatively high discretization. If high accuracy is required, typically 10 – 15 elements are needed for the air passage¹ Figure 3 shows simulation results for the steady-state heat flow rate of a compact flat tube condenser of 2

¹For multi-layered heat exchangers this includes the sum of elements for all layers, (air segments/layer)*(number of layers).

cm depth comparing the finite volume and the analytical approach. Air temperatures at inlet and outlet were 320 K and around 336 K, respectively.

A symbolic solution for the outlet state can only be found if the water content of the air remains constant along the flow path, which is only the case for very low or zero inlet humidities or air heating. In the finite volume model, when the wall surface temperature drops below the saturation temperature of the bulk flow, the amount of condensing humidity will be determined by applying a heat and mass transfer analogy approach. Assuming a similarity in the shape of the temperature boundary layer of a convective fluid flow and that of the respective concentration boundary layer, the mass transfer coefficient β can be determined from

$$\beta = \frac{Le^{(m-1)}\alpha}{\rho c_p}$$

where Le is the Lewis number, α the coefficient of heat transfer and with $m = 1/3$ valid for most applications [12]. The driving potential of water condensation is then formed by the water content in the bulk flow X and that for saturation at surface temperature $X_{sat}(T_{wall})$. Assuming the ideal gas law applies, the condensate flow rate \dot{m}_w is computed from

$$\dot{m}_w = \beta \rho A (X - X_{sat}(T_{wall}))$$

with ρ as the bulk flow density. The model allows outlet humidities below 100% and water condensation at the same time. The correct determination of the latent heat is important, as it can make up around 50% of the total transferred heat.

Heat transfer and pressure drop correlations for air side specific geometries from the literature are part of the library. Additional user correlations can be incorporated on the component top level by using replaceable classes.

2.3 Air-refrigerant heat exchangers

Condensers/gas coolers and evaporators in automotive refrigeration cycles are mostly of cross, cross-co or cross-counter flow type and consist of louvered fins and extruded microchannel flat tubes, both made of aluminium, as schematically shown in Figure 4a). The models in the library are composed of refrigerant and air cross flow elements with walls between the two media [11]. Heat conduction in the solid material in fluid flow direction is neglected. The dynamic behavior of the component is mainly influenced by the amount and distribution of the solid wall material and

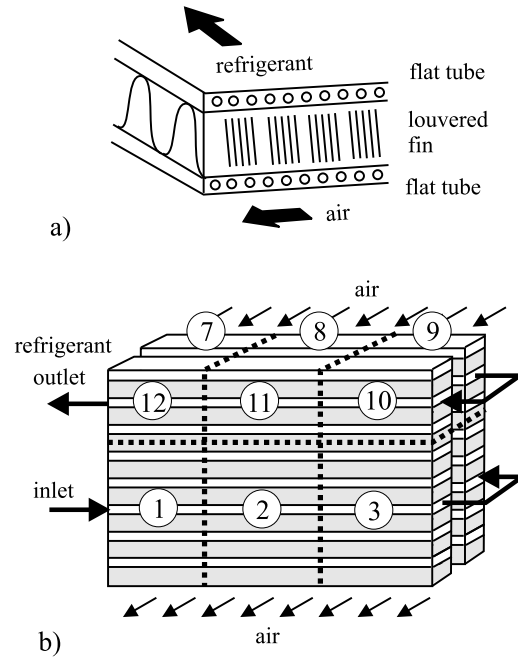


Figure 4: a) cross flow of air and refrigerant, b) 4-pass condenser with horizontal refrigerant flow and a refrigerant side discretization of 3 per pass

associated heat capacity. On both sides of the wall, several parallel flow channels are lumped into one homogeneous flow for efficiency reasons. The refrigerant path through the component is treated as one pipe flow with variable cross section and one air element associated with each flow segment. Each air element is further discretized or symbolically integrated along its flow. Automatic coupling of air elements is made according to the parameter-specified and component type dependent 3D orientation, e.g. as the evaporator shown in Figure 6 and to the user defined segmentation of the refrigerant flow. Both parameters are merged into a 3D-matrix, which defines the position of each refrigerant segment with respect to a fixed coordinate system. The condenser in Figure 4b) would yield a 2 by 2 by 3 matrix which is used for conditional connect statements of air inlets and outlets in the component. This approach allows for a wide variety of flow paths and a 2D-interface for inhomogeneous air inlet. However, the interface resolution is directly coupled to the number of refrigerant passes through the heat exchanger and their segmentation.

2.4 Internal heat exchangers

For systems using the refrigerant R744 (CO₂) as the cycle fluid, it is quite common to have an internal heat exchanger between the high pressure side, after the

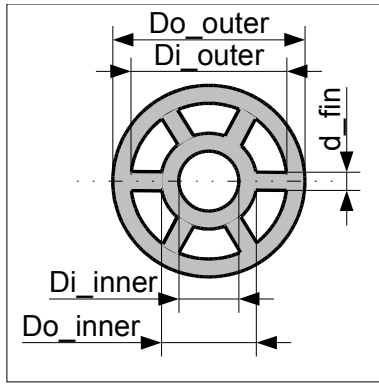


Figure 5: Cross section of tube-in-tube internal heat exchanger.

gascooler, and the low pressure side, between the accumulator and the compressor. The base classes for the internal heat exchanger are identical to those for the refrigerant side of flat tube heat exchangers.

Currently used internal heat exchangers come in a wide variety of geometries. Tube-in-tube type internal heat exchangers as in Figure 5 can be parametrized directly from the geometrical data. For other types of internal heat exchangers, the user has to compute parameters like the hydraulic diameter and the heat transfer areas by hand.

3 Swash plate compressor model

The compressor is modeled as a steady-state map that relates suction- and discharge states and mass flow. Due to the wide variety of mechanical constructions, a simple parameterization of a swash-plate or swashing compressor has to be based on an extensive set of measurements. The measurements are used to adapt the free parameters of efficiency functions that are chosen to have physically reasonable asymptotics for high pressure ratios and low rotational speed. The form of the functions is similar to the one presented in [4], and varies slightly for different compressor types. The compressor model uses three functions to characterize the compressor efficiencies, the volumetric efficiency λ_{eff} , the effective isentropic efficiency η_{eff} and the isentropic efficiency η_{is} . The efficiencies are defined as

$$\begin{aligned}\lambda_{eff} &= \frac{\dot{m}_{eff}}{Vn\rho(p_s, T_s)} \\ \eta_{eff} &= \frac{P_{is}}{P_{eff}} = \frac{(h_{d,is} - h_s)\dot{m}_{eff}}{2\pi|M|n} \\ \eta_{is} &= \frac{h_{d,is} - h_s}{h_d - h_s}\end{aligned}$$

In the definitions above, p is the pressure, T temperature, V displacement volume, ρ density, h specific enthalpy, P power, \dot{m} mass flow and M the torque of the compressor. In the subscripts d refers to the discharge side, s to the suction side, is to isentropic conditions and eff to effective values.

In order to simplify the situation in early development stages, the efficiency functions are factored into two parts: one that captures the influence of the pressure ratio and rotational speed, $f(\pi, n)$ and another one that takes into account the control of the swash plate angle and rotational speed, $g(x, n)$. Measurements of the influence of the swash plate angle are not always available, and due to this separation it is still possible to derive efficiencies for the full load case. A typical form of the efficiency functions is given below.

$$\lambda_{eff} = \left(\pi_0 - \frac{p_d/p_s}{\pi_0 - 1} \right)^2 \left(\frac{x - x_0}{1 - x_0} \right) (a_2 n^2 x + a_1 n x + a_0)$$

$$\begin{aligned}\eta_{is} &= f(\pi, n) \cdot g(x, n) \\ f(\pi, n) &= a \frac{\pi_0 - \pi}{\pi_0} - ab \left(\frac{1}{b} \frac{\pi_0 - 1}{\pi_0} \right)^\pi \\ g(x, n) &= \left(1 - \left(\frac{x - 1}{x_0 - 1} \right)^k (c + 1 - b)x \right) \\ a &= a(n) = a_1 n + a_0 \\ b &= b(n) = b_3 n^3 + b_2 n^2 + b_1 n + b_0 \\ c &= c(n) = c_1 n + c_0 \\ k &= k(n) = k_1 n + k_0\end{aligned}$$

The effective isentropic efficiency has the same functional form as the isentropic efficiency. Two of the constants have physical significance, π_0 is the upper limit of the pressure ratio at which the discharge mass flow decreases to 0. Similarly, x_0 is the lower limit of the relative displacement control signal where the compressor does not discharge any more. The parameters a_i , b_i , c_i and k_i are free parameters that have to be adapted to measurement data.

4 Expansion devices and valves

The library includes simple orifice and thermostatic expansion valve (TXV) models. Several models of these short flow restrictions are based on the computation of mass flow of compressible fluids as described in DIN EN 60543-2-1, computing a flow coefficient K_v in m^3/h . The model takes into account the choking of flow above the critical pressure ratio. Simpler models

with a constant ζ -parameter,

$$\Delta p = \frac{|\dot{m}| \dot{m} \zeta}{2A^2 \rho}$$

and with quadratic scaling based on nominal parameters are also available. The TXV is based on the DIN valve model, with a PI-controller with a suitable time constant representing the bulb dynamics.

4.1 Short orifice tube

A geometrical model of a short orifice tube is also included according to the correlation in [7]. The orifice tube model has been validated against measured data from the reference with good results over a wide range of operating conditions. The mass flow error is less than 5-10% in all but extreme cases. For sub-cooled conditions the liquid flow equation 2 is used, and for fully choked flow equations 3-4 are used. Inbetween these extremes the mass flow is interpolated based on upstream quality.

$$\dot{m}_l = C_1 D_{tube}^2 \sqrt{2\rho_l(p_1 - p_f)} \quad (2)$$

$$\dot{m}_c = \frac{\pi}{4} p_1 D_{tube}^2 \sqrt{\frac{M^2 \kappa}{RT}} \quad (3)$$

$$\lambda \frac{L_{tube}}{D_{tube}} = \frac{1 - M^2}{\kappa M^2} + \frac{\kappa + 1}{2\kappa} \log \frac{M^2(\kappa + 1)}{2(1 + \frac{\kappa - 1}{2M^2})} \quad (4)$$

In the equations above λ is the friction coefficient, κ is the ratio of specific heats and p_1 is upstream pressure. The adjusted downstream pressure, p_f , depends on subcooling temperature, critical pressure and tube dimensions [8]. Note that Equation 4 is an implicit equation for the Mach number M . It is used exactly as quoted in the orifice tube model.

5 User interface

The library makes full use of recent Dymola features to make the models easy to use. Component parameter dialogs are structured using tabs and grouping, with appropriate text and graphical explanations. All non-numerical input values can be selected from drop-down menus and the lists of choices for correlation models and geometry records are automatically updated using the annotation *choicesAllMatching*. As an example, the geometry parameter dialog for a flat tube evaporator is shown in Figure 6.

To further enable an easy understanding of simulation results, dynamic diagrams have been integrated into

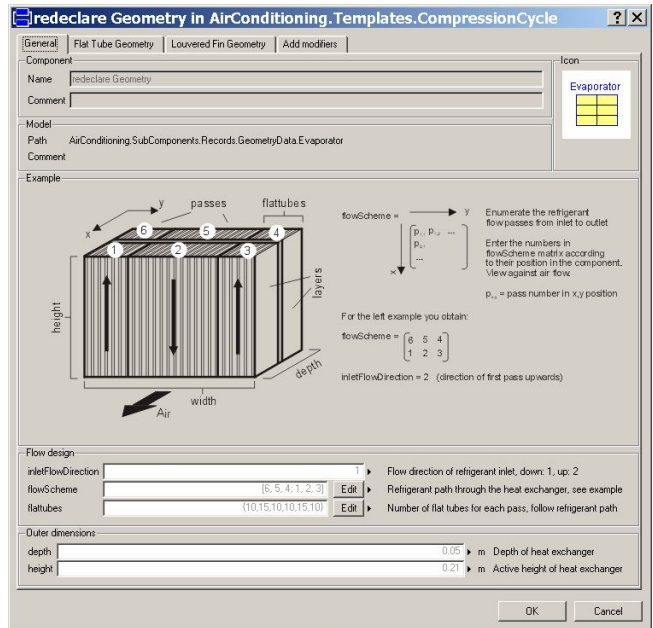


Figure 6: Parameter dialog for specifying evaporator geometry parameters. Illustrations and explanations provide help for the input fields and different parameters are grouped under tabs; General, Flat tube geometry and Louvered fin geometry.

example models using the library UserInteraction. Dynamic components include value displays showing e.g. instantaneous temperature and transferred heat, spatial plots showing temperatures, quality or other properties along the refrigerant flow direction and ph-diagrams that illustrate the full refrigerant cycle behavior. Examples of dynamic diagrams are shown in Figure 1.

6 Initialization

Robust steady-state initialization is critical for using dynamic AC models also for steady-state applications and system design optimization. From a tool and library implementation viewpoint all of the pieces below are important to allow robust initialization.

- Reduce the number of required input parameters for initialization for distributed parameter systems, but still achieve convergence for reasonable input values. This has to be done in the library design, and it often requires that *template*² models are provided that reduce inputs for specific configurations, including the boundary conditions.

²These are example models tailored for different applications in AirConditioning.

n_ref	2	4	6	10
total variables	11132	21992	32852	54572
dynamic states	38	74	110	182
iteration vars	301	601	901	1501
Init time [s]	3.3	13.8	34.4	113

Table 1: Steady-state initialization times for different discretizations of a six pass evaporator testbench from AirConditioning 1.0 using Dymola 5.3b.

The remaining parameters should be those that are typically measured for the device.

- Improvements of solver robustness in the simulation tool. Dymola recently introduced two new features: a global homotopy method for the solution of large systems and much improved handling for scalar systems. Due to the tearing technique, scalar systems are much more frequent than would be expected otherwise.
- Be aware of particular problem cases in the model equations and avoid them or rewrite them in a way that is numerically easier to handle.

Using all these techniques, initialization problems with thousands of iteration variables are possible to solve with the current Dymola version. Results from a computation benchmark are shown in Table 1.³ An open point for even larger equation systems is to use sparse methods also after symbolically tearing and reducing the size of initial equation systems.

7 Model encryption

To securely exchange accurate first principle based simulation models without revealing proprietary data to third party users, a careful balance has to be found between two conflicting requirements:

- If the model information is completely hidden, the model is similar to a black-box model and will often not be of much use to the end user.
- If too many model details are revealed, many others can be reconstructed with little effort.

Encrypted save total models in Dymola keep only the connector variables, top-level parameters and outputs visible to a user. By default, the new encryption method hides as much data as is possible. If users

³The benchmark was performed on a 3.2 GHz Pentium 4 with 512 MB memory.

require additional input parameters or outputs, these have to be propagated explicitly to the top level by the owner that exports the model and makes it available. The method *Encrypted save total* consists of two distinct phases:

1. First, the model is pre-processed in a step called *scrambling*, which flattens the model (removes the composition hierarchy), evaluates all expressions in the model that can be evaluated, and changes all variable names in the model to generic ones. The evaluation of parameters removes most sensitive parameters completely from the model.
2. In a second step, the scrambled model is also encrypted. In the user interface, the encrypted model shows only the information needed to use and run the model; connectors and public, top-level parameters.

The unique advantage of the new encryption method is that sensitive information is irretrievably removed from the model in many cases. Consider e.g. the computation of a volume from parameters width, length and height: $V = w * l * h$. After scrambling, only the value for V remains in the scrambled code. Obviously it is impossible to back-calculate the original parameters from this information.

8 Transient simulation of automotive systems

In the past, the influence of AC-systems on fuel consumption has been neglected by legislative bodies and automotive manufacturers. This situation is currently changing, and accurate fuel consumption estimates are needed also for the case of a running AC unit. Figure 7 shows some of the key system parameters when running a New European Driving Cycle (NEDC) that contains an urban as well as an extra-urban section. In [9], more results from simulating driving cycles using the AirConditioning library are presented. Models from AirConditioning can be coupled directly to the PowerTrain Modelica library [10] for fuel consumption calculation.

9 Summary

AirConditioning is a comprehensive Modelica library for the simulation of automotive air conditioning systems. AirConditioning contains models for current,

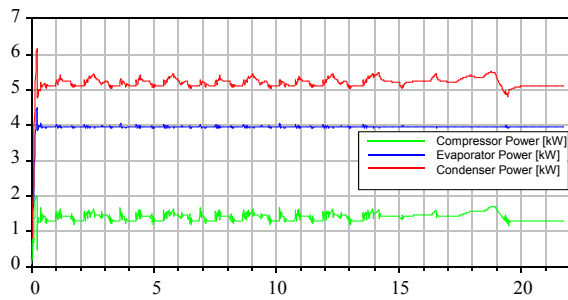


Figure 7: AC-system key parameters during NEDC driving cycle. Condenser power is the top line, compressor power the bottom line.

R134a based systems as well as systems under development using R744. It has been chosen by a group of automotive OEM and suppliers as a standardized tool for exchanging models for automotive AC-systems. Dynasim AB has added a new encryption method to accommodate the exchange of models containing proprietary data. The refrigerant and air side models have been adapted to cover the accuracy needed for component simulation and the flexibility and speed needed for system simulation. AC components and systems can be simulated in steady-state and dynamic conditions, and the models can be coupled to other Modelica libraries, e.g. for powertrain models.

References

- [1] Eborn, J. *On Model Libraries for Thermo-hydraulic Applications*. Ph.D. thesis TFRT-1061-SE, Dept of Automatic Control, Lund Inst. of Technology, Lund, Sweden, 2001.
- [2] Tummescheit, H. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. Ph.D. thesis TFRT-1063-SE, Dept of Automatic Control, Lund Inst. of Technology, Lund, Sweden, 2002.
- [3] Försterling, S. Personal communication, 2004.
- [4] Försterling, S. *Vergleichende Untersuchung von CO₂-Verdichtern in Hinblick auf den Einsatz in mobilen Anwendungen*, Ph.D. thesis, TU Braunschweig, 2004.
- [5] Casella, F. and Leva, A. Modelica open library for power plant simulation: design and experimental validation, In *Proc. of 3rd International Modelica Conference*, Linköping, Sweden, 2003.
- [6] Olsson, H., Elmqvist, H. and Tummescheit, H. Using Automatic Differentiation for Partial Derivatives of Functions in Modelica, In *Proceedings of the 4th International Modelica Conference*, Hamburg, 2005.
- [7] Singh, G.M., Hrnjak, P.S. and Bullard, C.W. Flow of Refrigerant R134a through Orifice Tubes, *HVAC & Refrigeration Research*, **7**:3, pp. 245–262, July 2001.
- [8] Kim, Y. and O’Neal, D.L. A Semi-Empirical Model of Two-Phase Flow of Refrigerant-134a through Short Orifice Tubes, *Experimental & Thermal Fluid Science*, **9**:4, pp. 426–435, 1994.
- [9] Limperich, D., Braun, M., Schmitz, G. and Prölb, K. System Simulation of Automotive Refrigeration Cycles, In *Proceedings of the 4th International Modelica Conference*, Hamburg, 2005.
- [10] Dynasim AB, <http://www.dynasim.se/models.htm>, Accessed January 2005.
- [11] Pfafferott, T., *Dynamische Simulation von CO₂ Kälteprozessen*, Ph.D. thesis, Department of Thermodynamics, TU Hamburg-Harburg. In *Berichte aus der Thermodynamik*, Shaker-Verlag, Aachen, 2005.
- [12] Incropera, F.P. and DeWitt, D.P. *Fundamentals of Heat and Mass Transfer*, Wiley & Sons, 5th ed., New York, 2002.

System Simulation of Automotive Refrigeration Cycles

Dirk Limperich, Marco Braun, DaimlerChrysler AG

Katrin Pröhl, Gerhard Schmitz, Hamburg University of Technology, Dept. of Thermodynamics

Dirk.Limperich@DaimlerChrysler.com, k.proelss@tu-harburg.de

1 Introduction

Numerical simulation in the automotive design process is gaining increasing significance. This also applies for thermodynamic and thermohydraulic systems i.e. also for the air conditioning system (HVAC) of an automotive vehicle. In order to optimize the efficiency of HVAC-systems and to obtain a better understanding of the complex transient system behaviour of automotive refrigeration cycles a Modelica library named ACLib was developed in a joint research project by DaimlerChrysler AG, Airbus Deutschland GmbH and TUHH. It was based on the thermohydraulic models of the free ThermoFluid library [2] and was applied to automotive refrigeration cycles running on the refrigerants R134a and carbon dioxide in an early development stage, when only limited experimental data was available [1, 5].

In order to share development costs and to combine the system knowledge and expertise on vehicle boundary conditions on one hand and the detailed component knowledge of the supplier on the other hand, a standardization process for a refrigeration cycle simulation tool was initiated by DaimlerChrysler. Modelica/Dymola was chosen by a pool of several OEM's and suppliers for this task, after several tools had been extensively investigated.

A new Modelica library, named AirConditioning Library, which meets the requirements of industrial application, and partly based on the former ACLib is currently developed by Modelon AB. Due to its user's group as mentioned above, special emphasis is placed on decoupling of physical equations and possibly confidential and encrypted component data as well as on high model flexibility.

2 Automotive Refrigeration Cycle

A Heating Ventilation and Cooling system (HVAC) is the primary element in controlling environmental temperatures of an enclosed automotive cabin. The

HVAC systems also provide fresh outdoor air and adjust the temperatures and humidity to improve comfort and increase efficiency (e.g. increase cabin air circulation). Figure 2.1 shows the design of a common automotive HVAC system. The design of the AC-System (e.g. R134a refrigeration cycle) is important for the cooling performance and demands a high attention. A common R134a refrigeration cycle consists of a compressor, condenser, high pressure receiver, expansion device evaporator and several hoses and tubes.

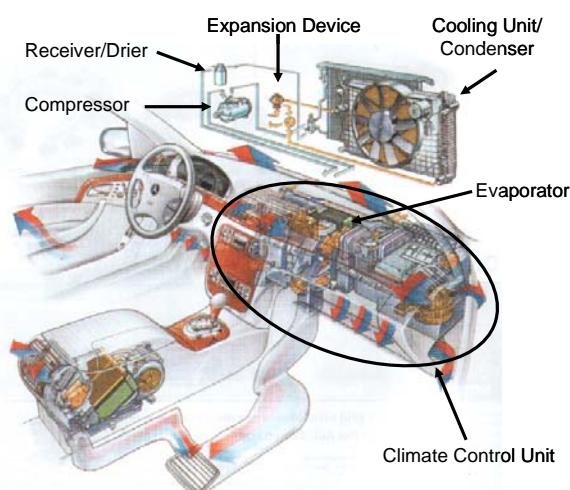


Fig. 2.1: Design of an automotive HVAC-Unit

As shown in figure 2.2, the R134a refrigeration cycle is a subcritical vapor process. The process path in the p -Diagram is represented by the numbers 1-2-3-4 and shows the compression (1-2), isobaric heat rejection at the condenser (2-3), adiabatic expansion (3-4) and the isobaric evaporation (4-1). In steady state the high pressure receiver is also represented by number 3. In most cases the receiver is totally filled with liquid R134a, assumed the refrigeration plant is sufficiently charged.

The refrigerant mass is an important factor of a vapor cycle and is one motivation for a complete transient simulation. During different boundary conditions (e.g. changing air temperature, air massflow through heat exchanger) conditions the refrigerant mass is moving to different parts of the system and must be observed. The task is to find out

the optimal charge of the system and the changing process behavior during any variation of the compressor speed, air massflow and temperature.

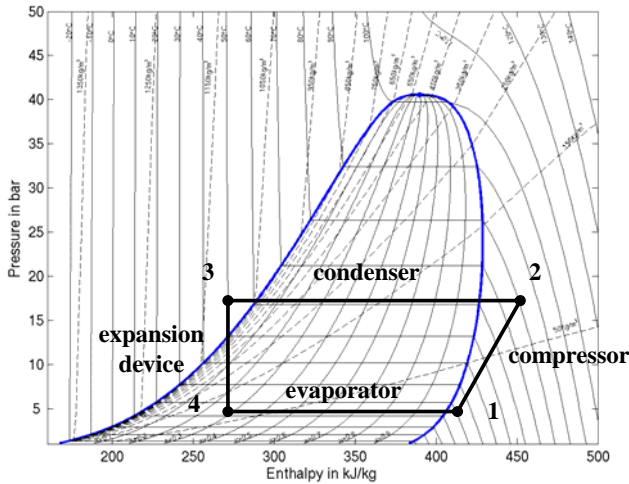


Fig. 2.2: Ideal case of R134a vapor cycle

Due to the additional fuel consumption which is about 100 liter gasoline [7] per year an exact and realistic simulation is necessary for efficiency optimization.

3 Heat Exchangers

In order to capture the transient as well as the steady-state behaviour of the complete automotive refrigeration cycle, detailed models of both main heat exchangers, condenser/gas cooler and evaporator, are required. They need to reproduce correctly the heat transfer between refrigerant and air flow and their respective property changes under given boundary conditions. Commonly used in automotive applications are compact cross flow heat exchangers that use finned flat tubes with internal microchannels as shown in Figure 3.1. Cross-co as well as cross-counter flow versions are also widely used for evaporators.

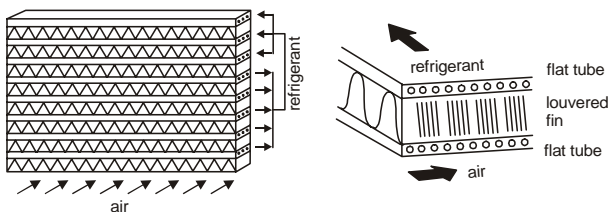


Fig. 3.1: Schematic of fluid flow in compact cross flow air-refrigerant heat exchangers

Based on physical parameters and heat transfer correlations from the literature, the heat exchanger

model from the AirConditioning library is suitable for a wide range of applications without requiring experimental input data. It can be used for the evaporator on the low pressure as well as for the condenser / gas cooler in the high pressure side. Due to the object oriented approach of the used Modelica language and a standardized interface the heat exchanger component can be used in variable cycle positions and also as multiple instances with different parameterization, i.e. as two evaporators operated in parallel.

The component specific parameterization, geometry data, heat transfer and pressure drop correlations, is decoupled from the physical equations and therefore allows storage of confidential and encrypted component data in a separate location.

3.1 Modelling approach

The fluid component models in the AirConditioning library are based on fluid flow models realized in the free Modelica library ThermoFluid [2][3]. This approach applies a finite volume method (FVM) allowing a numerically robust simulation of thermo-hydraulic systems including flow reversal. Mass and energy balances on one and the momentum balance on the other hand are solved on a staggered grid with upwind property propagation.

The dynamic formulation of energy and mass balances allows a representation of the transient system behaviour. However, the major contribution to transient component response rather originates from heat capacities of the solid wall material.

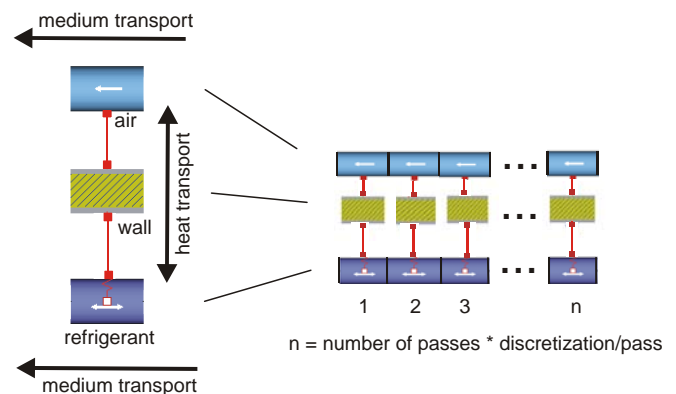


Fig. 3.2: Object diagram of heat exchanger composition from air, wall and refrigerant submodels.

The heat exchanger model is composed from two fluid objects (air and refrigerant) and one wall element. The wall mass is determined from detailed

geometry input data and therefore reflects distributed capacities. Heat conduction in the solid material is modelled one-dimensional and perpendicular to both fluids, longitudinal conduction is neglected for efficiency reasons and because no significant loss in accuracy is expected [4]. Heat is transferred between wall and fluid using a heat connector class (Fig. 3.2). Further information on the heat exchanger composition approach can also be found in [5].

Heat transfer correlations for both fluids from the literature, e.g. by Chang et. al. [6] for airflow through louvered fins, are part of the library and used as replaceable classes in the component. They are easily replaced by correlations determined from experimental data. In the same way pressure drop correlations, geometry data or model switches as e.g. air humidity condensation can be set by the user in a top level dialog.

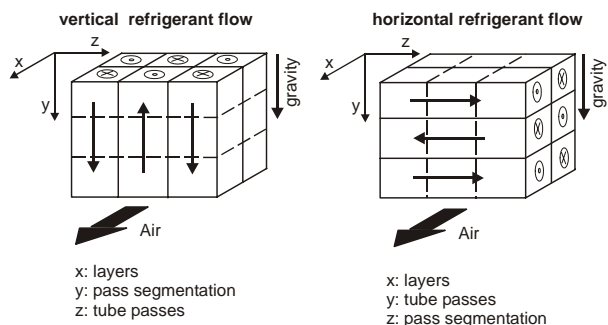


Fig. 3.3: Orientation of component triple in 3D-space

A single pipe approach combines all parallel refrigerant flows through the component in a single flow with variable cross section, resulting in an array of cross flow elements. They have to be formed into a 3D structure to support different flow schemes. Using a defined coordinate system with respect to air flow and gravity (fig. 3.3), the generic heat exchanger model can handle arbitrary flow patterns. It also allows a defined interface for inhomogeneous air inlet, which can be coupled with external 2D data. However, resulting from the one-dimensional flow approach in favour for numerical efficiency, the air inlet (and outlet) resolution is restricted by flow discretization (separated by dotted lines in figure 3.3) and the number of refrigerant passes (separated by solid lines in figure 3) in the component.

3.2 Validation of evaporator model

Simulations in a test configuration have been run with the described models. The test configuration (fig. 3.5) consisted of a source providing mass flow

and enthalpy at the heat exchanger inlet and a sink generating a defined pressure at the outlet. The source and sink were used to set the boundary conditions resulting from data measured at the component. The following comparison was made for a cross-counterflow evaporator from an automotive R134a-system built at Chrysler (Michigan, USA). The heat exchanger is shown in Figure 3.4. The geometric parameters of the component are all known. In Table 3.1 the measured data and the results of the simulations at steady state are shown. The comparison of experimental data and simulation results show very good correspondence in transferred heat. The calculation of the refrigerant side pressure drop and the air side water condensing (drainage) has to be revised.



Fig.3.4: R134a-Evaporator, cross counter flow

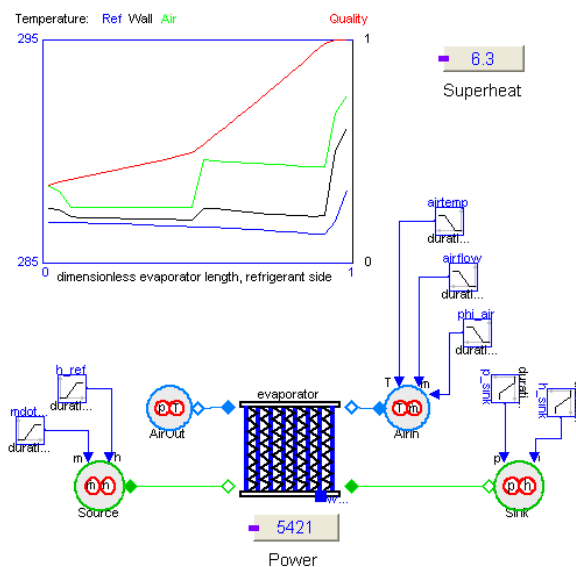


Fig. 3.5: Object diagram of test configuration

Boundary conditions for the evaporator from measured data							
\dot{m}_{Air}	\dot{m}_R	p_R	$T_{R,in}$	$h_{R,in}$	$T_{Air,in}$	$r.H_{Air,in}$	
[kg/s]	[kg/s]	[MPa]	[K]	[kJ/kg]	[K]	[%]	
0.132	0.043	0.633	296.5	289.6	330.3	19.2	
0.132	0.048	0.434	281.8	271.2	316.5	18.9	
0.132	0.054	0.439	281.3	272.7	316.5	19.1	
Measured data			Simulation				
$h_{R,out}$	$T_{Air,out}$	\dot{Q}_R	$r.H_{Air,in}$	$h_{R,out}$	$T_{Air,out}$	\dot{Q}_R	$r.H_{Air,in}$
[kJ/kg]	[K]	[kW]	[%]	[kJ/kg]	[kJ/kg]	[kW]	[%]
410.7	296.7	5.19	82.5	411.3	298.6	5.2	89.9
401.4	283.7	6.25	79.7	385.6	283.6	5.5	89.8
400.2	282.7	6.84	70.8	380.3	282.8	5.78	89.1

Table 3.1: Comparison of measured data at the evaporator with the simulation results in steady state

4 New European Driving Cycle

The New European Driving Cycle (NEDC) consists of defined vehicle speeds in an urban as well as an extra-urban section (fig 4.1). The NEDC is part of the emission test EURO 4 and is also widely used as a standard for fuel and energy consumption experiments and evaluation.

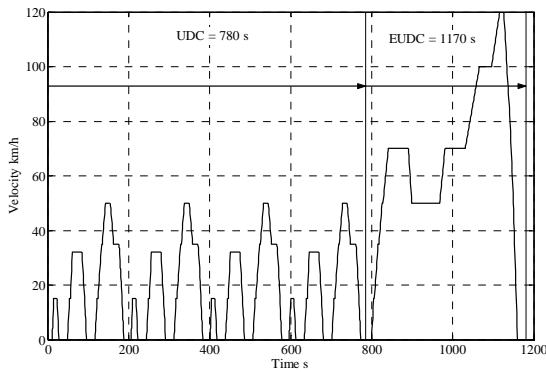


Fig. 4.1: Driving speed during NEDC

The resulting compressor speed and air velocity are in the following used as boundary conditions in a complete cycle simulation of a defined passenger car (fig 4.2). The air inlet temperature for the evaporator is constant at 310 K and the air massflow is constant 0.166 kg/s. The boundary condition for the condenser is also shown in figure 4.2, the air temperature is constant at 320 K and the air massflow depends on driving speed.

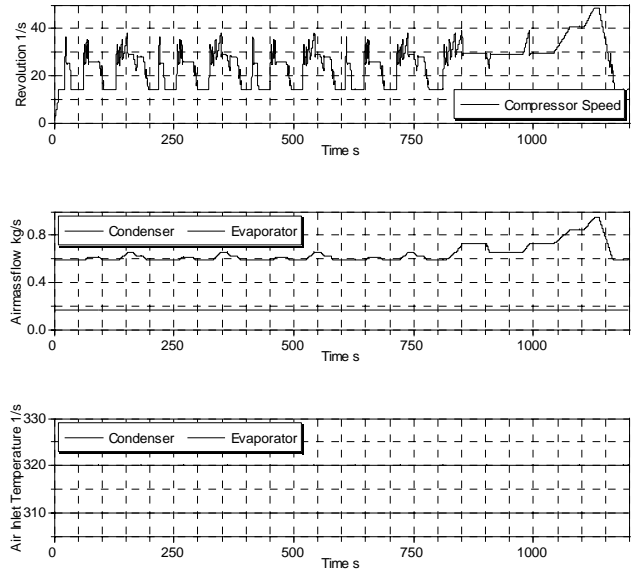


Fig. 4.2: Boundary conditions for the refrigeration cycle during the NEDC

The Modelica object diagram of the model used for the NEDC-cycle simulation is shown in figure 4.3. The refrigeration cycle consists of an external controlled compressor, a condenser, a receiver (with integrated drier), thermostatic expansion valve, evaporator and several pipes. The total volume and the ratio of the high pressure side and suction side volume are equal to the real refrigeration cycle. Also the refrigerant charge of the simulation model is equal to the real system.

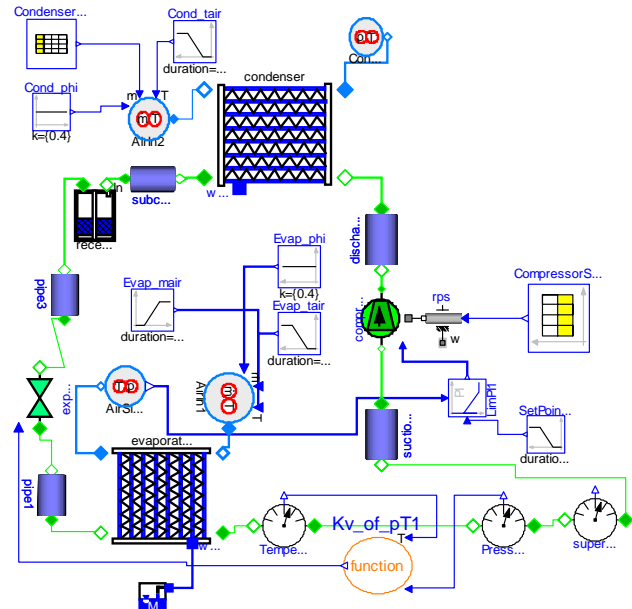


Fig. 4.3: Object diagram of complete R134a refrigeration cycle

In the following part, two simulation runs will be discussed. The refrigeration cycle shown in figure 4.3 provides a basis for the two experiments. For both experiment the described NEDC boundary conditions were used. Only two different compressors were used during the simulation runs. They are referred to as compressor A and B in the following. Compressor A has a 6 percent higher displacement and one more cylinder, so that the characteristic diagrams of both compressors are also different.

The results of the simulation runs are shown in the figures 4.4 -4.6. Three characteristic values of the refrigeration cycles are shown and compared: the air temperature behind the evaporator, the cooling capacity and the required compressor power.

When using compressor B in the cycle the air temperature behind the evaporator during UDC part is mostly 1K higher than using the compressor A.

Only during the EUDC part there is nearly no difference between the temperatures (fig. 4.4).

The comparison of the cooling capacity and the compressor power shows the same behavior (fig. 4.5 and fig. 4.6). The total energy consumption for type A is 733 Wh and for type B 720 Wh. Interesting is, that during the EUDC the smaller compressor needs less power for equal cooling capacity.

The solution of that experiment is that a 6% smaller compressor could perform nearly the same cooling capacity.

It is concluded, that the simulation tool is able to predict the change of cycle behavior when the accuracy of the component models is high enough.

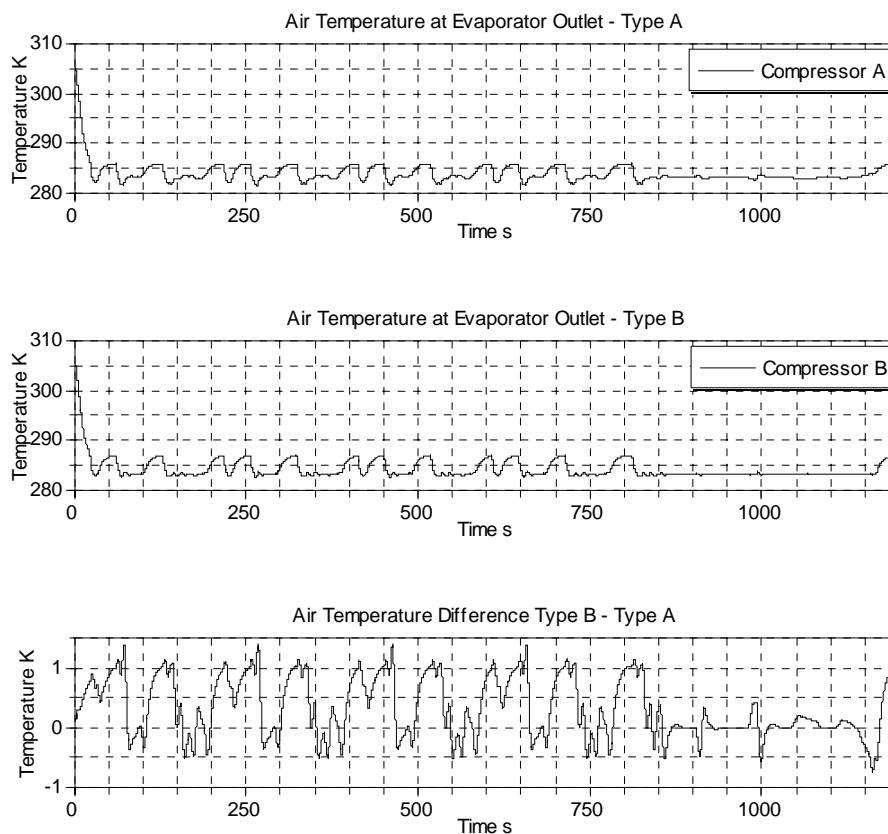


Fig. 4.4: Air Temperature behind evaporator - reference temperature for controlling $T=283.15\text{K}$

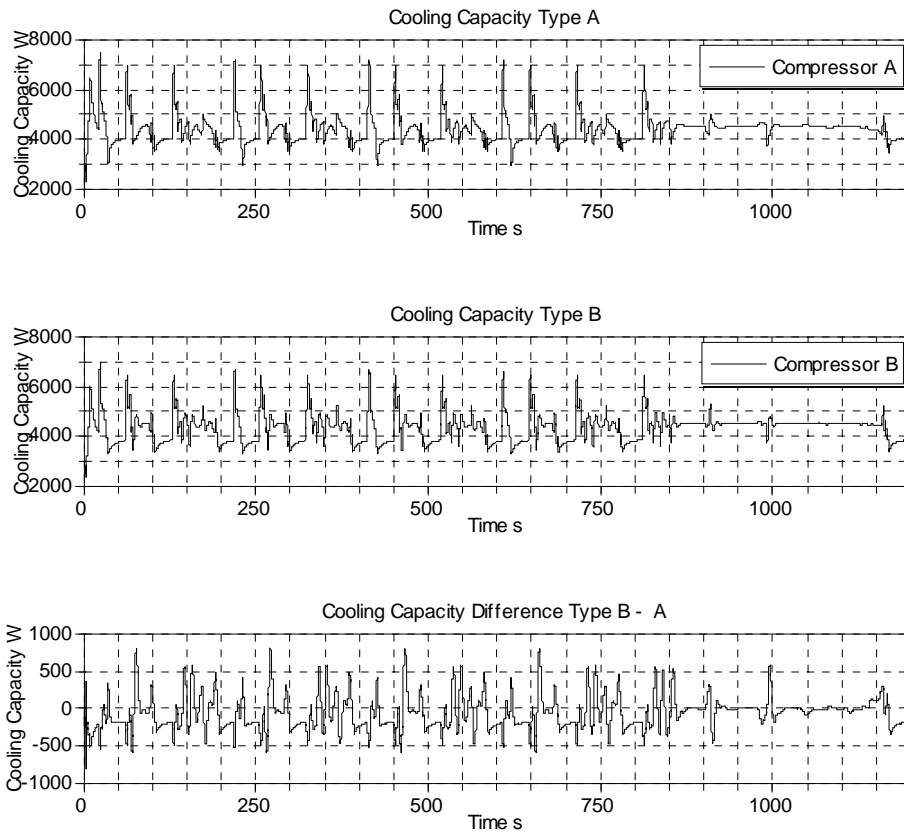


Fig. 4.5: Comparison of cooling capacity using compressor A and B

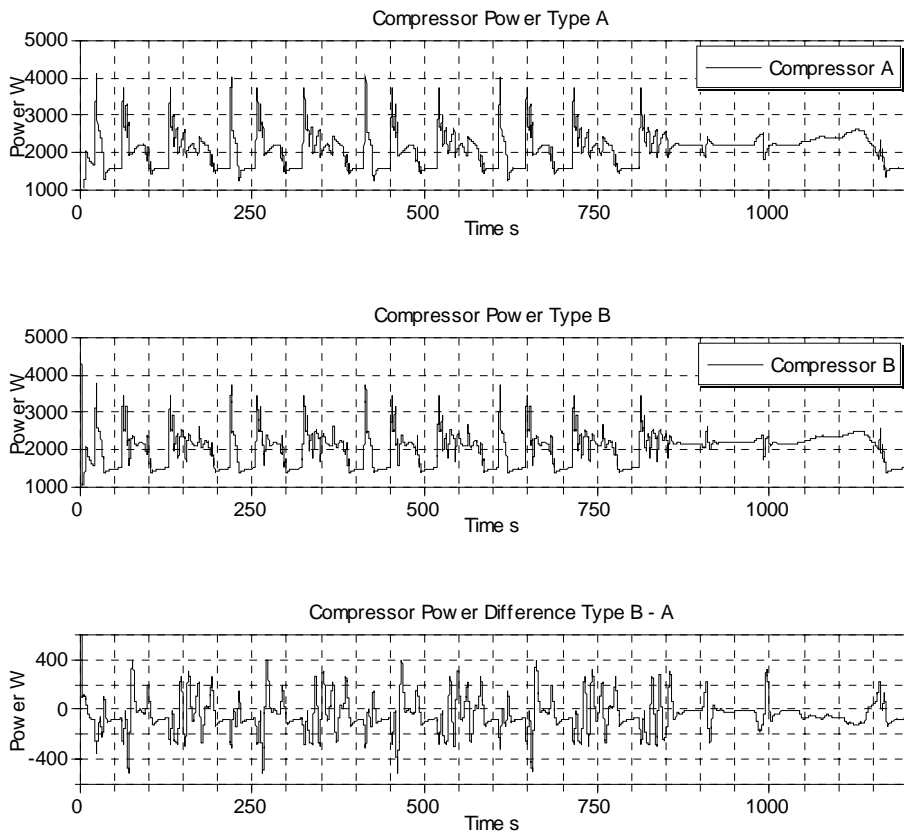


Fig 4.6: Comparison of Compressor power for compressor A and B

5 Common Simulation Tool in German Automotive Industry

In Germany, a working group with members of Audi, BMW, DaimlerChrysler and Volkswagen have compared different simulation tools in order to standardize the simulation of refrigeration cycles. The advantages of standardization are mainly the ability to integrate the supplier into the simulation process. This makes it possible to simulate components of different suppliers on one simulation platform. After a benchmark test the group decided to use Dymola Modelica for simulating refrigeration cycles.

In the future models of refrigeration cycle components are needed during the development process. If the supplier is not able to create such a model, detailed information of the geometry in combination with measured data has to be provided during the offer phase.

6 Conclusions

The dynamic simulation of an automotive refrigeration cycle with Dymola/Modelica as part of the design process is described in the paper. In a cooperative effort between Hamburg University of Technology (TUHH) and DaimlerChrysler AG, a model library for modeling refrigeration cycles has been developed based on the AirConditioning library. Based on geometrical data, the single components can be modeled and composed to an entire cycle.

The validation of the evaporator is one example for the component simulation and the results are in good accordance with the experimental data.

The simulation of the NEDC has shown that a prediction of the process behavior is possible, so that the simulation is able to support the design of refrigeration cycles for automotive applications.

At last the standardization has the advantage or the chance that the component supplier's expertise as well as the automotive manufacturer's knowledge of vehicle parameters can be combined in a reliable simulation.

- [1] Limperich D., Pfafferoth T. and G. Schmitz: Numerical Simulation of Refrigerant Cycles with New Methods. Proceedings of International Congress of Refrigeration, Washington D.C., 2002.
- [2] Tummescheit, H.: *Design and Implementation of Object-Oriented Model Libraries Using Modelica*. PhD Thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, 2002.
- [3] Eborn, J.; *On Model Libraries for Thermo-hydraulic Applications*, PhD-Thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, 2001
- [4] Asinari P., Cecchinato L. and E. Fornasieri: *Effects of thermal conduction in microchannel gas coolers for carbon dioxide*. Int J Refrigeration 2004 27: 577-58.
- [5] Pafferoth, T.: *Dynamische Simulation von CO₂-Kälteprozessen für mobile Anwendungen*. PhD thesis, Hamburg University of Technology, Shaker Verlag, Aachen 2005
- [6] Chang Y.J, Wang CC: *A generalized heat transfer correlation for louver fin geometry*. Int J Heat Mass Transfer 1997; 40(3): 533-544
- [7] Schwarz, W.: *Prognose der R134a-Emissionen aus Fahrzeug-Klimaanlagen bis 2010/20*. Öko-Recherche, Büro für Umweltforschung, Frankfurt/M
<http://www.oekorecherche.de>

First Results in Cluster Simulation of Alternative Automotive Drive Trains

Mathias Hommel
Brieffach 1778
Volkswagen AG
D-38436 Wolfsburg, Germany

Abstract

Control software plays an important role in the development of alternative drive trains. Energy management intervenes with the control of the combustion engine, the transmission or an additional electrical machine in different ways. In order to develop the energy management before or parallel to the vehicle construction phase, a complex software development process is required that equally supports modeling, simulation and implementation.

In the R&D of Volkswagen cluster simulation was established to simulate the drive train of a vehicle as well as to develop algorithms for the relevant electronic control units (ECU).

The methodology of cluster simulation will be represented in the following article.

1 Introduction

For more than 20 years now Volkswagens deals with alternative concepts for automotive drive trains. First there were the electric vehicles with a comparatively simple control that converted the driver's wish into a driving torque. Today, however, as combustion engine and electrical machine can be connected with each other in multiple ways, one needs a complex control in order to influence the torque distribution depending on the driving conditions. The intention in this case is to positively influence comfort and driving capability.

The control of the different components of the drive train plays a central role in this context. The so-called energy management coordinates the torques of the drive train. The electrical energy storage capacitor is controlled by the so-called battery management and so forth. For example, energy management and battery management influence each other in a complex way.

The simulation of the drive train plays an essential role during the specification of the components as well as during modeling the control algorithms. The objective of this drive train simulation is a fast and manageable process for developing controller algorithms resulting in an automatic code generation within a software-in-the-loop (SIL) process. Manageable in this context means that the developer can react in a quick way to altering structures of the drive train. The simulation time should be faster than real time in order to be able to carry out parameter variations and code development fast. Therefore a cluster computer was built for the drive train simulation consisting of ten dual processor computers. The individual computers themselves are connected with each other with a Myrinet¹ network, which is an optical network.

Within the project SUVA (Surplus Value Hybrid Vehicle) that was supported through the European Community [1] Volkswagen built up a Volkswagen Bora Hybrid with a hybrid drive train (Figure 1).

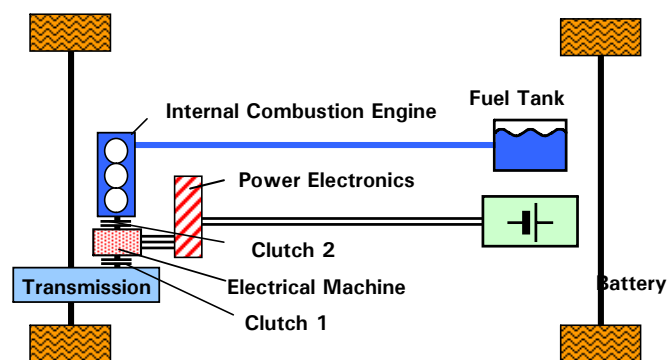


Figure 1: Structure of the Volkswagen Bora Hybrid

The drive train of the Volkswagen Bora Hybrid consists of a combustion engine (1.4 liters, 55 kW, 3 cylinders, turbo diesel), an electrical machine

¹ Myrinet is a registered trademark of Myricom, Inc., USA, <http://www.myricom.com>

(asynchronous machine, 25 kW peak), a clutch between the combustion engine and the electrical motor (clutch No. 2 in Figure 1), the automatic transmission (a dual clutch transmission, named by Volkswagen DSG^{®2}, [2]) and an energy storage (6 Ah NiMH-battery, 288V).

The transmission concept is such that the transmission is provided with a dual clutch on the gearbox input side (see Fig. 6 in chapter 2.1 as well). This clutch is represented in Figure 1 as being outside of the transmission for the sake of simplification (designated as clutch 1 in Fig. 1).

Due to this arrangement this specific drive train is called a parallel hybrid drive train since both combustion engine and electrical machine simultaneously or separately supply torque to the entire driving torque of the vehicle - acceleration to the strategy that is worked out in the above mentioned energy management ECU. For the classification of the different hybrid vehicles please refer to the relevant literature [3].

2 Simulation Model

The differences of the block-oriented or causal modeling using for example Matlab/Simulink³ and the acausal modeling using for example Dymola⁴ were described sufficiently [4].

While in the development of ECU algorithms the causal, graphical, signal-based modeling become more and more accepted in prototyping, acausal modeling has its advantages in the description of physical systems. The physical structure is maintained and the description corresponds to the local physical equations of the components that are independent of their environment, as well as their coupling to the entire system of equations.

For this reason the cluster simulation is realized by a simulator link-up: On the one hand Matlab/Simulink is used for modeling the ECU algorithms, the driver model (which generates the accelerator and brake pedal) and the driving cycle (which generates the reference vehicle speed value, height, air pressure and so on).

On the other hand Modelica⁵/Dymola is used for modeling the plant that is the closed loop controlled system vehicle. Furthermore, executable files of ECU algorithms in the form of a DLL (dynamic link library) are incorporated into the simulation (see Fig. 2). As it is shown in Figure 2 in the cluster simulation the essential algorithms of the relevant ECUs are simulated in Matlab/Simulink such as:

- the internal combustion engine ECU (ICE Controller⁶),
- the ABS/ASC-system (Brake Controller),
- ECU of the gearbox (Gearbox Controller) as well as
- the ECU of the electric machine (E-Machine Controller).

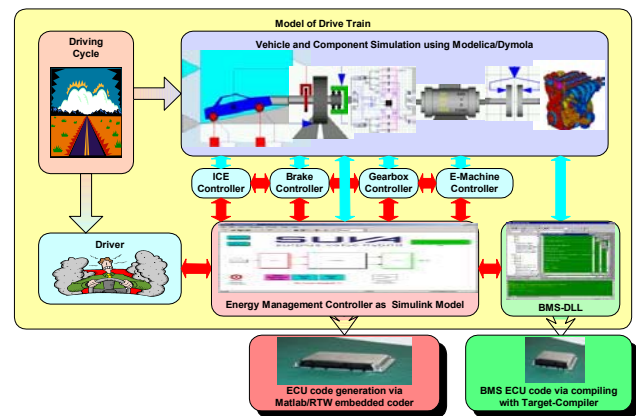


Figure 2: Structure of the drive train model of the cluster simulation

Furthermore, the controller of the energy management ECU which is modeled in Matlab/Simulink is integrated as well as the DLL of the controller of the battery management ECU (BMS, battery management system).

The plant was modeled in Modelica/Dymola as already described above and can be linked to the cluster simulation either as a Dymosim.exe or as a Dymola model.

Executable files (so-called executables or exe) were generated from all models since it is to be expected that through the detailed modeling the performance of the cluster simulation is lowered due to simulation of uncompiled models.

² DSG is a registered trademark of Volkswagen AG, Germany

³ Matlab und Simulink are registered trademarks of The Mathworks, Inc., USA

⁴ Dymola is a registered trademark of Dynasim AB, Sweden

⁵ Modelica is a registered trademark of the Modelica Association

⁶ The term controller synonymously stands for a closed loop control algorithm that is the functional software of an ECU (electronic control unit).

The cluster simulation is a so-called forward simulation in contrast to the so-called backward simulation. Starting from a driving cycle the comparison of the actual value of the vehicle velocity with the reference value is done by the driver model which then generates the accelerator or brake pedal command. The strategy of the energy management then controls the components of the drive train to generate the necessary driving torques in order to follow the reference value of the vehicle velocity of the driving cycle. The actual value of the vehicle velocity is traced back to the driver model.

In the following, the individual, modeled systems are shortly described as well as the structured components library in Dymola and the simulator coupling.

2.1 Model of the Plant

The vehicle is structured on the highest modeling level into the three large blocks: chassis (CHS), power train (PTR) and auxiliaries (AUX) (Figure 3).

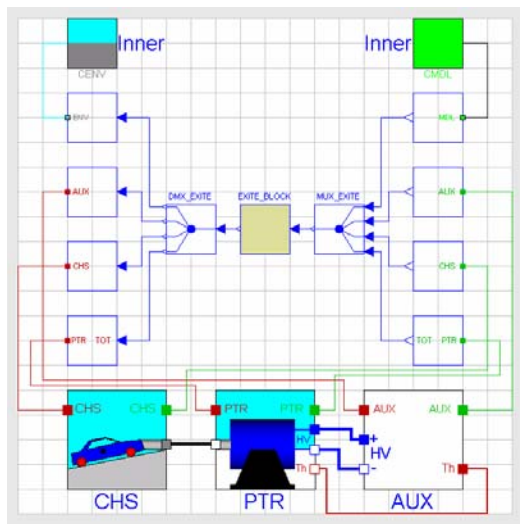


Figure 3: Structure of the vehicle model

The block EXITE_BLOCK represents the simulator coupling described in the chapter 2.4.

The model of the chassis (CHS) incorporates a vehicle model (BOD i.e. body) without lateral dynamics considering all relevant driving resistances as well as a model of the contact of the tire with the street (Figure 4).

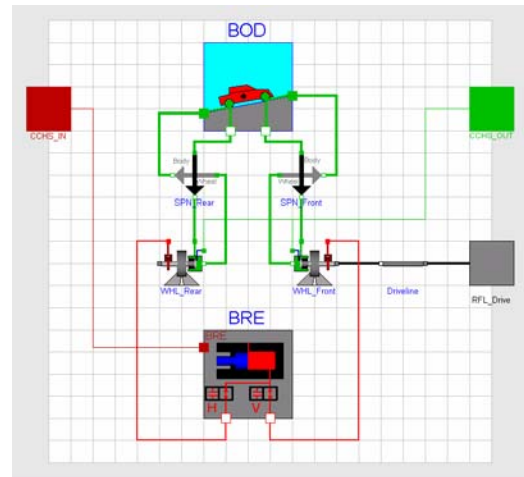


Figure 4: Model of the chassis (CHS)

Furthermore it incorporates a simple hydraulic model for the excitation of the brakes (BRE, which stands for brake model).

The model of the auxiliaries (AUX) consists of the modeled electrical consumers of the vehicle electrics (14V).

The model power train (PTR) represents the relevant components such as the internal combustion, ICE, engine (VKM, German: Verbrennungskraftmaschine), the fuel reservoir (TNK, German: Tank), the clutch between ICE and electrical machine, called separating-clutch (TRK, German: Trennkupplung; clutch 2 in Figure 1), the gearbox (GTR, German: Getriebe), the high voltage battery (BAT) and the electrical machine (Figure 5).

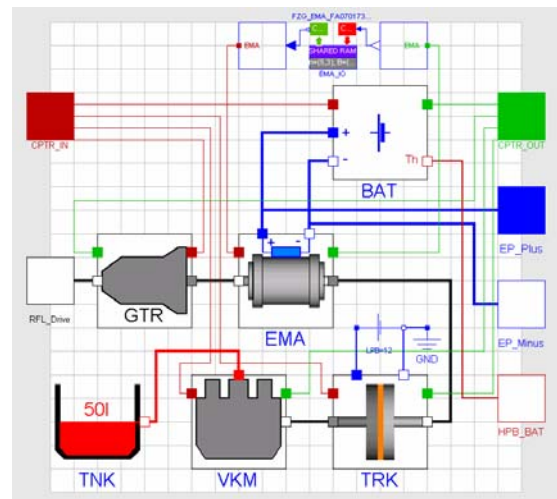


Figure 5: Model of the power train (PTR)

In the upper part of Figure 5 the shared-memory data-link is displayed which is described in chapter 3.1.

Only the internal combustion engine was modeled based on maps (efficiency maps). All other components are equation-based models and are described shortly.

The model of the fuel tank is a simple model of the flow of the fuel.

The model of the separating-clutch is a complex mechatronic model of the clutch and the flywheel. Even the hydraulic actuator and the mechanics of the separating-clutch were modeled.

The induction motor was modeled as an electromechanical drive in α - β stator coordinates based on the well known equations [5].

The NiMH battery was modeled with electrical and thermal characteristics including ventilation.

The model of the automatic transmission is a complex mechanical and hydraulic representation of the DSG[®] (Figure 6).

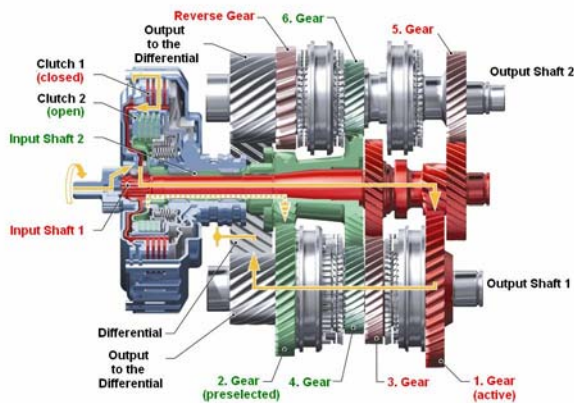


Figure 6: Section of the DSG

2.2 The Vehicle Modeling Library – VML

Dymola supports object-oriented modeling. Class libraries can be created in so-called packages. Since with the aid of the cluster simulation different drive train configurations can be examined, from the start the emphasis was put on a hierarchically structured component library (Figure 7).

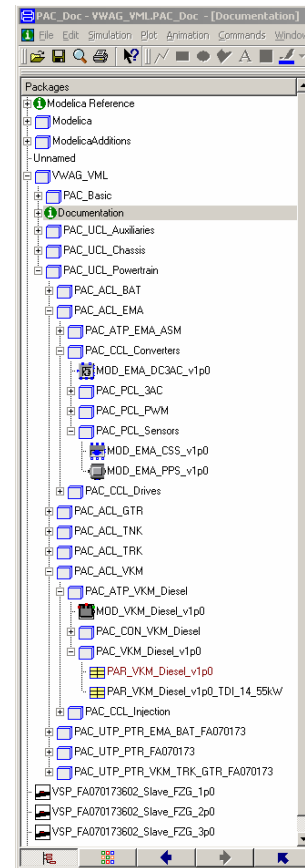


Figure 7: Structure of the Dymola library VML

This library handles the components and the component structure as well as the variants of components. In the following, attention is shortly being paid to the construction of the component-library VML.

There are basically four levels of hierarchy: the UCL - classes of the models of complete units, the ACL - classes of the aggregate models, the CCL - the classes of the components and their parts – the PCL. In order to structure the system vehicle into manageable subsystems the vehicle is structured into the abovementioned subsystems chassis, powertrain and auxiliaries which were named units. The main parts of a unit are referred to as aggregates, such as the ICE or the gearbox of the unit powertrain. Parts of aggregates are referred to as components such as the converter or the electrical drive itself of the aggregate electrical machine. So-called parts are elements of components such as different sensors or the voltage conversion of the component converter.

This structure results from the principle of decomposition assuming the following: units consist of aggregates, aggregates consist of components and components consist of parts. The structure is displayed by the structure of packages in Dymola. The packages by themselves are subdirectories on the hard disk.

Classes of units (UCL) and aggregates (ACL) furthermore can contain packages of connectors (Pac_CON_...) and models (Pac_UTP_... and Pac_ATP_... resp.; TP stands for a special type or model). Archived models of particular aggregates or units are stored in the type package. Parameterizing is supported by using records.

2.3 Modeling of the Controller

The difficulty of the simulation of the drive train is not mainly modeling the physical system; it can be modeled with sufficient accuracy with more or less effort.

The problem rather is the modeling of the control algorithm of the components that represents the control characteristic. So the functionality of the ICE control, the transmission control, the electrical machine control and the ABS/ASC control was modeled with relatively large effort. All models in Matlab/Simulink were modeled discrete (for instance fixed step size 10 ms).

The algorithm of the battery management system (BMS, that is the ECU that controls the battery with respect to its boundaries) could be directly inserted into the cluster simulation as a DLL since the algorithm was developed by the author himself. And the complete algorithm of the energy management ECU (Vehicle Management Unit, VMU), which was developed in Matlab/Simulink, could be inserted too. For these two last-mentioned ECU algorithms there is a software development process with which one can generate the flash code directly out of the simulation by means of automatic code generation or compilation with the target compiler for the ECU (see Figure 2). In case of the BMS this process is a C programming language software development process and for the VMU a Matlab/Simulink/Real-Time-Workshop (RTW) software development process. Both processes were used in this way within the above-mentioned SUVA project.

2.4 Master Model

In the so-called master model which was modeled in Matlab/Simulink the interface data of all submodels are exchanged via a network of the representatives of all submodels which is fed back on itself (Figure 8).

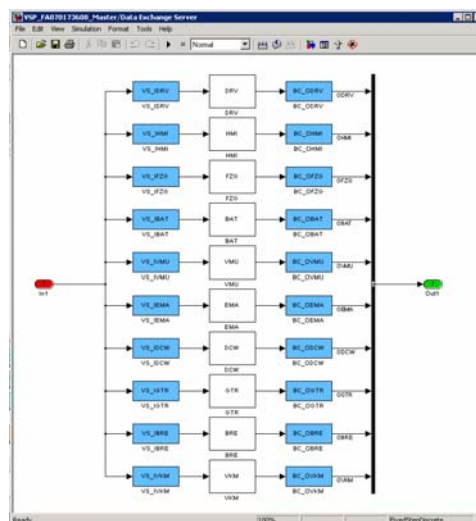


Figure 8: Feedback structure of the representatives of the submodels (the output is fed back without delay onto the input)

As already mentioned in chapter 2.1, the simulator coupling is carried out by the tool EXITE of the company Extessy AG, Germany [6]. EXITE realizes a simulator coupling on the basis of a client-server-linkage. The server is the representative of a submodel which only provides the interfaces according to the regarding submodel. The client is the submodel itself.

The Extessy AG provides a simulator coupling for different simulation tools for example for Matlab/Simulink, ASCET-SD⁷, Saber⁸ and Dymola.

Several methods for data communication between the client and server are supported such as the simple sequential communication and the full-duplex communication. EXITE relies on the ISO-OSI layer model of communication. So the communication protocols TCP/IP and MPI are supported too. The Master Model handles all interface data of all submodels, therefore it reflects the so-called communication matrix which shows which submodel exchanges which data with which submodel in what sample time. In this cluster simulation over 900 signals are exchanged mainly because of the emulation of the CAN bus (Controller Area Network – a commonly used network in the automotive context). 500 of these signals are relevant stimulation inputs for the simulation.

The VMU and the BMS are stimulated in this way with all signals available in reality. Thus this

⁷ ASCET-SD is a registered trademark of ETAS GmbH, Germany

⁸ Saber is a registered trademark of Synopsys, Inc., USA

simulation represents a real SIL simulation concerning these two ECU algorithms. The relevant signals are considered for the other ECU.

3 First Results

3.1 Shared-Memory-Coupling

The control of the electrical machine was realized with the switching frequency of the converter which is 8kHz. The model coupling of the electrical machine in Dymola and its control in Matlab/Simulink would slow down the simulation extremely. For this reason, a shared-memory data-link was created so that two processes access the same storage area. The processes are controlled so that both are processed on different processors of a dual processor computer. Thus the vehicle model is carried out on one processor as a Dymosim.exe. On the other processor the electrical machine control is processed as a Matlab/Simulink/RTW-executable. The shared memory block is represented in the upper part of the Dymola model in Figure 5.

3.2 Partitioning

The distribution of the submodels on the individual computers was done according to performance aspects, because the slowest simulation determines the overall performance of the cluster. For this reason the partitioning shown in Figure 9 was carried out.

PC 1 Driver Model, Driving Cycle and Human- Machine-Interface	PC 2 Brake Control and DC/DC-Converter Control	PC 3 Energy Management
PC 9 Vehicle Model and Control of Electrical Machine	PC 0 Master Model (Communication Matrix)	PC 4 Battery Management System
PC 7+8 not yet used	PC 6 ICE Control	PC 5 Gearbox Control

Figure 9: Distribution of the submodels on the individual simulation PC

It was expected that the vehicle model determines the performance of the overall system due to the detailed modeling of the components. For this reason two more computers were reserved (PC 7 and 8 in Figure 9) in order to split the vehicle model into sev-

eral parts and to simulate them separately in case of simulation overload.

3.3 Benchmarks

As described in chapter 2.4, different combinations of the communication protocols and of the communication methods are possible.

All benchmarks of cluster simulation were carried out with the full-duplex communication method with the protocol MPI/GM⁹ via Myrinet, the optical network.

The Matlab/Simulink models were modeled discrete. For the Dymola drive train model the integration algorithm **Isodar** (a multi-step-solver with a variable step size for continuous and discrete systems) has proved to be very robust.

In Table 1 the results of several benchmarks are listed. The third column contains the ratio of simulation time to simulated time (RT/tsim).

Table 1: Results of the performance measurements

Sim. No.	Configuration	RT/tsim
(1)	all models as dummies – i.e. empty models	67.1
(2)	all ECUs as RTW-exe, except gearbox ECU; vehicle as dummy models	8.9
(3)	all ECUs as RTW-exe, except VMU-ECU; vehicle as dummy models	5.9
(4)	vehicle model as Dymosim.exe, all ECUs as RTW-exe, EMA control with 8 kHz shared-memory data-link	1/390
(5)	as (4), EMA control with 4 kHz shared-memory data-link	1/216
(6)	vehicle model split into the electrical high voltage part, modeled in Matlab/Simulink and compiled to an executable and the remainder as Dymosim.exe; all ECUs as RTW-exe	1/15 ¹⁰

To examine the influence of the Matlab/Simulink ECU models on the performance of the cluster simulation in simulation No. (2) and (3) of table 1 the gearbox controller and the VMU controller, respectively, were replaced by empty models (so-called dummies). As mentioned above, all other Matlab/Simulink models were RTW-executables. Even the Dymola plant was simulated by a dummy. The simulations were 8.9 and 5.9 times faster than real time. This means that because of the complexity of the model of the gearbox controller the performance of the cluster simulation will be more influenced by the gearbox controller than by the energy management controller. Furthermore it is obvious that even

⁹ GM is a registered trademark of Myricom, Inc., USA

¹⁰ Estimated value based on simulation of the vehicle model without shared memory data-link and without control of electrical machine

if the plant could be simulated faster as the gearbox controller, the cluster simulation would be only a maximum of 5.9 times faster than real time when performing SIL-simulation for VMU controller algorithm development.

For the sake of comparison, the simulated time for a complete empty-cluster simulation is given in simulation No. (1). Only dummy models were simulated. It follows that the pure communication of empty models is 67.1 times faster than real time. Per simulation step (10ms fixed step), approximately 150 μ s is required (operating system and Matlab/Simulink overhead).

In simulation No. (4), all dummy models were replaced by their respective models. The vehicle model was compiled by Dymola into the executable Dymosim.exe. All Matlab/Simulink models were compiled by Matlab/RTW into executable files. The simulation of the control of the electrical machine (fixed step size) and therefore the data exchange via shared-memory data-link between the electrical machine in Dymola (variable step size) and the control of the electrical machine in Matlab/Simulink was carried out with a 8kHz switching frequency of the converter of the electric drive. As a result the simulation was 390 times slower than real time. Of course this result is caused by the communication step size of 125 μ s. As mentioned above, the integration algorithm used in Dymola was **lsodar** with a tolerance of 1E-5. (The model did not run by a tolerance of 1E-4.) The data exchange between vehicle model and the control of the electrical machine organized by EXITE every 10ms activates an event in Dymola. As a consequence additional CPU time is required through reinitialization during solving the differential equations and thus the performance of the system slows down.

Reducing the switching frequency and in this way the frequency of the data exchange between Dymola and Matlab to 4kHz still leads to a simulation which is 260 times slower than real time. Moreover, with this lower switching frequency at maximum rotational speed of the electrical motor no effective mechanical torques can be generated.

The transition of modeling the electrical machine in α - β stator coordinates to d-q field coordinates and, hence, the loss of the universal description of the machine for the benefit of the symmetrical machine would reduce the simulated time and the effort due to data exchange in such a way that the simulation rate would be moved into manageable proximity. The data exchange via shared-memory then could be done in 10ms steps. Only one disadvantage would arise: the harmonic pattern and consequently

the torque ripple of the electrical machine would be simulated no more. However, the torque ripple supplies an insignificant contribution with regard to its effects onto the torque characteristics of the drive shaft during the software development of the energy management algorithms and so it could be neglected.

Keeping this in mind, the high voltage electric part of the drive train is presently removed out of the Dymola vehicle model onto a Matlab/Simulink model which also includes the controller of the electrical machine (estimated simulation time see simulation No. (6) in Table 1). Thus the performance of the drive train simulation could be increased at least to 15 times slower than real time.

4 Alternatives for Increasing the Performance

It has been shown that the influence of the modeling of a complex controlled system on the performance of the entire simulation is quite important. To put it precisely: in the present cluster simulation the Dymola model determines the overall performance. For this reason possible alternatives for improving the performance of the cluster simulation will be described in this chapter.

4.1 Simplification of the Modeled Plant

The vehicle model must be redesigned in such a way that models with only small influence on the entire simulation or those with a vague description are reduced to simple constants or low-pass filters of first order.

The auxiliaries model for example: a constant efficiency and a constant load at the 14V power supply can be accepted. The effect on the development of the energy management algorithm is minimal and the error can be accepted.

4.2 Elimination of Stiffness

The stiffness values of the system have an essential influence on the performance of the cluster simulation. These must be identified and eliminated. Since that was not done until now, a further increase in performance can be expected.

4.3 Calculating Vehicular Submodels Separately

As it was shown in chapter 3.3 it is possible to shorten the simulated time by splitting the vehicle

model into several parts. A further alternative for increasing the performance therefore is the identification of effects with large time constants and submodels which can be calculated separately. So a second or a third Dymola session could be opened and for example the simulation of the auxiliaries could be calculated separately.

4.4 Usage of more Efficient Solvers

The simulation slows down due to events (10ms data exchange via EXITE, discontinuities in modeling) and because of the variable step size and the solver used in Dymola.

The most efficient way for speeding-up the simulation is the use of a single step solver for continuous systems with variable step size and state event handling. The usage of Dynasim's GODESS library (GODESS stands for generic ODE solving system) that incorporates such solvers is presently proved.

4.5 Replacing Modeled Controller by its ECU-DLLs

If the real ECU code of a controller is available, complex modeling can be avoided. Furthermore, the ECU code is often more efficient. Thus, the BMS code could be included into the cluster simulation. In the same manner the DSG[®]-ECU code could be linked to the cluster simulation. By doing so and together with all other herein mentioned possibilities for increasing the performance of this simulation there could be an increase in simulation time which would result to a 9 times faster performance than real time (see simulation No. (2) in Table 1).

5 Conclusions and Future Work

A complex mechatronic simulation was presented in a heterogeneous cluster of simulators used for hybrid drive train simulation in the automotive industry. The objective was to clarify whether or not it is possible to set up a manageable SIL process with extensive computational aid. As a result it can be said that on principle ECU algorithms can be developed with the aid of the presented method. An advantage of SIL compared to traditional applications in the vehicle is that the control algorithms can be developed robust in respect of fluctuations in components and environmental data and in a reproducible manner. Effects of the communication between the ECUs can also be examined. Decisive for the manageability of

such complex simulation is the level of detail of the submodels and the solver used.

With Modelica/Dymola as an object-oriented, multi-domain modelling tool it is possible to alter plant structures in a fast way.

One next step has to be the validation of the simulation. For this purpose, an approximately 300 km long driving cycle has already been measured.

In future more ECU algorithms will be linked as DLL into the cluster simulation which makes the control characteristics more realistic and reduces the amount of work necessary for modeling control algorithms.

For the development of controller algorithms and for the specification of components, an automated simulation will be designed; with it, parameters can be changed within their boundaries by predefined scripts or Monte-Carlo analysis, allowing massive parameter variations to be carried out automatically.

In order to obtain a manageable SIL, the cluster simulation has to be redesigned to be faster than real time.

6 Acknowledgements

This work was done within a project of Volkswagen R&D in cooperation with various partners. For this reason the author would like to thank the respective colleagues of the Audi Electronics Venture GmbH, of the Extessy AG as well as of the Lineas Automotive GmbH and there in particular Wolfgang Borgs and Vadim Bulakhov for their assistance in modeling and putting the cluster simulation into operation.

References

- [1] J.-W. Biermann, C. Bunz, M. Crampen, S. Köhle, D. Mesiti: Drei OEM, ein gemeinsames Antriebskonzept – Drei neue Hybridfahrzeuge, entwickelt im EU-Projekt SUVA, 13. Aachener Kolloquium Fahrzeug- und Motorentechnik, Aachen, 2004
- [2] Timo Götte, Thomas Pape: DSG – Das Direkt-schaltgetriebe von Volkswagen, VDI-Tagung "Getriebe in Fahrzeugen 2004", Friedrichshafen, 22.-23. Juni 2004
- [3] Roland Kube, Michael Böckl, Mathias Hommel, Siegfried Köhle: Energy Management Strategies for Hybrid Drive Train Sys-

tems Using Infrastructure Information, EUR-motor New Advances in Electronics Engineering, Energy Management – Today and tomorrow, 23.-24.9.2004, Aachen

- [4] M. Tiller, D. Linzen: A Comparison of Different Methods for Battery and Supercapacitor Modelling, SAE 2003-01-2290, 2003
- [5] Werner Leonhard: Control of Electric Drives, Springer Verlag, 2. edition, 2000
- [6] EXITE Handbuch Version 1.3.0, EXTESSY AG, Wolfsburg, 2004

Session 3b

Thermodynamic Systems II

Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library

C. Kral, A. Haumer, M. Plainer
Arsenal Research, Faradaygasse 3, 1030 Vienna, Austria

Abstract

SimpleFlow-library was created to model heat and coolant flows of simple thermal equivalent circuits. The main components of this library and their applications are presented in this paper. Furthermore, a thermal model of a surface cooled squirrel cage induction machine is introduced. The simulated temperatures are compared with measuring results which were obtained in the laboratory.

1 Introduction

Typical cooling models consist of a *thermal network model* and a *cooling circuit* of a device (e.g. an electrical machine) which is going to be cooled. The mechanism of coolant flow is different from heat conduction [1], described by the *thermal network model*. Therefore in the second section the *SimpleFlow*-library is introduced. Basic equations and components of the cooling model are presented, as well as the structure of the library. The third section introduces a complete thermal network model of a surface cooled squirrel cage induction machine (totally enclosed fan cooled), using the elements from `Modelica.Thermal.HeatTransfer`. The simulation is presented in the fourth section, whereas the measurement is described in the fifth section. The sixth section compares simulation and measurement results.

2 SimpleFlow Library

The description of coolant flows due to forced convection is difficult. The developed *SimpleFlow*-library was designed to model such coolant flows under the following conditions:

- Splitting of media flows is simple.

- Mixing of media flows obeying mixing rule can be realized easily.
- Reversing the direction of flow is possible.
- No complex media properties are needed.
- The medium is considered to be incompressible.
- Mixtures of different media are not taken into account. Each individual cooling circuit has to have a designated medium.
- Medium properties are considered to be constant.
- Pressure changes are only caused by pressure drops (due to friction of the coolant flow at solid surfaces).

The library design has been restricted to simple media as coolants, only taking basic thermodynamic effects into account, such as the transport of heat by a flowing medium. These prerequisites allow a very easy handling of the library and are sufficient for a wide range of applications, including the cooling of devices. Cooling of electrical machines is an important topic, because the forecast of machine temperature increases allows to improve the machine design as well as to reduce the machine size and mass, which ends up in competitive advantages. In these applications temperature rise of the coolant as well as pressure drop of the coolant flow are rather small, so the above-mentioned conditions are fulfilled satisfactorily. Other applications not fulfilling the above-mentioned conditions like complex thermodynamic processes have to be modelled using `Modelica.Media` and `Modelica.Fluid`, which are currently under development. So the *SimpleFlow*-library is not designed to compete with these high sophisticated thermodynamic libraries but to ease the modelling of simpler applications.

2.1 Equations

SimpleFlow-library has to take simple thermodynamic equations in to account. The following quantities have been chosen to describe the state of a coolant flow:

- *pressure* (`p`) and *temperature* (`T`) as potentials
- *mass flow* (`mflow`) and *simple energy flow* (`sEflow`) as flow quantities

The naming of *simple energy flow* is chosen to keep in mind that only the heat transported by the media's thermal capacity is taken into account, avoiding mix-up with thermodynamic energetic quantities like enthalpy.

The basic equations of a flow element are collected in partial models, placed in subpackages named `Partials` and `Friction` [2], [3], [4], [5]:

- Pressure drop is a function of mass flow: linear dependency is assumed to a limit where laminar flow is effective, and quadratic dependency is modelled for higher mass flows approximating turbulent effects.
- Mass flow balance:

```
flowPort_in.mflow +
flowPort_out.mflow = 0;
```
- energy flow balance:

```
flowPort_in.sEflow +
flowPort_out.sEflow + Q_flow =
m * cp * der(T);
```

 where `Q_flow` is the energy flow exchanged with the environment outside the medium, `m` is the medium's mass, `cp` represents specific heat capacity of the medium and `T` is the medium temperature within the element.
- Energy flow at the port where the mass flow leaves the element:

```
flowPort_out.sEflow =
flowPort_out.mflow * cp * T;
```
- Mixing rule at the port where the mass flow enters the element:

```
flowPort_in.sEflow =
flowPort_in.mflow * cp * flowPort.T;
```

The actual connectors of any component are `flowPort_a` and `flowPort_b`. If the medium flows from `a` to `b`, `a` is assigned to `in` and `b` is assigned to `out`. For the opposite flow directions `a` is assigned

to `out` and `b` is assigned to `in`. This means, that there are two sets of equations used depending on the actual flow direction of the medium. The handling of these two sets of equations is supported by the Modelica statement `semiLinear` [6].

Modelica ensures the correct summation of mass flows and energy flows as well as equity of potentials pressure and temperature of connected ports. Since the mixing rule is applied at the inlet port of an element according to the actual flow direction, the temperature of the port where the mass flow leaves the preceding element does not necessarily show the medium's temperature but the (possible) mixing temperature of the following element. The medium's temperature is represented by the internal state `T`.

Besides the definition of common media (air and water) and appropriate sensors for pressure and pressure drop, temperature and temperature drop, mass flow and energy flow the library puts the following components at the user's disposal.

2.2 Sources

- Infinite ambient with constant or prescribed temperature and pressure which is not influenced by ingoing or outcoming flows.
- An element which allows to define pressure level in a closed circuit, since flow elements only define pressure drops.
- Simple fans (neglecting the media mass within the fan) and pumps (taking the media's thermal capacity into account), allowing to define either pressure drop or mass flow.

2.3 Components

- Isolated pipes with and without consideration of medium mass
- Pipes (with and without medium's mass) with a thermal connector where heat is exchanged with a thermal network.
- A predefined simple cooler, containing a vector of cooler elements, each consisting of a pair of pipes, coupled with a thermal conductor.

The usage of the library is demonstrated with a couple of simple examples. These elements together with `Modelica.Thermal.HeatTransfer`

allow the modelling of complex applications like the cooling of an electric machine.

3 Thermal Equivalent Circuit

The components of a thermal equivalent circuit can be imported from `Modelica.Thermal.HeatTransfer`. The thermal networks are designed in the style of electrical components and circuits. The components of such a network are:

- Nodes are regions of constant temperature. The potential of a node represents the absolute temperature of that node. The SI unit of the absolute temperature is K.
- A loss source in the thermal circuit is equivalent to a current source in an electric circuit. There are loss sources where the precalculated losses have to be corrected by the actual temperature of the corresponding node in order to consider copper losses correctly. Other loss sources such as iron losses do not need a temperature dependent correction. The SI unit of the heat flow is W.
- Thermal resistors represent regions of heat conduction. For technical application such as electric machines, heat transfer is mainly heat conduction and convection. Heat radiation is usually not considered. The SI unit of a thermal conductance is K / W. A thermal conductor is the reciprocal of a thermal resistor. Its SI unit is W / K.
- Thermal capacitors represent the ability of storing heat energy in a certain region. The SI unit of a thermal capacitor is Ws / K.

The utilized thermal equivalent circuit is shown in fig. 1. With respect to the thermal heat conduction paths the induction machine is divided into three axial sections. The outer sections are the drive end (A-side) and the non-drive end (B-side). End-rings (ERA and ERB), end cap air (AIA and AIB), winding heads (WHA and WHB), housing (HOA and HOB) and cooling ribs (RIA and RIB) refer to either of these sides. The middle section consists of the rotor yoke (RYO), rotor slots (RSL), rotor teeth (RTO), the air gap (AGP), stator slots (SSL) and stator teeth (STO), stator yoke (SYO), housing (HOM) and cooling ribs (RIM).

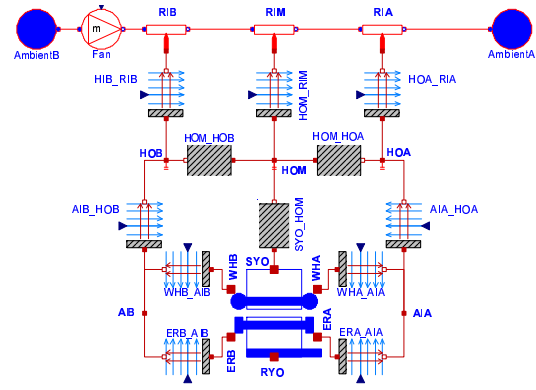


Figure 1: Thermal equivalent circuit of a surface cooled squirrel cage induction machine

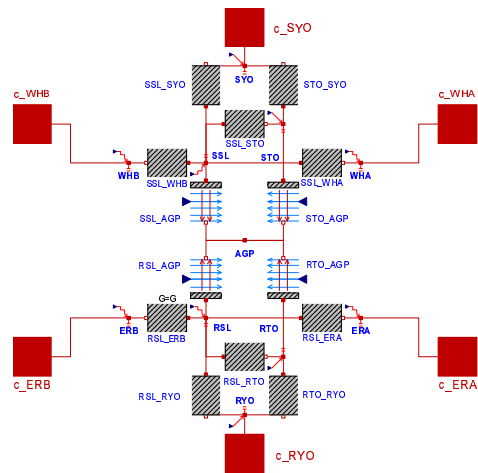


Figure 2: Thermal network of the active part

To achieve reusability, the active part – which is the same for many types of cooling – is modelled as a separate submodel with appropriate connectors (see fig. 2)

The losses of the induction machine have to be separated in accordance to the introduced model. Stator copper losses have to be divided into slot losses (LSSL) and the losses with respect to the winding heads (LWHA and LWHB) of each side. The ratio of these losses is directly proportional to the respective coil length within these sections. Rotor heat losses have to be divided into rotor slot losses (or bar losses; LRSL), and the losses with respect to the end rings of each side (LSRA and LEAB). Stator and rotor iron losses have to be determined with respect to yoke and teeth (LSYO, LSTO, LRYO and LRTO). Copper losses are precalculated and have to have temperature correction in order to model the actual losses accurately.

Therefore, there exist four types of nodes:

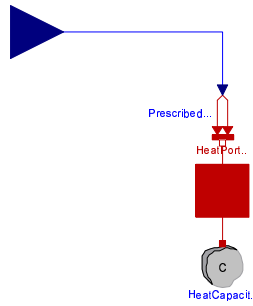


Figure 3: Node with constant losses

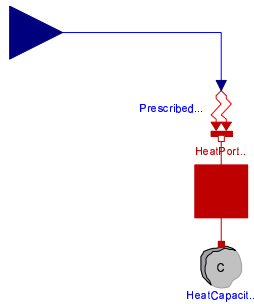


Figure 4: Node with temperature dependent losses

- Node without any properties, such as end cap air (AIA and AIB)
- Node with thermal capacity, such as the parts of the housing (HOA, HOB, HOM)
- Node with thermal capacity and losses (without temperature dependent correction), such as the stator yoke (SYO); see fig. 3
- Node with thermal capacity and losses (with temperature dependent correction), such as the stator slots (SSL); see fig. 4

Temperature dependent correction is done by the following formula:

$$\text{Losses}(T) = \text{Losses}(T_0) [1 + \alpha(T - T_0)] \quad (1)$$

where T_0 designates the reference temperature at which the temperature dependent copper losses have been calculated.

Two types of thermal conductances have been used:

- `Modelica.Thermal.HeatTransfer.ThermalConductor` with constant thermal conductance
- `Modelica.Thermal.HeatTransfer.Convection` where the actual thermal conductance is prescribed by a signal input.

This allows to define thermal conductance dependent on actual machine speed.

It is advantageous if loss components are directly available from machine design software. Otherwise, these components have to be estimated with respect to the current density or flux density and the mass of these sections.

4 Simulation

The geometric design data of the induction machine were available by courtesy of the machine manufacturer. The electromagnetic quantities such as magnetomotive forces (mmf), flux densities, current densities etc. were determined by motor design software *ASYN*. The output data of the motor design software deal as input parameters for the determination of the relevant thermal parameters of the machine. These parameters are the thermal conductances and capacitances as shown in fig. 1 and have been calculated as follows [2], [3].

- Thermal conductances in a homogenous region:

$$\frac{1}{R_{th}} = \lambda \frac{A}{l} \quad (2)$$

where λ designates the material specific thermal conductivity, A is the cross section and l is the length of heat conduction.

- Thermal conductances of heat transfer at a surface between solid and coolant flow:

$$\frac{1}{R_{th}} = \alpha A \quad (3)$$

where α designates the heat transfer coefficient which is dependent on coolant properties as well as the velocity of coolant flow and A is the surface area.

- Thermal capacity:

$$C = mc \quad (4)$$

where m is the mass of the considered region and c is the material specific heat capacity.

Heat transfer between cooling ribs and air flow was also modelled in three axial sections, using the elements of the *SimpleFlow*-library to describe the air flow. Air flow rate is adjusted proportional to the actual machine speed.

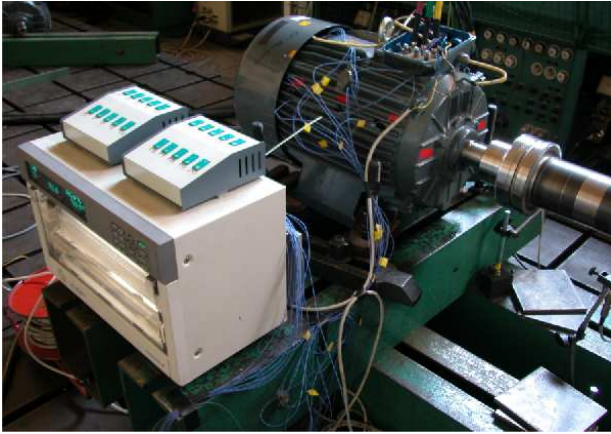


Figure 5: 18.5 kW squirrel cage induction machine with nickel-chromium-nickel temperature sensors and temperature recorder

5 Measurement

Measurements were carried out for a four pole, 18.5 kW squirrel cage induction machine with surface cooling. The machine is shown in fig. 5. The stationary parts of the machine were equipped with nickel-chromium-nickel temperature sensors:

- one sensor in the stator slot (two additional PT-100 sensor were already available in this machine)
- one sensor in a stator tooth
- three sensors in the winding head of each side of the machine in order to average the measured temperature in these areas
- one sensor in the stator yoke
- one sensor on each side of the end cap air (A-side and B-side)
- three sensors in the housing (A-side, middle, B-side)
- one sensor for ambient temperature
- one sensor for the air temperature in the cooling ribs at the B-side (blow-in)
- one sensor for the air temperature in the cooling ribs at the A-side (blow-out)

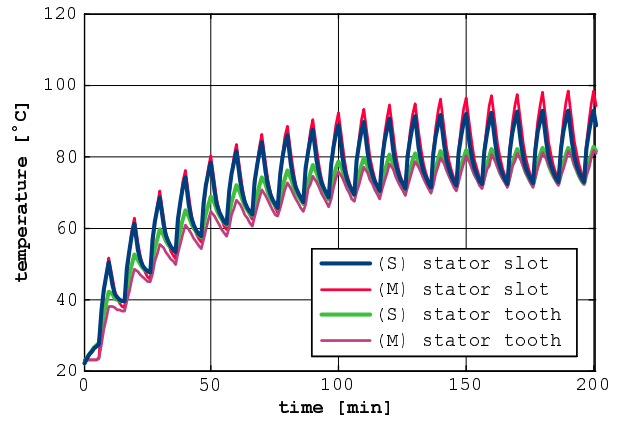


Figure 6: Simulated (S) and measured (M) stator slot (SSL) and stator tooth (STO) temperature

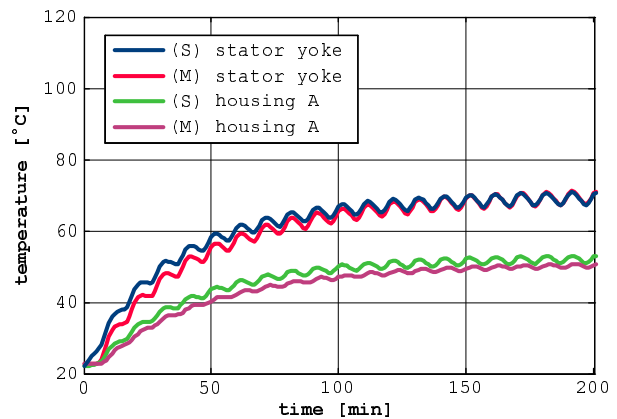


Figure 7: Simulated (S) and measured (M) stator yoke (SYO) and stator housing, A-side (HOA) temperature

6 Simulation and Measurement Results

Some results of computer simulation (S) and measuring (M) are compared in fig. 6–10. The investigations refer to continuous duty with intermittent periodic loading (duty cycle S6). The motor was loaded with 140% of nominal load for four minutes and no-load for six minutes.

Simulations and measurements match both qualitatively and in quantity.

7 Conclusions

A detailed thermal equivalent circuit of an asynchronous induction machine with squirrel cage was presented. The machine model was built using components from

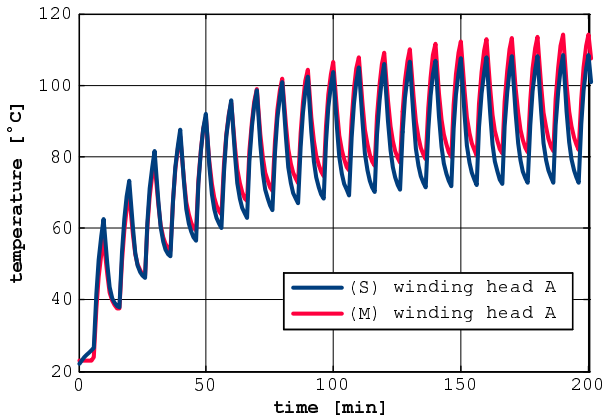


Figure 8: Simulated (S) and measured (M) temperatures of winding head, A-side (WHA)

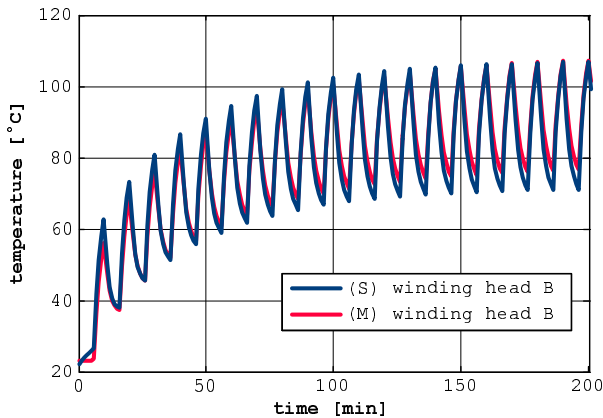


Figure 9: Simulated (S) and measured (M) temperatures of winding head, B-side (WHB)

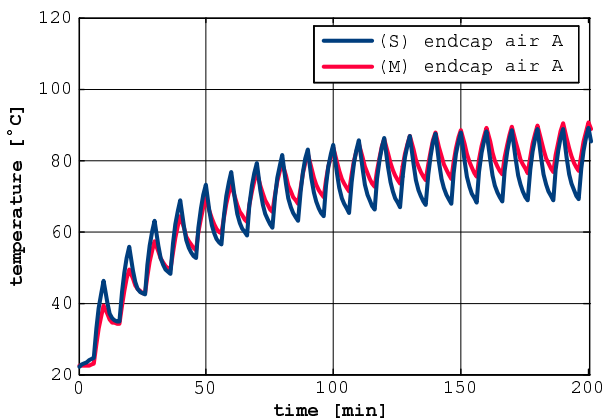


Figure 10: Simulated (S) and measured (M) temperatures of end cap, A-side (AIA)

`Modelica.Thermal.HeatTransfer`. This package does not cover the mechanisms of heat transport through a coolant flow, though. Therefore, *SimpleFlow*-library was developed, which is intended to handle applications like cooling of electric machines in a simple way. Other applications which do not fulfill the assumptions of *SimpleFlow*-library have to use the more complex `Modelica.Fluid` and `Modelica.Media` libraries. Since the application presented in the example meets the assumptions of *SimpleFlow*-library very well, the simulation results match with measurements.

The *SimpleFlow*-library is also suitable for other cooling types of electrical machines. Models for such cooling circuits (e.g. open circuit ventilated) are under test. The determination of the relevant parameters is going to be performed with a specific precalculation software which is currently developed.

Models simulating the temperature rise of electrical machines are a very important application because they lead to design optimizations and competitive advantages.

References

- [1] G. Rippar and B. Zechmeister, "Simulation of networks of heat sources (in German)," *Elin-Zeitschrift*, vol. 1, no. Heft 1, 1971.
- [2] VDI-Gesellschaft für Verfahrenstechnik und Chemieingenieurwesen, *VDI-Wärmeatlas*. Berlin: Verlag Springer, 2002.
- [3] *Dubbel Interaktiv 2.0*. Verlag Springer electronic media, 2002.
- [4] G. Merker and C. Eiglmeier, *Fluid- und Wärmetransport – Wärmeübertragung*. Stuttgart, Leipzig: B.G. Teubner, 1999.
- [5] G. Merker and C. Baumgarten, *Fluid- und Wärmetransport – Strömungslehre*. Stuttgart, Leipzig, Wiesbaden: B.G. Teubner, 2000.
- [6] H. Elmqvist, H. Tummescheit, and M. Otter, "Object-oriented modeling of thermo-fluid systems," *Modelica Conference*, 2003.

Modelling Heat Exchangers by the Finite Element Method with Grid Adaption in Modelica

Stefano Micheletti*, Simona Perotto*, Francesco Schiavo†
 Politecnico di Milano,
 P.zza Leonardo da Vinci 32
 20133 Milano, Italy

Abstract

In this paper we present a new *Modelica* model for heat exchangers, to be used within the *ThermoPower* library. The novelty of this work is a combined employment of finite elements with grid adaption.

The modelling of a generic single-phase 1-D heat exchanger is discussed, along with its approximation via the *Stabilized Galerkin/Least-Squares* method. The grid adaption procedure is first introduced from a general viewpoint and then within the *Modelica* framework. Finally, some preliminary results are shown.

1 Introduction

Heat exchangers (HEs) play a relevant role in many power-production processes, so that their accurate modelling, at least for control-oriented analysis, is a key task for any simulation suite [13].

Accurate modelling of such devices is usually a complex task, the reason being that the control-relevant phenomena are associated with thermal dynamics described by *Partial Differential Equations* (PDEs). On the other hand, different complexity levels of representation may be necessary, depending on the specific simulation experiment to be performed.

Within this framework, the power-plant modelling library *ThermoPower* [5] exploits the *Modelica* language modularity features, offering to the users several interchangeable component models, with varying levels of detail.

As for the HEs, the models currently provided are differentiated by the numerical scheme employed for the

PDEs discretization, adopting either a finite volume method (FVM) or a finite element method (FEM), with different strategies for single-phase or two-phase fluid flow [5]. Furthermore, a moving-boundary evaporator model has been recently added to the library.

In this paper we present a new model for single-phase HEs, based on the use of the finite element method with grid adaption. The objectives of this work are twofold: to develop a new HE model with high accuracy and reduced computational complexity and to show how complex mathematical techniques can be successfully used in *Modelica* for the modelling of distributed-parameters physical systems.

The proposed model is an improvement of the actual FEM model [6], obtained by a *grid adaption* technique: the grid nodes (i.e., the points where the solution is computed) change their positions so as to adapt dynamically to the solution variations. Such model can significantly improve the modelling accuracy, by removing the non-physical solution oscillations observed for the actual FEM model, whilst using fewer nodes and containing the computational burden.

The paper is organized as follows: in Section 2.1 we recall the modelling of a generic single-phase 1-D heat exchanger, while in Section 2.2 we discuss its approximation via the *Stabilized Galerkin/Least-Squares* method. In the third section the grid adaption problem is introduced from a general viewpoint, while in Section 4 we address the moving mesh method on which the *Modelica* implementation, analyzed in Section 5, is based. Some preliminary numerical results are provided in Section 6. Finally, the last section draws some conclusions and outlines possible future developments.

*MOX, Dipartimento di Matematica "F. Brioschi", {stefano.micheletti,simona.perotto}@mate.polimi.it

†Corresponding author, Dipartimento di Elettronica e Informazione, francesco.schiavo@elet.polimi.it

2 The Heat Exchanger Model

In the context of object-oriented modelling, it is convenient to split the model of a generic heat exchanger (HE) into several interacting parts, belonging to three different classes [5]: the model of the fluid within a given volume, the model of the metal walls enclosing the fluid and the model of the heat transfer between the fluid and the metal, or between the metal and the outer world. In this paper, we focus on the modelling of the first class. We improve the framework proposed in [6] by introducing suitable grid adaption techniques.

The model presented in this paper can represent single-phase HEs, which constitute a significative part of the industrial applications (e.g., the primary side of a Pressurized Water Reactor nuclear power plant [3]). However, also two-phase flows could be handled as well.

2.1 The Fluid Model

Let us deal with a compressible fluid within a pipe-shaped volume V with a rigid boundary wall, exchanging mass and energy through the inlet and outlet flanges, and thermal energy through the lateral surface. We assume that

- the longitudinal dimension x is far more relevant than the other two;
- the volume V is “sufficiently” regular (i.e., the cross-sectional area is uniform and V is such that the fluid motion along x is not interrupted);
- there are no phase-changes (that is the fluid is always either single-phase or two-phase);
- the Reynolds number Re is such that turbulent flow conditions are assured along all the pipe, which in turn guarantees almost uniform velocity and thermodynamic state of the fluid across the radial direction.

Notice that, when water or steam is assumed as the working fluid, the last hypothesis does not hold at very low flow rates (laminar flow regime). However, in practice, most industrial processes never operate in such conditions.

Under the hypotheses above it is possible to define all the thermodynamic intensive variables as functions of the longitudinal abscissa x and time t . Within this framework, the dynamic balance equations for mass,

momentum and energy can be formulated as follows:

$$A \frac{\partial \rho}{\partial t} + \frac{\partial w}{\partial x} = 0, \quad (1)$$

$$\frac{1}{A} \frac{\partial w}{\partial t} + \frac{\partial p}{\partial x} + \rho g \frac{dz}{dx} + \frac{C_f \omega}{2\rho A^3} w|w| = 0, \quad (2)$$

$$\frac{\partial h}{\partial t} + \frac{w}{\rho A} \frac{\partial h}{\partial x} = \frac{1}{\rho} \frac{\partial p}{\partial t} + \frac{\omega}{\rho A} \phi_e, \quad (3)$$

where A is the pipe cross-sectional area, ρ the fluid density, w the mass flow-rate, p the fluid pressure, g the acceleration of gravity, z the pipe height, C_f the Fanning friction factor, ω the wet perimeter, h the specific enthalpy, ϕ_e the heat flux entering the pipe across the lateral surface. The fluid velocity can be defined as $u = w/(\rho A)$. Notice that in (2) and (3) we have neglected the kinetic and the diffusion term, respectively. In the case of water-steam flows it is convenient to choose the pressure and the specific enthalpy as the thermodynamic state variables, so that the expressions of the balance equations have the same form for single-phase and two-phase flows [12]: thus all the fluid properties, such as the temperature T , the density ρ and the partial derivatives $\partial\rho/\partial h$ and $\partial\rho/\partial p$ can be computed as functions of p and h .

2.2 The Approximation Procedure

In view of power generation plant modelling, the most relevant phenomenon is described by equation (3), so that the focus for the present paper is the approximation of this latter by FEM and grid adaption. Actually, the mass and momentum equations (1) and (2) describe the fast pressure and flow rate dynamics, while the energy one (3) describes the slower dynamics of heat transport by the fluid velocity. These faster modes are typically not taken into account in HEs modelling [6]. In particular, note that, assuming the pressure p uniform along x (with possible jumps at the HE boundary) and neglecting the inertial term $\partial w/\partial t$ in (2), the integration of the mass and momentum balance equations (1) and (2) is reduced to

$$w_{in} - w_{out} = A \int_0^L \frac{\partial \rho}{\partial t} dx, \quad (4)$$

$$p_{in} - p_{out} = \Delta p_F + \Delta p_H, \quad (5)$$

where w_{in} , w_{out} , p_{in} , and p_{out} are the mass flow-rate and pressure at the HE inlet and outlet, while Δp_F and Δp_H are the pressure drops due to friction and fluid head, respectively. For further details on the approximation for equation (1) and (2) we refer to [6].

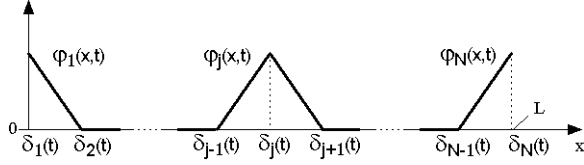


Figure 1: Some typical hat functions

Equation (3) is discretized with the stabilized *Petrov-Galerkin method GALS (Galerkin/Least-Squares)*, using suitable Dirichlet weak boundary conditions at the inflow [11].

We refer to [6] for further details about the application of the *GALS* method to heat exchangers.

In the following we provide some details about the approximation procedure by means of piecewise linear finite elements of equation (3), while referring to [16] for an exhaustive coverage of the finite element approximation theory.

We remark that we generalize the standard *GALS* method to the case of time-dependent shape and test functions, since, using the grid adaption strategy, the length of each mesh element varies in time.

Let the spatial domain $[0, L]$ be subdivided into $N - 1$ elements identified by N (≥ 3) nodes. The length of the i -th element is denoted as $\ell_i(t)$, while the abscissa of the i -th node is indicated in the sequel with $\delta_i(t)$.

On this partition we introduce the space of the piecewise linear functions, whose typical basis (hat) functions are shown in Fig. 1.

Their analytical expressions are the following:

$$\begin{aligned} \varphi_1(x, t) &= \begin{cases} \frac{\delta_2(t) - x}{\ell_1(t)} & 0 \leq x \leq \delta_1(t), \\ 0 & \text{otherwise,} \end{cases} \\ \varphi_N(x, t) &= \begin{cases} \frac{x - \delta_{N-1}(t)}{\ell_{N-1}(t)} & \delta_{N-1}(t) < x \leq L, \\ 0 & \text{otherwise,} \end{cases} \\ \varphi_i(x, t) &= \begin{cases} \frac{x - \delta_{i-1}(t)}{\ell_{i-1}(t)} & \delta_{i-1}(t) < x \leq \delta_i(t), \\ \frac{\delta_i(t) - x}{\ell_i(t)} & \delta_i(t) < x \leq \delta_{i+1}(t), \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \quad (6)$$

with $i = 2, \dots, N - 1$ and where

$$\delta_i(t) = \sum_{j=1}^{i-1} \ell_j(t), \text{ for } i = 1 \dots N. \quad (7)$$

Notice that, in view of the grid adaption procedure, the basis functions defined in (6) are both space and time

dependent. This unavoidably leads to an increase of the number of unknowns since the displacement of the grid nodes is to be determined as well.

As for the test functions involved in the *GALS* method, they are defined by

$$\psi_i(x, t) = \varphi_i(x, t) \pm \frac{\alpha}{2} \frac{\partial \varphi_i(x, t)}{\partial x}, \quad (8)$$

where α ($0 \leq \alpha \leq 1$) is a stabilization coefficient. Notice that for $\alpha = 0$ the standard (i.e., non-stabilized) method is obtained.

For the reader's ease, we provide also the expression of the time derivative $\dot{\varphi}_i = \partial \varphi_i(x, t) / \partial t$ of the basis function φ_i , namely

$$\dot{\varphi}_i(x, t) = \begin{cases} \frac{-\dot{\delta}_{i-1} - (x - \delta_{i-1}) \dot{\ell}_{i-1}}{\ell_{i-1}^2} & \delta_{i-1} < x \leq \delta_i, \\ \frac{\dot{\delta}_{i+1} - (\delta_{i+1} - x) \dot{\ell}_i}{\ell_i^2} & \delta_i < x \leq \delta_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

Let us expand the quantities h , ρ , w , ϕ_e in terms of the basis functions φ_i as:

$$\begin{aligned} h(x, t) &= \sum_{i=1}^N h_i(t) \varphi_i(x, t) = \bar{h}(t)^T \bar{\varphi}(x, t), \quad \bar{h} = [h_1 \dots h_N]^T, \\ \rho(x, t) &= \sum_{i=1}^N \rho_i(t) \varphi_i(x, t) = \bar{\rho}(t)^T \bar{\varphi}(x, t), \quad \bar{\rho} = [\rho_1 \dots \rho_N]^T, \\ w(x, t) &= \sum_{i=1}^N w_i(t) \varphi_i(x, t) = \bar{w}(t)^T \bar{\varphi}(x, t), \quad \bar{w} = [w_1 \dots w_N]^T, \\ \phi_e(x, t) &= \sum_{i=1}^N \phi_i(t) \varphi_i(x, t) = \bar{\phi}(t)^T \bar{\varphi}(x, t), \quad \bar{\phi} = [\phi_1 \dots \phi_N]^T, \end{aligned} \quad (10)$$

with $\bar{\varphi}(x, t) = [\varphi_1(x, t), \dots, \varphi_N(x, t)]^T$.

Applying the *GALS* finite element method to (3) leads to the following set of N ODEs:

$$\begin{aligned} &\int_0^L \left(\sum_{i=1}^N \dot{h}_i \varphi_i \right) \psi_j dx + \int_0^L \left(\sum_{i=1}^N h_i \dot{\varphi}_i \right) \psi_j dx + \\ &\int_0^L \left(\frac{\sum_{i=1}^N w_i \varphi_i}{A \sum_{i=1}^N \rho_i \varphi_i} \sum_{i=1}^N h_i \frac{d\varphi_i}{dx} \right) \psi_j dx + \\ &\int_{\partial\Omega^{in}} \left(\frac{\sum_{i=1}^N w_i \varphi_i}{A \sum_{i=1}^N \rho_i \varphi_i} \sum_{i=1}^N h_i \varphi_i \right) \psi_j dx = \\ &\int_0^L \frac{\dot{p}}{\sum_{i=1}^N \rho_i \varphi_i} \psi_j dx + \int_0^L \left(\frac{\omega \sum_{i=1}^N \phi_i \varphi_i}{A \sum_{i=1}^N \rho_i \varphi_i} \right) \psi_j dx + \\ &\int_{\partial\Omega^{in}} \left(\frac{\sum_{i=1}^N w_i \varphi_i}{A \sum_{i=1}^N \rho_i \varphi_i} h_{in} \right) \psi_j dx, \quad \forall \psi_j \text{ with } j = 1, \dots, N, \end{aligned} \quad (11)$$

where h_{in} is the fluid specific enthalpy at the inflow boundary $\partial\Omega^{in}$. Such set of ODEs can be represented by the following compact matrix notation:

$$M\dot{\bar{h}} + M_D\bar{h} + \frac{1}{A}F\bar{h} + \frac{1}{A}C\bar{h} = R\dot{p} + \frac{\omega}{A}Y\bar{\phi} + \frac{1}{A}K\bar{w}, \quad (12)$$

where M, M_D, F, C, R, Y, K are defined as follows:

$$\begin{aligned} M_{ji} &= \int_0^L \phi_i \psi_j dx, & M_{Dji} &= \int_0^L \phi_i \psi_j dx, \\ F_{ji} &= \int_0^L \frac{\sum_{k=1}^N w_k \phi_k}{\sum_{k=1}^N \rho_k \phi_k} \frac{d\phi_i}{dx} \psi_j dx, \\ C_{ji} &= \int_{\partial\Omega^{in}} \frac{\sum_{k=1}^N w_k \phi_k}{\sum_{k=1}^N \rho_k \phi_k} \phi_i \psi_j dx, \\ R_j &= \int_0^L \frac{\psi_j}{\sum_{k=1}^N \rho_k \phi_k} dx, & Y_{ji} &= \int_0^L \frac{\phi_i}{\sum_{k=1}^N \rho_k \phi_k} \psi_j dx, \\ K_{ji} &= \int_{\partial\Omega^{in}} \frac{h_{in}}{\sum_{k=1}^N \rho_k \phi_k} \phi_i \psi_j dx. \end{aligned} \quad (13)$$

The matrices C and K , which enforce the boundary conditions into equation (12), depend on the inflow boundary $\partial\Omega^{in}$. It can be noted that, as we are considering the 1-D case, the inflow boundary is constituted, at most, by the points $x = 0$ and $x = L$, depending on the sign of $w = w_{in}$. Thus the only test functions that are non-zero at the inflow are ψ_1 and ψ_N and the only non-vanishing entries of the matrices C and K are

$$\begin{aligned} C_{11} &= \begin{cases} \frac{w_1}{\rho_1} \left(1 - \frac{\alpha}{2}\right) & w|_{x=0} > 0, \\ 0 & otherwise, \end{cases} \\ C_{NN} &= \begin{cases} \frac{w_N}{\rho_N} \left(1 + \frac{\alpha}{2}\right) & w|_{x=L} > 0, \\ 0 & otherwise, \end{cases} \\ K_{11} &= \begin{cases} \frac{h_{in}|_{x=0}}{\rho_1} \left(1 - \frac{\alpha}{2}\right) & w|_{x=0} > 0, \\ 0 & otherwise, \end{cases} \\ K_{NN} &= \begin{cases} \frac{h_{in}|_{x=L}}{\rho_N} \left(1 + \frac{\alpha}{2}\right) & w|_{x=L} > 0, \\ 0 & otherwise. \end{cases} \end{aligned} \quad (14)$$

The matrices C and K are consequently diagonal.

3 The Grid Adaption Philosophy

The discretization of complex phenomena described by systems of partial differential equations by means

of FEM, can be cast into the framework of *model reduction*, i.e., the approximation by a finite dimensional model of a conceptually infinite dimensional one. Several parameters (e.g., the mesh spacing, the degree of the polynomial finite elements, tuning parameters related to the discretization procedure) govern the accuracy of the approximation. As an effective tool to assess such approximation property, some estimators/indicators, as the local cell residual, are typically employed [1, 9, 18]. Once the error indicator has been computed on a given mesh, the information that it contains can be used to generate a better mesh that gives more accuracy. This is the basis of *adaptive error control*.

Many engineering problems are characterized by solutions exhibiting a complex structure, e.g., singularities near corners, boundary layers or shocks. In such cases, the idea is to distribute the mesh spacings according to local features of the solution, that is to concentrate the elements in the regions where the solution changes rapidly and, vice versa, to coarsen them where the solution is smoother, with the aim of obtaining a solution sufficiently accurate and with a reasonable computational load.

Typically an adaptive error control procedure consists of a discretization method combined with an adaptive algorithm. There are three main types of adaptive techniques for FEM: i) the *h-method*: the mesh is refined and coarsened locally according to certain *error estimators*; ii) the *p-method*: the polynomial degree is chosen in each element according to some *smoothness indicator*; iii) the *r-method*: the element vertices are relocated to concentrate them in desired regions on the basis of a *monitor function*.

In the following we focus on this last philosophy which is usually referred to as *moving mesh method* [7, 10, 14, 15, 17]. In this method, a mesh equation involving the nodes speed is solved to compute the mesh points location together with the solution of the differential equation at hand. In principle, starting from a given mesh, the idea is to move the mesh nodes, while keeping their number fixed, towards regions of rapid solution variations, e.g., steep wave fronts and shocks.

3.1 Grid Adaption as a Control Problem

An interesting point of view to tackle the grid adaption procedure is to state it as a control problem. As a matter of fact, the grid adaption is based on a feedback mechanism that can be represented as in Fig. 2.

Within this framework, the *process* is represented by the N ODES obtained from *GALS* discretization, the

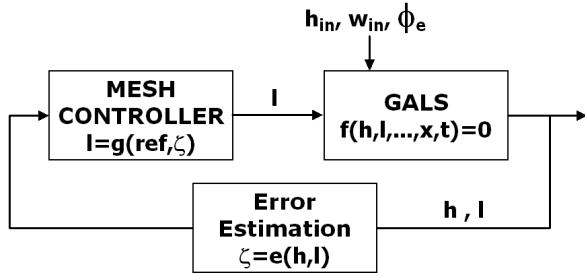


Figure 2: Grid adaption as a control problem

sensor is represented by some estimate of the discretization error and the controller is defined by the grid adaption strategy. The time-varying boundary values for the HE (h_{in} and w_{in}) and the heat flux entering its lateral surface (ϕ_e) are, from the point of view of feedback grid adaption, process disturbances, while the length of the elements (ℓ_i) can be regarded as the (vectorial) control variable ℓ .

The aim of the control system is to minimize the estimated error. In this paper we adopt the *equidistribution principle* [2] to design the controller (i.e., the mesh adaption strategy): the aim is to dynamically obtain an equidistributed error over the elements.

4 The Moving Mesh Method in Modelica

The application of the GALS method to equation (3) leads to a set of N ODEs whose unknowns are the nodal values for the fluid specific enthalpy. Moreover, due to the grid adaption strategy, we have to include other $N - 1$ unknowns, i.e., the lengths ℓ_i of the elements. The coupled equations yield the so-called DAE-system.

The mesh point positions have to be calculated in such a way that

- 1) the length of each element is strictly positive (*constitutive constraint*: $\ell_i > 0 \forall i = 1 \dots N - 1, \forall t \geq 0$);
- 2) the total length of the elements is equal to L (*completeness constraint*: $\sum_{i=1}^{N-1} \ell_i = L, \forall t \geq 0$).

These constraints can be easily fulfilled when dealing with *imperative* languages (i.e., algorithm oriented). In such a case, a specific grid adaption procedure is first allowed to yield a mesh characterized by values for the lengths ℓ_i “illegal” with respect to the criteria 1) and 2). Then a suitable refinement algorithm is used

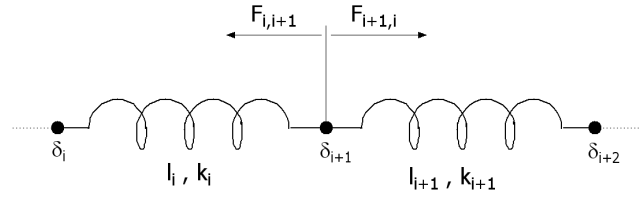


Figure 3: The spring model for grid adaption

to correct such values so that the constitutive and completeness constraints are satisfied.

On the other hand, when dealing with a *declarative* language such as *Modelica*, a different approach has to be taken: the constitutive and completeness constraints have to be intrinsically fulfilled. Such result can be easily obtained using a physical approach for the implementation of the adaption procedure.

Let us consider Fig. 3: each element can be identified with a spring of length ℓ_i and specific elastic constant k_i , with the first and the last spring fixed to the domain boundaries $x = 0$ and $x = L$, respectively.

Let $F_{i,j}$ be the force that the i -th spring exerts on the j -th one. Usually it is assumed that

$$F_{i,j} = 0 \quad \forall j \neq i - 1, i + 1, \quad (15)$$

that means that each spring interacts only with the two adjacent ones. Furthermore, the force that two adjacent springs exert on each other can be expressed as

$$F_{i,i+1} = k_i \ell_i \quad F_{i+1,i} = k_{i+1} \ell_{i+1}. \quad (16)$$

Supposing that the spring constants k_i are non-negative, an effective choice for the unknowns ℓ_i in terms of the k_i is:

$$\ell_i = \frac{k_i}{\sum_{j=1}^{N-1} k_j} L, \quad \forall i = 1 \dots N - 1. \quad (17)$$

This automatically guarantees the completeness constraint as

$$\sum_{i=1}^{N-1} \ell_i = \sum_{i=1}^{N-1} \frac{k_i}{\sum_{j=1}^{N-1} k_j} L = L. \quad (18)$$

Moreover, if all the spring constants are positive, then the constitutive constraint is fulfilled as well. It is important to notice that such strategy is independent of the particular grid adaption procedure at hand. To make effective the chosen adaption procedure it is necessary to relate the elastic constants k_i to the local monitor function ε_i , as

$$k_i = \frac{1}{\ell_i \varepsilon_i}, \quad \forall i = 1, \dots, N - 1. \quad (19)$$

The strategy we adopt aims at concentrating the grid points in the domain regions where the monitor function ε is larger. This can be justified by analyzing equations (19) and (17): the larger the monitor function, the smaller the associated spring constant and, consequently, the smaller the length of the corresponding element.

The monitor ε_i is usually defined as a function of a “residual”, identified in the sequel with the symbol ζ_i , directly related to the approximate solution obtained with the *GALS* method.

The monitor function $\varepsilon_i = \varepsilon_i(\zeta_i)$ can be chosen arbitrarily, provided that it is definite positive, though it is much more effective when it monotonous as well.

One of the most used monitor function sharing these properties is the so-called *arclength* [4], given by

$$\varepsilon_i = \sqrt{1 + \mu \zeta_i^2}, \quad (20)$$

where μ is a positive coefficient used to “tune” the grid adaption.

In [7] it is shown that this choice yields good results when applied to transport equations.

Another example of monitor function, successfully used in [10], is the *curvature* monitor function, given by

$$\varepsilon_i = \sqrt[4]{1 + \mu \zeta_i^2}. \quad (21)$$

Using the *arclength* or the *curvature* monitor function, particular care has to be taken in the choice of the parameter μ , since it is a sort of “gain” of the mesh controller: the larger μ , the faster the grid adaption becomes (see Fig. 2). The value of such parameter can either be fixed or tuned by the user. In this latter case, lower and upper bounds for μ should be provided, since a low value can make the adaption mechanism too weak and then useless, while a too large value can negatively affect the numerical stability of the adaption algorithm.

The tuning of the parameter μ becomes even more critical when using a fixed time-step explicit method to solve the resulting non-linear DAE system, which is often the case when simulating industrial plants in connection with the control system [3]. Such sensitivity depends on the fact that, somehow, the parameter μ regulates how “fast” the grid adaption is: a large value makes the adaption too fast, thus introducing dynamics with time constants significantly smaller compared with the fixed time step, resulting in a numerical instability.

The last step to complete the grid adaption scheme is the definition of the residual ζ_i over the elements.

4.1 Definition of the Residual

The residual definition is a key choice in the grid adaption framework. When using the *arclength* monitor function, a common choice for the residual is the approximate gradient:

$$\zeta_i = \frac{h_{i+1} - h_i}{\ell_i} \approx \frac{\partial h}{\partial x}, \quad \forall i = 1, \dots, N-1. \quad (22)$$

This choice aims at concentrating the grid points within the regions where large solution variations occur. This implicitly assumes that the discretization error is large in such areas.

However, in case of problems with a “sharp-but-not-steep” solution, it has been shown that the *arclength* monitor function with approximate gradient given by (22) performs poorly (see [10]). In such a case, a better approximation can be obtained using the *curvature* monitor function (21) with a second order approximation of the 2nd order spatial derivative:

$$\zeta_i = \frac{h_{i+1} - 2h_i + h_{i-1}}{\ell_i^2} \approx \frac{\partial^2 h}{\partial x^2}, \quad \forall i = 1, \dots, N-1, \quad (23)$$

where it is understood that $h_0 = h_{in}$.

In [14], it is shown that, for problems involving more than one moving front in the solution, the use of the *curvature* monitor function can lead to better results than the use of the *arclength* one.

In this paper we show results obtained with grid adaption based on these two residual definitions and monitor functions.

5 Modelica Implementation

The developed model has been implemented in a *Modelica* component called `Flow1DfemAdapt` which is going to be included within the library *ThermoPower* [5]. The component is perfectly interchangeable with the actual library components for 1-D HEs, since it uses the same connectors: two flanges for fluid flow and a terminal for heat flux (Fig. 4).

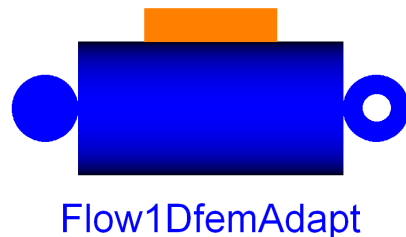


Figure 4: Component Icon

The *Modelica* implementation is quite close to the one presented in [6] with some difference in the energy equation and completed with the equations for the grid adaption.

The discretized energy equation contains a new term:

$$M \cdot \text{der}(h) + (M_D + F/A) \cdot h + C/A \cdot h = R \cdot \text{der}(p) + Y \cdot \omega_{\text{grid}} / A \cdot \phi + K/A \cdot w;$$

where the additional tridiagonal matrix M_D is coded with nested “for” loops as shown in [6].

The selection of the residual and of the corresponding monitor function depends by the user via the integer parameter *Residual*:

```

if Residual==1 then
  for i in 1:N - 1 loop
    res[i] = (h[i+1] - h[i])/l[i];
    err[i] = sqrt(1+mu*res[i]^2);
  end for;
else
  res[1]=(h[1 + 1] - 2*h[1]+h_in)/l[1]^2;
  err[1] = (1+mu*res[1]^2)^0.25;
  for i in 2:N - 1 loop
    res[i] = (h[i+1]-2*h[i]+h[i-1])/l[i]^2;
    err[i] = (1+mu*res[i]^2)^0.25;
  end for;
end if;
end if;
    
```

Finally, the length of the elements is obtained solving the following $N - 1$ algebraic equations:

```

for i in 1:(N - 1) loop
  k[i] = 1/(err[i]*l[i]);
  l[i] = k[i]/sum(k)*L;
end for;
    
```

6 Simulations

In this section we show simulation results in order to evaluate the different performances of the grid adaption strategies. All the simulations have been performed within the Dymola [8] simulation environment.

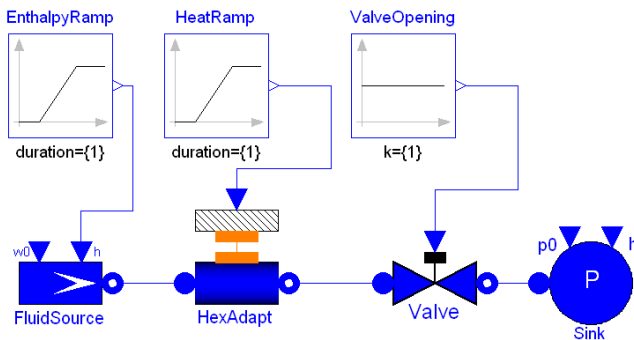


Figure 5: Reference Simulation Layout

The reference simulation layout is shown in Fig. 5, consisting in an ideal flow source connected to a HE

which is followed by a valve and by an ideal pressure sink. An ideal heat-flux source is connected to the HE distributed heat-flux terminal. Such setup has been selected in order to highlight the differences of the approximation schemes on the HE outlet specific enthalpy.

The HE internal pressure is held constant since the mass flow-rate and the valve opening are set to a fixed value and the sink pressure is constant as well. Thus, supposing the specific enthalpy of the fluid within the HE does not vary substantially, it is possible to assume that the fluid density is almost constant.

In case the heat-flux is set to zero as well, it is possible to show that the analytical solution for the transport equation (3) is a ramp-wave travelling along the HE with constant velocity u . It is then possible to evaluate the model approximation performances with an *a-posteriori* error indicator, evaluating the square deviation

$$E(x) = \int_0^t (\hat{h}(x,t) - h(x,t))^2 dt, \quad (24)$$

of the approximate solution h from the analytical one \hat{h} .

The indicator E is spatially distributed, so we extract from it two different indicators:

$$IE = \int_0^L E(x) dx, \quad (25)$$

$$OE = E(x)|_{x=L},$$

denoting the *integral error* (IE) and the *output error* (OE).

For the sake of approximation, as we compute the square deviation $E(x)$ at the grid points only, the indicator IE is evaluated via a linear piecewise interpolation.

The numerical data employed for the HEs modelling are the length $L = 10m$ and the cross-sectional area $A = 3.14 \cdot 10^{-4} m^2$. The heat-flux ϕ_e is set to zero. The fluid entering the HE is liquid water at pressure $p = 10^5 Pa$, with initial specific enthalpy $h_{in} = 10^5 J/Kg$ and flow rate $w_{in} = 1 Kg/s$. Thus, the transit time turns out to be $31.25 s$.

The time-integration of the system is performed with a fourth order *Runge-Kutta* scheme with a fixed time step $T_s = 0.1 s$. The chosen time step turns out to be adequate for the simulations of the dynamics represented by (3) [3].

The first test case aims at checking the effectiveness of the grid adaption strategy when abrupt changes of the solution are involved.

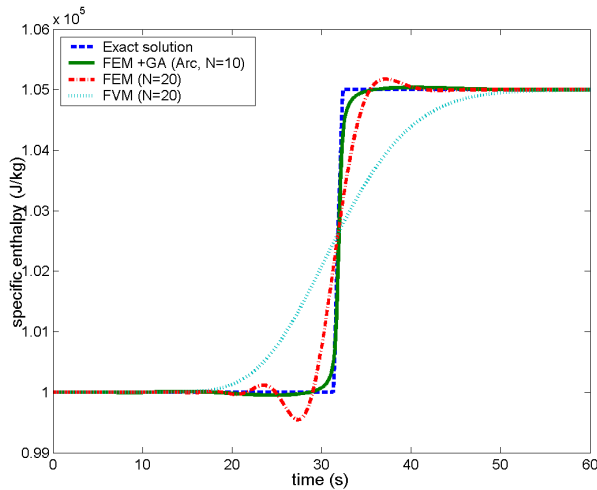


Figure 6: Approximate enthalpy provided by three different numerical schemes and exact enthalpy

The time interval of the simulation is chosen as $[0, 60]s$. The inflow enthalpy h_{in} is described by a ramp function with a rising time of 1s starting at 1s. The corresponding increment of the enthalpy is of the 5% of the initial value.

In Fig. 6 the HE outlet specific enthalpy associated with three different numerical schemes is compared with the exact solution (blue line). In particular the cyan, the red and the green lines correspond to the finite volumes (20 nodes), finite elements (20 nodes) and finite elements with grid adaption based on the *arclength* monitor function (10 nodes). The “gain” of the mesh controller has been set to the value $\mu = 3.5 \cdot 10^{-4}$ after several simulations as a trade-off between accuracy and numerical stability.

Fig. 6 shows that grid adaption with *arclength* monitor function can significantly improve the quality of the approximation. On the other hand, to thoroughly compare the three proposed algorithms, their computational effort has to be taken into account as well, since the grid adaption procedure is not cost-free. A full comparison of the various methods is summarized in Table 1.

Method	N	CPU time	OE	IE
FVM	20	0.302s	$3.59 \cdot 10^7$	$6.98 \cdot 10^7$
FEM	20	0.356s	$7.50 \cdot 10^6$	$1.71 \cdot 10^7$
FEM	50	1.06s	$2.61 \cdot 10^6$	$1.65 \cdot 10^7$
FEM+GA	10	0.579s	$4.64 \cdot 10^5$	$8.39 \cdot 10^6$
FEM+GA	15	1.109s	$4.37 \cdot 10^5$	$2.43 \cdot 10^6$

Table 1: CPU time, output and integral error

Table 1 clearly shows that, for the case at hand, the solution obtained with the proposed grid adaption strategy with relatively few nodes ($N = 10$) is a far better approximation of the exact solution (at least in term of the indicators *IE* and *OE*) than the ones obtained with FVM and FEM with a number of nodes $N = 20$ or $N = 50$. However, the computational overhead due to grid adaption is not negligible, as highlighted by the CPU time column.

The results show that the use of the proposed grid adaption strategy is convenient when the demand on the accuracy of the solution is relatively strong. This can be obtained by a small number of mesh nodes though the CPU time can increase. Alternatively, standard FVM or FEM can be employed but a higher number of nodes is required to obtain the same level of accuracy.

The second test case shows that the good results obtained with grid adaption using the *arclength* monitor function do not hold when the *curvature* monitor function is used, as can be seen by the curves in Fig. 7. The simulation time interval is now chosen as $[0, 80]s$. The inflow enthalpy h_{in} is represented by a function characterized by three stages: a raising ramp from $t = 1s$ to $t = 2s$, a plateau during 18 seconds, a decreasing ramp from $t = 20s$ to $t = 21s$. The net increment of the enthalpy is the 5% of the initial value, while the final value coincides with the initial one. The “gain” of the mesh controller has been chosen equal to $3.5 \cdot 10^{-4}$ and $3.5 \cdot 10^{-8}$ for the *arclength* and the *curvature* monitor function, respectively. Larger values of μ for the curvature choice lead to numerical instabilities.

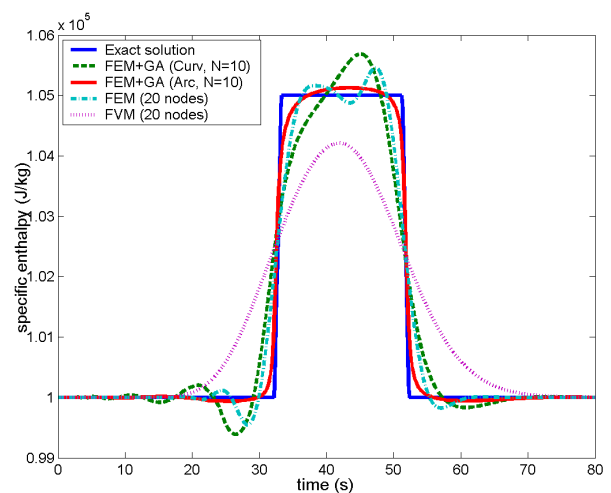


Figure 7: Approximate enthalpy provided by four different numerical schemes and exact enthalpy

The CPU time and the values of IE and OE are gathered in Table 2.

While for the *arclength* monitor function similar considerations as in the previous test case hold, we note that the technique based on the *curvature* monitor function does not introduce significant benefits, concerning both the CPU time and the accuracy (compared, for instance, with the FEM case with $N = 20$).

Method	N	CPU time	OE	IE
FVM	20	0.515 s	$7.55 \cdot 10^7$	$1.41 \cdot 10^8$
FEM	20	0.546 s	$1.51 \cdot 10^7$	$7.97 \cdot 10^7$
FEM+GA*	10	0.622 s	$2.72 \cdot 10^6$	$3.44 \cdot 10^7$
FEM+GA†	10	0.719 s	$2.54 \cdot 10^7$	$1.79 \cdot 10^8$

* *Arclength* monitor function

† *Curvature* monitor function

Table 2: CPU time, output and integral error

In the last test case we study the effect of the grid adaption on the approximate solution under a sudden cooling of the lateral surface of the HE. The time interval is $[0, 120]$ s. The inflow enthalpy h_{in} is the same as in the first case, while at $t = 60$ s the heat-flux is decreased with a step variation to $\phi_e = -795 \text{ W/m}^2$, i.e. 500 W are lost through the lateral surface of the HE.

Assuming that the fluid density is approximately constant, the exact solution for the outlet enthalpy is the delayed inlet increasing ramp followed by a decreasing ramp starting at $t = 60$ s.

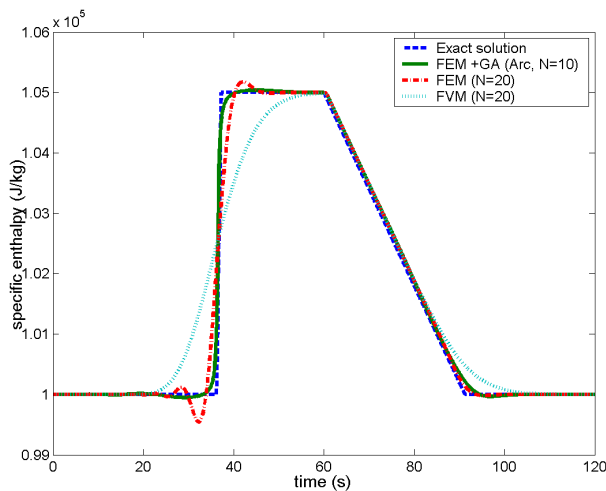


Figure 8: Effect of a heat-flux decrease (adaption with *arclength* monitor function)

In Fig. 8 the approximate solution of the three schemes FVM ($N = 20$), FEM ($N = 20$), FEM+GA ($N = 10$, *arclength* monitor function with $\mu = 3.5 \cdot 10^{-4}$) is provided together with the exact HE outlet enthalpy.

It turns out that grid adaption significantly improves the quality of the solution with respect to FEM or FVM when abrupt changes are involved, while the difference is less evident where the solution is smooth.

7 Conclusions and Future Work

In this paper we present a new model for 1-D single-phase heat exchangers in *Modelica*. The model, fully compatible with the ones already available within the library *ThermoPower*, is based on an approximation of the energy balance equation by the *GALS* finite element method with grid adaption.

The mathematical model and its approximation have been addressed in detail, as well as the grid adaption strategy within the a-causal framework *Modelica*.

The effectiveness of the proposed technique has been assessed on some test cases and compared with the standard FV and FE methods. The main conclusion is that grid adaption turns out to be effective when high accuracy is required. In more detail, even if the “placement” of each mesh node is more expensive in terms of CPU time, a smaller number of nodes is required to guarantee a certain level of accuracy, compared with FVM and FEM.

Future work will be devoted to a more theoretically sound selection of the optimal value for the “gain” μ of the mesh controller. Moreover, the employment of a dynamical residual will be further investigated.

References

- [1] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica*, 10:1–102, 2001.
- [2] C. Boor. *Good Approximation by splines with variable knots*. Springer Lecture Notes Series 363. Springer Verlag, 1973.
- [3] A. Cammi, F. Casella, M.E. Ricotti, F. Schiavo, and G.D. Storrck. Object-oriented simulation for the control of the IRIS nuclear power plant. In *16th IFAC world congress*, Prague, Czech Republic, July 4-8, 2005.
- [4] W. Cao, W. Huang, and R.D. Russel. An *r*-adaptive finite element method based upon moving mesh PDEs. *Journal of Computational Physics*, 149(2):221–244, March 1999.

- [5] F. Casella and A. Leva. Modelica open library for power plant simulation: Design and experimental validation. In *3rd Modelica Conference*, Linköping, Sweden, November 3-4, 2003. <http://sourceforge.net/projects/thermopower>.
- [6] F. Casella and F. Schiavo. Modelling and simulation of heat exchangers in Modelica with Finite Element Methods. In *3rd Modelica Conference*, Linköping, Sweden, November 3-4, 2003.
- [7] E.A. Dorfi and L.O.'C. Drury. Simple adaptive grids for 1-D initial value problems. *Journal of Computational Physics*, 69:175–195, 1987.
- [8] Dymola. *Dynamic Modelling Laboratory*. Dymasim AB, Lund, Sweden.
- [9] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. Introduction to adaptive methods for differential equations. *Acta Numerica*, pages 105–158, 1995.
- [10] W. Huang, Y. Ren, and R.D. Russell. Moving mesh methods based on moving mesh partial differential equations. *Journal of Computational Physics*, 113:279–290, 1994.
- [11] T.J.R. Hughes, L.P. Franca, and G.H. Hulbert. A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/Least-Squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73(2):173–189, 1989.
- [12] F.P. Incropera and D.P. DeWitt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 1985.
- [13] A. Leva and C. Maffezzoni. Modelling of power plants. In D. Flynn, editor, *Thermal Power Plant Simulation and Control*, pages 17–60, London, 2003. IEE.
- [14] Y. Liu. *On Model Order Reduction of Distributed Parameters Models*. Licentiate Thesis, Royal Institute of Technology, Stockholm, 2002.
- [15] K.K. Miller and R.N. Miller. Moving finite elements, I & II. *SIAM Journal on Numerical Analysis*, 18(6):1019–1032, 1033–1057, 1981.
- [16] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer Verlag, 1997.
- [17] C. Sereno, A. Rodrigues, and J. Villadsen. The moving finite element method with polynomial approximation of any degree. *Computers & Chemical Engineering*, 15(1):25–33, 1991.
- [18] R. Verfürth. *A review of a posteriori error estimation and adaptive mesh-refinement techniques*. B.G. Teubner, 1996.

Calculation of Thermophysical Properties in the Modelica Library TechThermo

Wolf-Dieter Steinmann
 German Aerospace Center
 Institute of Technical Thermodynamics
 Pfaffenwaldring 38-40, 70569 Stuttgart
 wolf.steinmann@dlr.de>

Abstract

Many physical models describing thermodynamic systems require correlations for thermophysical properties to complete the set of equations. A Modelica library like TechThermo which is intended for general application in technical thermodynamics must include also a set of models for calculation of thermophysical properties. Since the total number of models in TechThermo should be limited, the range of application for selected model must be as wide as possible. The models for calculation of thermophysical properties in TechThermo are based on general concepts which allow the introduction of new working media by modification of a few model parameters. Models for calculation of multiphase or multi-component media are composed of models representing single phase behaviour and mixing models.

Keywords: thermodynamic system; thermophysical properties;

1 The TechThermo library

1.1 Aims of TechThermo

TechThermo is a basic library for engineering applications in thermodynamics. This basic library provides components which are relevant for the bulk of applications in this area. TechThermo is complemented by problem-specific libraries which include components relevant only for a limited scope of application (Figure1). There are different aspects how TechThermo can improve the efficiency of modeling activities:

- the library should allow a fast implementation of a model describing the simulation problem without generation of additional source code the detailing of the system may be limited, but this approach allows fast results and demands only limited knowledge of Modelica or numerical principles

- Experienced Modelica users should profit from TechThermo primarily by extending the models provided by the library thus minimizing the extent of work spent on implementation of trivial equations needed for describing a physical process.
- by standardization of interfaces the cooperation between model-developers is improved.

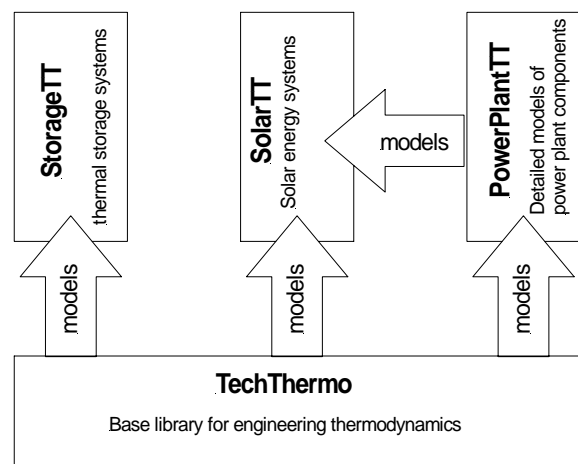


Figure 1: Application of basic library TechThermo in combination with problem-specific libraries

Although it would be comfortable to have a universal thermodynamic library which allows the modelling of any system by mere composition of basic models without input of further model equations, this approach was not chosen for TechThermo since the implementation of such a library seems not to be feasible in practice. Instead, the aim of TechThermo is to minimize the effort for supplementary models for a wide range of application.

1.2 Structure of TechThermo

Modelica-infrastructure:	
Interface	- definition of model interfaces
Source	- models for imposing boundary conditions
Fundamentals of engineering thermodynamics:	
Medium	- thermophysical correlations for working media
Basis	- descriptions of fundamental physical processes
Fundamentals of technical systems:	
Component	- models of basic technical units
Subsystem	- simplified models of technical systems

Tab1: The structure of TechThermo

The models of TechThermo are organized in six main packages and an additional folder with examples. These six packages can be attributed to one of three different groups:

- package Interface and package Source include the models which are needed for the exchange of information between connected models and models for imposing boundary conditions. These two packages can be regarded as the infrastructure needed in any Modelica simulation
- package Medium and package Basis comprise models describing fundamental processes in thermodynamics and correlations for thermophysical properties of working media. These two packages can be regarded as the infrastructure needed for modelling thermodynamic processes.
- package Component and package Subsystem contain models describing basic technical components like turbines and

heat exchangers and models representing simplified systems like solar collectors or fuel cells. These two packages can be regarded as infrastructure for modelling technical systems.

The six main packages are stored in separate files; a subset of packages can be selected provided the hierarchy is regarded (Tab. 1): while package Interface demands no models from the other five packages, package Source demands package Interface, Medium demands Source and Interface and so on.

2 Models including correlations for thermophysical properties

2.1 Relevance of correlations between variables describing thermodynamic systems

Thermodynamic systems can be described by various variables. The most common state variables for technical systems are

- spec. enthalpy
- pressure
- density
- spec. entropy
- temperature
- spec. internal energy
- steam quality
- vector with mass fractions

The minimal number of independent variables needed to define the state of a system depends on the number of phases and components, for a pure substance only two variables are needed. Depending on the physical process various combinations of variables may be used in a model. Mathematical correlations between state variables are needed if different sets of variables are used. These correlations must ensure that all sets of variables define the identical thermal state.

2.2 Representation of thermal state information in TechThermo

The connectors in the TechThermo library transfer information between models as energy flows. Three different kinds of energy flows are defined:

- combined heat and mass transfer

- heat transfer (without mass transfer), defined by heat flow rate \dot{q} and temperature t
- pure exergy, defined by exergy flow rate \dot{ex}

While the selection of connector variables defining heat transfer and pure exergy flow is obvious, various sets of variables listed in section 2.1 can be used. The connector for combined heat and mass transfer in TechThermo is defined by

- mass flow rate \dot{m}
- pressure p
- spec. enthalpy h
- composition vector $x_i[n_{comp}]$

Pressure and spec. enthalpy represent a minimal set of state variables. If models include physical processes described by other variables, these variables must be calculated from pressure and spec. enthalpy. One basic concept of TechThermo is the separation of models including correlations for thermophysical properties from the other parts of a model. A fourth connector for thermal state information is defined including all variables listed in section 2.1. This concept should be demonstrated here by the model `AirCompressor` representing an adiabatic compressor for air. The physical process is shown in Figure 2. Assuming an isentropic efficiency $\eta_{isentrop}$ the spec. enthalpy h_{out} at the outlet is

$$h_{out} = h_{in} + (State2.h - h_{in}) / \eta_{isentrop}$$

Variable `State2.h` is the spec. enthalpy after an isentropic compression from pressure p_{in} to pressure p_{out} . The entropy after the isentropic compression is identical to the entropy for inlet enthalpy h_{in} and pressure p_{in} . The model demands two thermophysical correlations:

1. spec. entropy s_{in} at inlet enthalpy h_{in} and pressure p_{in}
2. spec. enthalpy `State2.h` for entropy s_{in} and pressure p_{out} .

Figure 3 shows the diagram layer of `AirCompressor`. The model `CompressorNoProp1` represents the compressor model without specification of the working fluid. By addition of two models `AirPerfectGasCaloric` including the thermophysical property correlations for air the model is completed. Thermal state connectors are used to link `CompressorNoProp1` and the two `AirPerfectGasCaloric` models. By exchanging the

property models the compressor can be adapted to other working fluids. `AirCompressor` also includes two models `NotUsedVariables`; these models are used to define the remaining variables of the thermal state connectors by parameters to complete the set of equations.

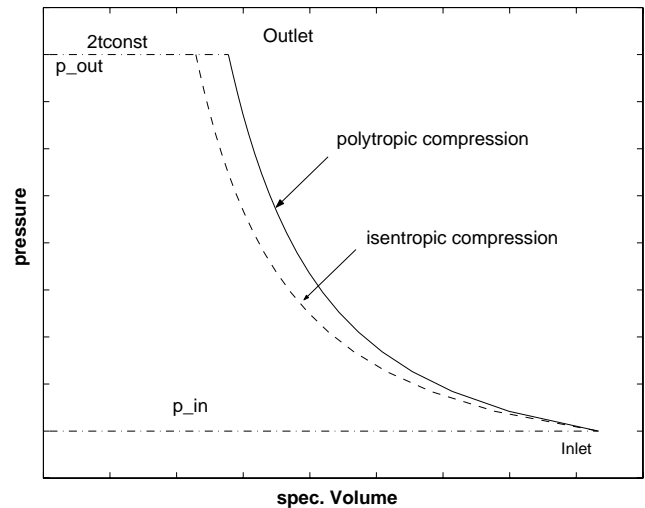


Figure 2: p-v diagram of adiabatic compression

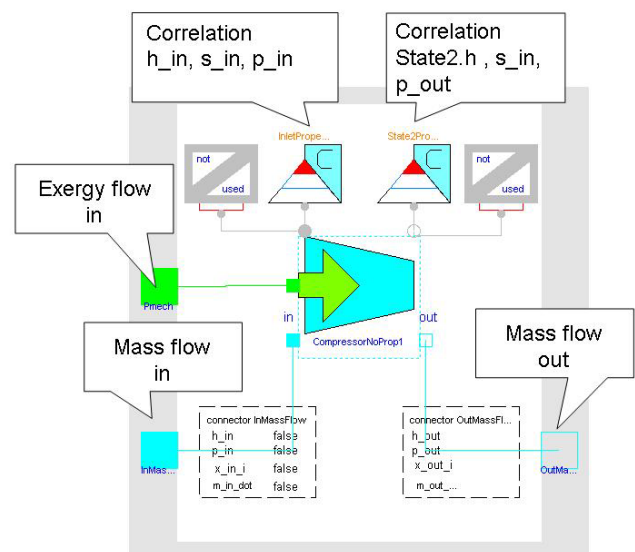


Figure 3: Diagram layer of the compressor model `AirCompressor` including two models for calculation of thermophysical properties of the working fluid.

3 The main package Medium in TechThermo

3.1 Selection of model describing property data

TechThermo is intended as a basis library for applications in technical thermodynamics. The models included should be used in a wide range of applications. Concerning the models for thermophysical correlations included in the main package Medium of TechThermo there are two basic aspects:

- only models based on general concepts are contained; these models can be adapted to various media by modification of a small number of parameters
- TechThermo does not include large multiparameter equations of state which are only available for a limited number of media.

The accuracy of general models usually is limited; if the models offered by TechThermo don't fulfil the demands, additional models are provided by problem-specific model libraries.

Mathematical models for thermodynamic property data are available within a wide range of complexity and accuracy. Criteria for selection of algorithms can be

- accuracy compared to reference values
- consistency; especially interesting near to phase transitions
- dependence on selected variables; many multiparameter equations of state can't be inverted and demand iterative solution if the set of independent variables changes
- numerical aspects like stability and required calculation time

The selection of the property models strongly influences the behaviour of the model. The choice should always be adapted to the specific simulation problem. Aspects that should be considered are

- the extend of variation for a state variable within a model; if the variation is limited, the application of simple linear property models may be sufficient without introducing significant errors; e.g. if a simulation deals with a gas at room temperature at ambient pressure the application of the ideal gas law is often sufficient, using real

gas property routines does not provide different results

- the accuracy of the property model should correspond to the accuracy of the other physical models; e.g. in two phase flow the results provided by models for pressure loss or heat transfer coefficients often show errors within the range of 30-50%, using complex models for calculating the density of the medium is not efficient in combination with models of limited accuracy.
- in dynamic simulations the assumption of thermal equilibrium in the working fluid may be not valid; the application of high accuracy property routines describing steady state systems does not improve the quality of the model compared to the real world.

Using high precision property routines does not necessarily improve the quality of simulation results; on the other side, the complex high precision property routine make the finding of a solution more difficult from a numerical point of view. The choice of the adequate level of complexity of property routines is essential for effective modelling and simulation.

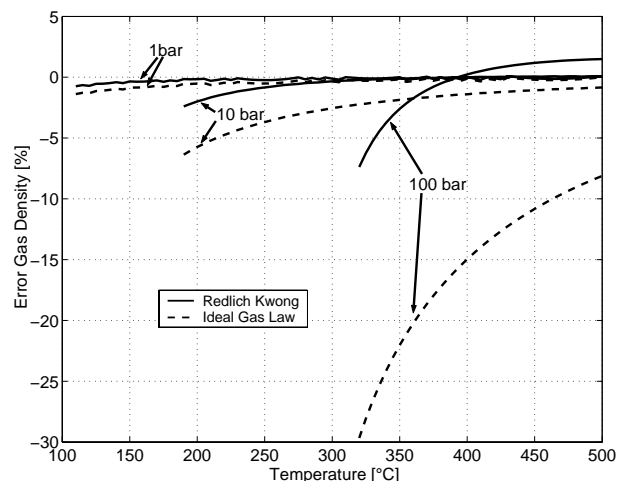


Figure 4: Accuracy of the results for density of steam provided by the Ideal Gas model and the Redlich-Kwong equation of different pressure levels.

Figure 4 shows the deviation from the reference values for the density of superheated steam provided by the Ideal Gas law and the Redlich-Kwong correlation. Depending on temperature and pressure range the difference in accuracy changes significantly: at low temperatures, the simple Ideal Gas law model shows almost the same quality like the Redlich-Kwong equation. For applications at high pressure levels the differences become significant, using the Ideal Gas law here probably is not acceptable.

3.2 Organisation of property models

The main package Medium includes six sub-packages with property models:

- Gas
- Liquid
- Solid
- MultiPhase
- MultiComponent

Two additional sub-packages (MediumSpecificData, MathTool) supply fundamental constants for various substances and mathematical tools like cubic equation solvers.

Property model can be separated into two groups:

- single phase models for pure substances
- multiphase and/or multicomponent models

In TechThermo, models describing a multiphase / multicomponent system are composed of basic single phase models for pure substances. Depending on the variables included in the models, these basic models can be divided into three groups:

- volumetric properties including pressure, temperature and density
- caloric properties including spec. enthalpy, heat of evaporation, spec. entropy and spec. internal energy
- transport properties like viscosity and heat conductivity

3.3 Basic concepts for Implementation of property models

Basic concepts for the implementation of property models should be demonstrated here by the example of the perfect gas law. The perfect gas is described by the ideal gas correlation between density ρ , temperature t and pressure p and a correlation between the caloric variables spec. enthalpy, spec. internal energy and entropy assuming a constant spec. heat capacity.

3.3.1 Definition of substances by record with fundamental constants

The Ideal Gas correlation is implemented in the model IdealGasVolumetricNoProp:

```

model IdealGasVolumetricNoProp "p/rho=RT"
  extends TTInterface.ThermalState.PropertyPort;

  replaceable TTMedium.MediumSpecificData.
  Data.MediumThermoFundamentalConstants
  SpecificConstants "record with medium specific
  constants";

  SIunits.SpecificHeatCapacity r_gas
  "spec. gas constant";

  parameter Boolean switch_r_gas_const=true
  "if switch_r_gas_const==true then specific gas
  constant r_gas is defined by
  parameter molar mass SpecificConstants.m_mol";

  equation

  if switch_r_gas_const==true then
    r_gas = GeneralCon-
    stants.R/SpecificConstants.m_mol;
  end if;

  StateCut.p = r_gas*(StateCut.t + 273.15)*
  StateCut.rho;

end IdealGasVolumetricNoProp;

```

The medium is defined by the spec. gas constant which is calculated from the molar mass m_{mol} and the general gas constant R . If the molar mass remains constant during the simulation (switch_r_gas_const is true) the value for the molar mass is taken from the record SpecificConstants which includes the fundamental constants of a medium. SpecificConstants is a record of type MediumThermoFundamentalConstants:

```

record MediumThermoFundamentalConstants
  "record defining reference state for
  thermophysical properties TTcode:CfD1"

  parameter SIunits.MolarMass m_mol "mo-
  lar mass";
  parameter SIunits.
  ThermodynamicTemperature t_critical
  "critical temperature";
  parameter SIunits.Pressure p_critical
  "critical pressure";
  parameter SIunits.Density rho_critical
  "critical density";
  parameter SIunits.SpecificHeatCapacity
  r_gas=GeneralConstants.R/m_mol "spe-
  cific gas constant";
  parameter Real omega_acentric "acentric
  factor";
end MediumThermoFundamentalConstants;

```

This record contains the molar mass and the state variables in the critical point. These constants are easily available for many substances. In the Tech-

Thermo library models are preferred which require only these fundamental constants for the specification of the substance.

The model `model AirIdealGasVolumetric` calculates the volumetric properties of dry air assuming the ideal gas law:

```

model AirIdealGasVolumetric
  "p/rho=RT for Air TTcode:Cal"
  extends TTMedium.Gas.Support.
  IdealGasVolumetricNoProp
  (redeclare TTMedium.
  MediumSpecficData.Data.
  AirThermoFundamentalConstants
  SpecificConstants);

```

`AirIdealGasVolumetric` extends the general model `IdealGasVolumetricNoProp` and defines the medium by the record `AirThermoFundamentalConstants`.

3.3.2 Flexible choice of used state variables

The correlations between the caloric variables are provided by the model `PerfectGasCaloricNoProp`.

assuming a constant specific heat.

There are various correlations for spec. enthalpy h and spec. internal energy u :

- 1 $h = c_p \cdot (t - t_0) + h_0$
 $u = c_v \cdot (t - t_0) + u_0$
- 2 $h = u + p / \rho$

There are also different options for the calculation of the spec. entropy:

- 1 $s = c_v \cdot \log(T / T_0) + R_{\text{gas}} \cdot \log(\rho_0 / \rho) + s_0$
- 2 $s = c_p \cdot \log T / T_0 + R_{\text{gas}} \cdot \log(p / p_0) + s_0$

Depending on the application, different formulation might be advantageous. The user can select options by structural parameters. Many property models in TechThermo offer alternative formulations for the calculation of material properties.

3.3.3 Icon representation of property models

The icons of the property model should provide first information about the included correlations. The basic icon is shown in Figure 5. The range of validity, the included variables and the substance is indicated in the corresponding areas of the icon.

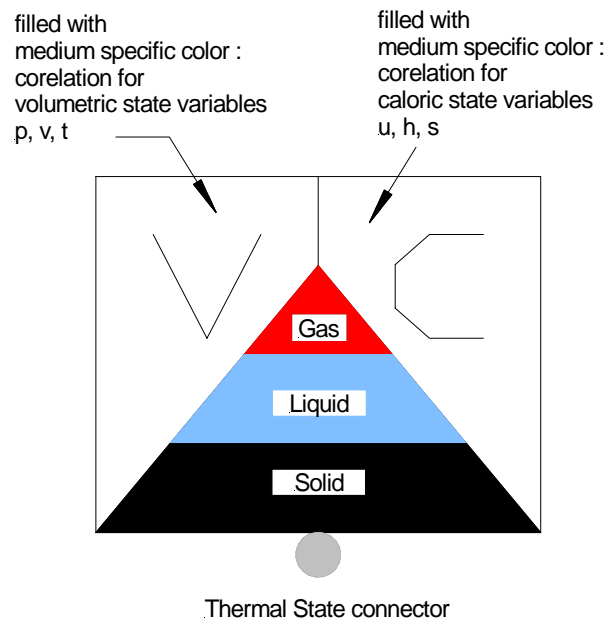


Figure 5: Icon used for models containing thermo-physical property correlations.

Figure 6 shows the icon of the `AirPerfectGasCalVol`. This model is composed of `IdealGasVolumetric` and a model `PerfectGasCaloric` as shown in Figure 7

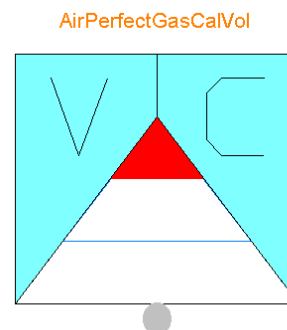


Figure 6: Icon for `AirPerfectGasCalVol` providing correlations for both caloric and volumetric state variables of dry air

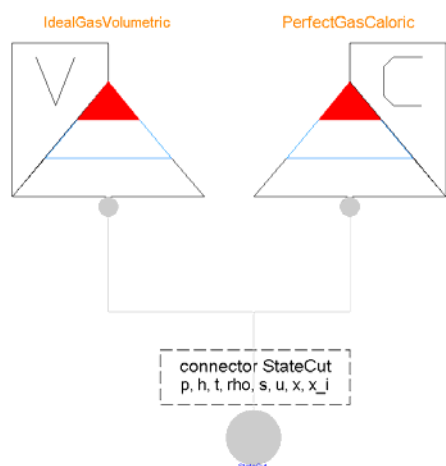


Figure 7: Diagram layer of `AirPerfectGas-CalVol`; combination of model `IdealGas-Volumetric` and model `PerfectGasCaloric`

3.3.4 Composed models

`AirPerfectGasCalVol` represents a simple form of a composed model; two basic models are joined to offer the complete set of state variables. The concept of composed models becomes especially interesting for multiphase models which are composed of models representing the separate phases. For wet steam with steam quality x the thermal state variables can be calculated from the properties of the liquid and the gas phase:

- $h = h' + x * (h'' - h')$
- $u = u' + x * (u'' - u')$
- $s = s' + x * (s'' - s')$
- $\rho = 1/\rho' + x * (1/\rho'' - 1/\rho')$

h' , u' , s' , ρ' are the properties of the saturated liquid, h'' , u'' , s'' , ρ'' are the properties of the saturated steam. Additional correlations are:

- $\Delta h = h'' - h'$
- $s'' - s' = \Delta h / (t_{\text{sat}} + 273.15)$
- $h = u + p / \rho$

with heat of vaporization Δh and t_{sat} saturation temperature. Model `WetSteamV01NoProp` is composed of five models to calculate the properties of wet steam:

- `TSatPSatAntoineNoProp` provides the correlation between saturation temperature and saturation pressure
- `HeatVaporizationNoProp` calculates the heat of vaporization dependent on the saturation temperature
- `VariableRhoCalVolNoProp` calculates the properties for the liquid phase for saturation pressure and saturation temperature
- `IdealGasVolumetricNoProp` calculates the properties for the gas phase assuming saturation pressure and saturation temperature
- `TwoPhaseMix` calculates the properties of wet steam from the information provided by the other four models

`WetSteamV01NoProp` can be specified for any substance provided the following constants for the substance are available:

- state variables in the critical point
- two constants needed for the Antoine correlation for saturation pressure /saturation temperature
- a single value for the heat of vaporization dependent on saturation temperature

The modular approach is advantageous regarding the consistency at the transition between different regions. Since the properties of the two phases of the wet steam are calculated using single phase property models there are no discontinuities at the transitions between the two phase region and the single phase regions.

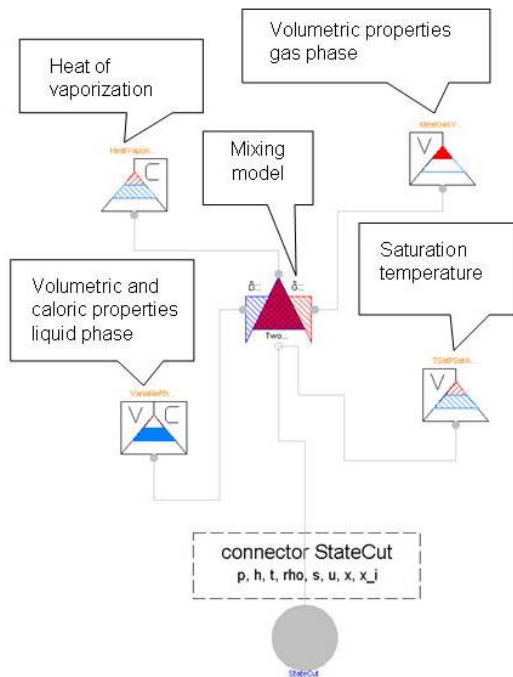


Figure 9: Diagram layer of `WetSteamV01NoProp` for calculation of properties of wet steam.

3.3.5 Alternative formulation for non-linear correlations

The convergence behaviour of a model is influenced by the formulation of non-linear property correlations. Which variable is chosen as dependent might affect the numerical performance of a model. If possible, TechThermo offers for the non-linear property model various formulations. One example is the Redlich-Kwong equation for the volumetric properties of gases:

$$p = \frac{RT}{\frac{1}{\rho} - b} - \frac{a}{T^{0.5} \frac{1}{\rho} \left(\frac{1}{\rho} + b \right)}$$

coefficients a and b are calculated from values for the critical state.

As already shown in Figure 4, this cubic algorithm provides better results as linear correlations like the ideal gas law. The model `RealGasVolumetricNoProp` offers three different options:

- 1 acausal formulation
- 2 temperature t as function of pressure p and density ρ
- 3 density ρ as function of pressure p and temperature t

The user can select one of these three options by a structural parameter. For options 2 and 3 a cubic equation solver is used.

Conclusions and Outlook

The TechThermo library includes a basic set of models with correlations for thermophysical properties. These models have been selected considering the range of application, so only models which can be used for many different substances are included. The property models in TechThermo should allow complete descriptions of a thermodynamic system in the initial phase of model developing. Since property models are separated from the other parts of a model by a thermal state connector, a quick exchange of property models is possible.

Further development aims at implementing property models which are more efficient regarding numerical aspects. These models should offer flexibility in the selection of the dependent variable and should optimize the calculation time / accuracy ratio. One approach here includes the transformation of state variables and the application of cubic equation solvers.

References

- [1] Reid, R.C., Prausnitz, J.M., Poling, B.E. : "*The Properties of Gases & Liquids*", 4th edition, McGraw-Hill Book Company, 1988

Development of a Simplified Transmission Hydraulics Library based on Modelica.Fluid

Michael M. Tiller

Research and Advanced Engineering, Ford Motor Company
mtiller@ford.com

Abstract

Modeling of hydraulic systems often leads to systems of equations that are stiff and difficult to solve. In many cases, stiffness of these systems can be traced to orifices and relatively small volumes within the model. Frequently, such volumes and orifices are only present to facilitate explicit state-space formulations of the underlying conservation principles.

In an effort to create more efficient models and to eliminate the need for insignificant or non-physical contributions from such components, the new Modelica.Fluid library [1] introduces a structured set of base classes (leveraging new features in the Modelica language) from which fluid component models can be built. These base classes allow for a wider range of component configurations by eliminating the need for extraneous volumes and orifices in hydraulic schematics.

Using the Modelica.Fluid library as a foundation, another library has been developed that includes hydraulic components for hydraulic transmission modeling. The models are aimed at addressing lingering performance and robustness issues with hydraulic circuits in transmission models and include several useful simplifications. Because these component models use a first-principles formulation (*i.e.* conservation of mass and energy), it is possible to mix simplified or idealized components with models that include complex dynamics. As such, model developers can focus on the dynamics of interest (*e.g.* dynamics associated with the design of a specific spool valve in a transmission) while still capturing the basic functional behavior for the other components in the system. The result of this approach is a practical continuum between functional and predictive modeling.

Keywords: Transmission, VMA, hydraulics, DAEs

1 Goals

Models for transmission hydraulics usually have one of two purposes. The first purpose is to be a *functional* reproduction of an existing or proposed transmission design. In this case, the response of the model is only intended to reproduce the functional behavior of the actual transmission hydraulics. Such models would naturally include delays, approximate rates of response, etc.

The other purpose is to be a predictive model of a particular transmission design. Such a model is referred to as a design-oriented model because it can be used to conduct “what if?” studies on potential design candidates. The key requirement for this kind of model is that it should not only be sensitive to the relationship (both transient and steady-state) between the inputs to the model and the outputs but it should also properly predict the hydraulic response *as a result of changes to the design parameters* (orifice sizes, volumes, diameters, etc). This latter possibility requires considerable physical detail and a solid, first-principles understanding of component behavior.

The goal of the component library described in this paper is to provide a path to move between these two types of representations easily. In this way, a single model of the hydraulic system can be developed that can be selectively refined to serve both purposes. In some cases, it is useful to consider both purposes in the same model. For example, as part of the design process for a particular valve a design-oriented model of that valve can be used in conjunction with functional models of the remaining components. This not only brings the potential for faster simulations to speed up the design process but also integrates nicely the cascading nature of system engineering based development processes.

2 Functional vs. Predictive

2.1 Background

The simplest way to demonstrate the differences between the formulation and performance of functional and predictive models is to look at an example in detail. However, even basic hydraulic circuits are a little difficult to understand without some background. For this reason, we will first explain how a basic hydraulic regulator valve works before moving on to a complete circuit model. For simplicity, we consider the single solenoid value, shown in Figure 1.

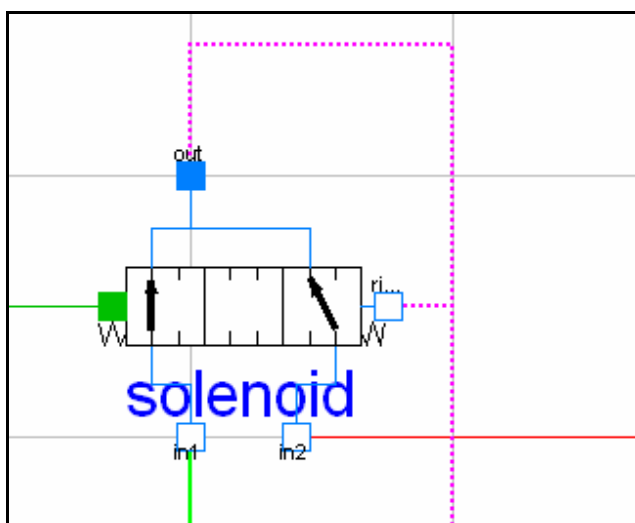


Figure 1: Sample Regulator Valve

A common characteristic of a regulator valve is that an axial force is applied to both sides of the valve. In most cases, the force is the result of a pressure applied by a fluid over the exposed area on that side of the valve. But the force can be generated by other means as well (*e.g.* in the case of a solenoid valve, an electro-magnetic force is applied to one side). The “output” port of the valve (shown in Figure 1, at the top of the valve) can be fed from two potential sources (indicated in Figure 1 by the ports on the bottom of the valve) depending on the position of the valve. In the case of all valves shown in this paper, the larger the force on the left side of the valve, the more flow will come from the bottom right input port. Conversely, the larger the force on the right side of the valve, the more flow will come from the bottom left input port. In simple terms, the output pressure will be biased toward the pressure in the input port on the opposite side of the larger force. To tune the performance of the valve, an “offset” force can be generated using a preloaded spring inside the valve body.

2.2 Functional Model

The model shown in Figure 2 represents an example of a pressure control circuit for an automatic transmission.

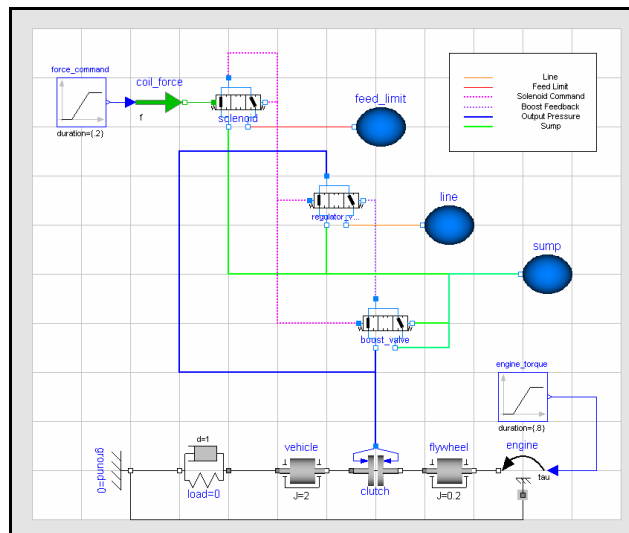


Figure 2: Sample Pressure Control Circuit

Using the background provided in Section 2.1, we can now explain the circuit shown in Figure 2 and deduce the following functional behavior:

- A force command signal (upper left) is sent to an electric coil in the solenoid. This coil applies the commanded force to the left side of the solenoid valve.
- Because the output pressure of the solenoid is also the pressure applied on the right side, the solenoid valve will seek a position that balances the electric coil force with the “output pressure” of the valve. In this way (and with a gain that depends on the areas involved), the output pressure of the solenoid valve is controlled.
- The solenoid output pressure is applied to the left sides of both the regulator and the boost valve. The boost valve (bottom valve) is designed with a preload such that it does not open until a critical pressure has been reached. Above that critical pressure, its output pressure starts dropping to the sump pressure. Otherwise, its output pressure follows the regulator valve pressure.
- Before the critical pressure of the boost valve is reached, the regulator valve functions much like the solenoid valve because its output pressure is effectively the balancing force on the right side. Again, given the areas involved a certain gain is achieved.

However, once the boost valve starts to open, the pressure on the right side of the regulator quickly drops giving the pressure on the left a greater mechanical advantage (and thus increasing the gain significantly).

- Finally, the regulator valve output is applied to the clutch. The clutch plates do not come into contact with each other until a critical pressure is reached. Prior to this, the flow into the clutch fills the gap that forms as the plates moves. This filling effect results in a delay between the regulator output pressure and the applied clutch pressure.

2.3 Detailed Model

The functional description in Section 2.2 describes how the circuit is supposed to function. However, the functional description assumes a steady-state response with no dynamic effects. In reality, there are many dynamic effects.

For example, each valve includes a small volume on each end that fills and empties as the force balance changes. In addition, the flow is regulated by orifices which open and close as the valve moves. The behavior of the orifices is non-linear and very sensitive to the machining of the spool itself. Furthermore, these circuits are designed to provide large flow rates which means the fluid itself can build up a significant amount of momentum. Finally, the compressibility of the transmission fluid (transmission fluid often includes a significant amount of trapped gas) combined with the small mass of the spool can result in high-frequency oscillatory responses.

While the dynamics described in this section are on a much smaller time scale than the functional dynamics described previously, they can have a very significant effect on noise and vibration in the mechanical system. As such, these kinds of detailed models (and the staggering amount of geometric component data they require) are very useful in the design of the pressure control circuits.

2.4 Structural Differences

The biggest difference between the functional and predictive models is the presence of dynamic terms. For example, the momentum balance on the regulator valve spool can be expressed as¹:

$$\sum F = p_L A_L - p_R A_R - m\ddot{x} - kx = 0$$

¹ For simplicity, so-called “flow forces” (*i.e.* reaction forces from changing the fluid momentum) are neglected.

where p_L is the pressure on the left side, A_L is the area on the left side, p_R is the pressure on the right side, A_R is the area on the right side, m is the mass of the spool and x is the position of the spool.

Similarly, the mass balance for the volume on the left side of the valve can be expressed as:

$$Q_i - Q_o - \dot{\rho}A(x - x_0) - \rho A\dot{x} = 0$$

where Q_i is the mass flow rate into the volume, Q_o is the mass flow rate out of the volume, ρ is the density, A is the cross-sectional area and x is the position of the valve.

The inclusion of the capacitive elements makes the formulation of the problem simpler because most, if not all, of the equations can be written as explicit differential equations, *e.g.*,

$$\begin{aligned}\dot{x} &= v \\ \dot{v} &= \frac{p_L A_L - p_R A_R - kx}{m} \\ \dot{\rho} &= \frac{Q_i - Q_o + \rho A v}{A(x - x_0)}\end{aligned}$$

However, as we will discuss later the simpler formulation is actually much more expensive to solve because of the high frequency dynamics in the system. Because many of these capacitive elements contribute nothing to the overall functional behavior, one simplification is to eliminate them. Rather than deleting them from the model, they can be “logically” deleted by setting their capacitances (*e.g.* m and A) to zero. As we can see from the differential equations, if we continue to rely on the explicit differential equations, such an approach would lead to a singular system of equations because the denominators would go to zero. However, if we reconsider the structure of the problem given that these capacitive terms are zero and allow purely algebraic constraints to appear in the problem formulation (effectively turning the problem into a system of differential-algebraic equations), then we get the relatively simple system:

$$\begin{aligned}x &= \frac{p_L A_L - p_R A_R}{k} \\ Q_i &= 0\end{aligned}$$

While many traditional dynamic system analysis tools are based on the notion that the dynamics *must* be characterized in terms of ordinary differential equations (ODEs), Modelica specifically broadens the general problem definition to support differential-algebraic equations (DAEs) [2]. This broader

problem definition combined with sufficient symbolic manipulation [3] (as in Dymola, [4]) means that such simplifications are a practical means of formulating simplified systems of equations using exactly the same component models.

The equations in this section touch on only a few of the structural differences between functional and predictive models. Modelica also provides features for expressing more complex behavior like parametric behavior formulations (often used to describe diodes or clutches [5]) and piecewise linear expressions (used to describe convected property balances in `Modelica.Fluid`). All of this means that many of the limitations that exist when forced to cast component behavior in terms of explicit differential equations can be completely eliminated. As a result, alternative formulations that are more natural, flexible and computationally efficient, like the ones used for the components in this paper, become possible.

It is worth noting that useful behavioral descriptions are typically sufficiently complex that the resulting algebraic equations are both non-linear and coupled. As a result, after symbolic manipulation simultaneous non-linear systems of algebraic equations (*i.e.* “algebraic loops”) often emerge from the explicit differential equations. However, the cost of the non-linear iterations is often much less than the cost of resolving the fine details associated with higher-order dynamics.

3 Comparisons

In this section we will quantify many of the differences in structure and performance between these two types of models. We start with the functional model and then include the higher-order dynamic effects. In addition to comparing the simulation results, we will also consider how these effects change the overall structure of the problem and what impact this has on simulation time.

3.1 Functional Validation

Before we compare the structure and performance of these two types of models, we should first perform a basic validation of the model. Note that while the example in this paper was chosen to provide a “real-world” context to the issues, the model itself was created specifically for this paper and is not a validated automatic transmission circuit.

There are two main characteristics of interest in this circuit. The first is to confirm the effect of the boost valve. This should create a “knee” in the pres-

sure response of the circuit. The other effect is the filling of the clutch volume as the clutch is stroked.

Figure 3 shows the functional validation of the circuit by plotting regulator valve output pressure and clutch pressure as a function of coil force. When the coil force reaches 35 Newtons, we can see the knee in the output pressure. In addition, the difference between the clutch pressure and the regulator valve output pressure is caused by the need to fill the gap formed as the clutch is moved into position.

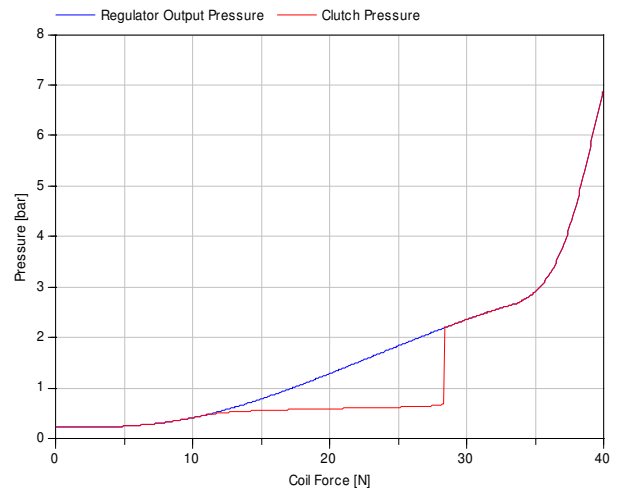


Figure 3: Functional Validation

3.2 Adding Predictive Details

To demonstrate the impact of including predictive details in the model, we will modify the functional model to include 3 important physical effects: spool dynamics, valve orifices and valve volumes. These are just a few typical examples of complexities that are required to truly predict the response of the hydraulic systems. Other effects are also significant (*e.g.* fluid inertia, mechanical limits, compressible media) but for simplicity they will be neglected.

For the functional model, the spool mass is assumed to be zero. As a result, the position of the spool in the functional model is determined by the steady state force balance on the spool. However, in the case where the spool has non-zero mass, the balance of the axial forces determines the acceleration of the spool².

To simplify the calculation of the valve output pressure, the functional model prescribes the output pressure by blending the two input pressures continuously as a function of the spool position. The appropriate mass flow rates to achieve this are computed implicitly. This is not particularly physical

² In addition to the spool mass, some damping must be introduced as well.

because it assumes that the valve can flow any amount of mass. In reality, such flow rates are limited by the sizes of the various orifices. For the predictive model, the flow through each path in the valve is computed explicitly based on the pressure drops between the ports. While the equilibrium position of the valve will be identical in each case, the presence of orifices results in constraints on how quickly the control circuit can respond. In addition, the nature of the orifice equation typically results in some numerical issues.

The final detail is the filling and mixing associated with the volumes at the ends of the valves. In the functional model, the volumes at the ends of the valves are neglected (*i.e.* no mass or energy capacitance). For the detailed model not only is this volume included, but it varies with spool position.

While these dynamics complicate the response of the circuit, they do not change the overall functional behavior. Figure 4 shows the response of the circuit with these physical details included. The conditions are nearly identical to those used to generate Figure 3.

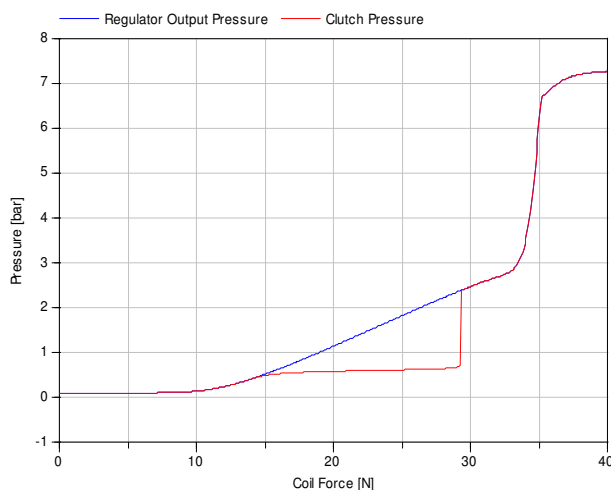


Figure 4: Detailed Model Validation

It should be noted that the conditions chosen for validation in Figure 4 are such that the dynamics have little impact. The biggest feature is the “smoothing” of the edges. However, because the progression of coil force proceeded on a quasi-steady time scale, the dynamics are not visible. The main purpose of Figure 4 is simply to show that with the physical details included, the circuit is still functionally equivalent.

3.3 Transient Response

To highlight the implications that the various physical details have on the transient response, we need to drive the circuit under more realistic operat-

ing conditions. Figure 5 shows what a typical solenoid command might look like. The square pulse at the start is used to fill the clutch volume. At the end of the square pulse, the friction materials in the clutch should just be coming into contact. At approximately 0.4 seconds, the solenoid force drops to allow the clutch to engage smoothly. The force is then slowly ramped up to increase the capacity of the clutch. Once the clutch is locked, the coil force jumps up to keep the clutch firmly engaged.

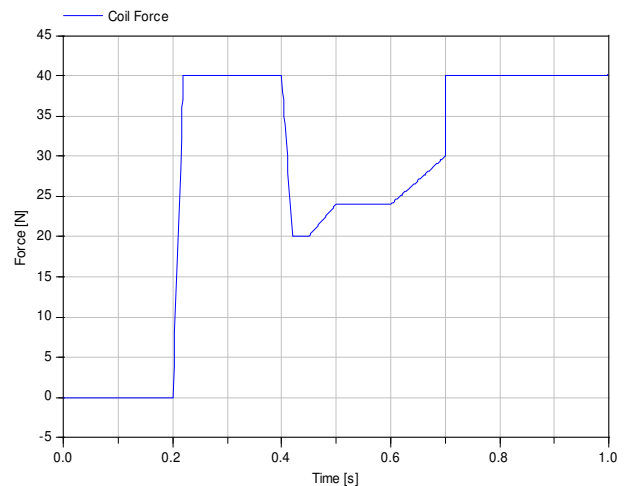


Figure 5: Solenoid Force Command

The functional model response is shown in Figure 6. Note that the clutch pressure does not respond until after the clutch volume is filled. Once the volume is filled, the response of the clutch pressure closely follows the force command profile shown in Figure 5.

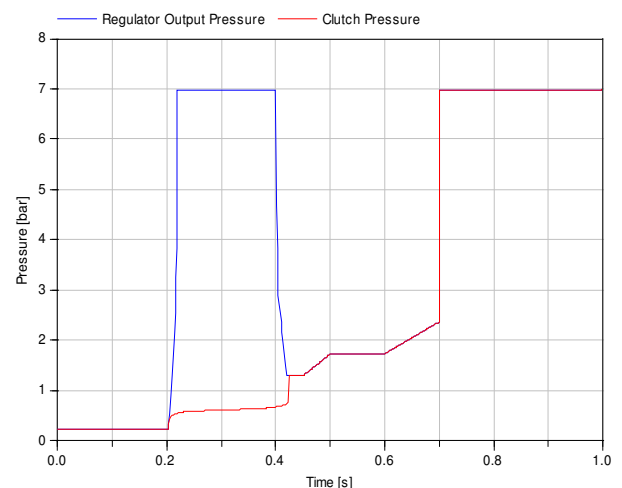


Figure 6: Functional Model Response

By comparison, Figure 7 shows how the more detailed model responds. Rather than responding cleanly to the coil force command, the regulator valve output pressure fluctuates as the spool settles into a quiescent state. In particular, the spool oscillates significantly in response to step changes in the

coil force. Most of these oscillations occur before the clutch starts to engage, but some of them can clearly be seen in the clutch pressure response. Another interesting effect shown in Figure 7 is the delay in the engagement of the clutch. This is due to the fact that the regulator output pressure does not drop immediately to a pressure that is in proportion to the coil force. Instead, the clutch pressure drops lower than it did in the functional model which causes the clutch volume to briefly empty before recovering. These dynamics introduce an additional delay before the friction materials come in contact.

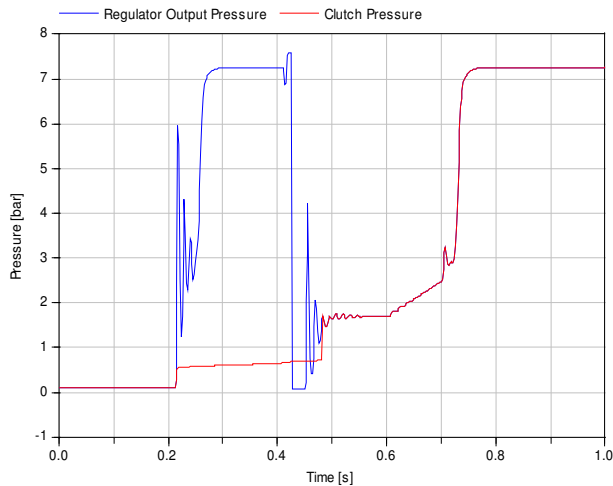


Figure 7: Detailed Model Response

3.4 Structure and Performance

Table 1 highlights some of the structural and performance differences between the simulations shown in Figure 6 and Figure 7. Because the functional model is approximately 5 times faster than the predictive models, it is useful for proving out control strategies or for performing hardware-in-the-loop testing. The performance differences between these types of models could become even more significant as additional physical details are added or as the overall complexity of the circuit increases.

	Functional Model	Detailed Model
# of states	6	20
Linear system sizes	{6, 2, 11}	{6, 10, 4, 13}
Nonlinear system sizes	{8, 9, 6}	{3, 13}
CPU time	0.09 [s]	0.46 [s]

Table 1: Quantitative Comparison

Another way to visualize the differences in the dynamics of the two models it to visualize the poles in each model. In this way, the range of time constants and natural frequencies can be quickly assessed. Figure 8 shows a map of the two poles present in the functional model. One of the poles has a time constant of 1 millisecond and corresponds to a specific first order response introduced in the filling model. The other pole has a natural frequency of 0.08 Hz and corresponds to the mechanical response of the clutch-inertia system shown at the bottom of Figure 2.

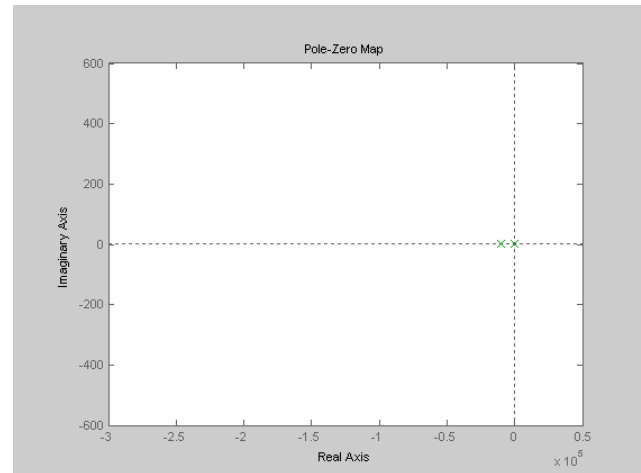


Figure 8: Poles in the Functional Model

For comparison, the poles of the predictive model are shown in Figure 9. An important difference between the poles shown in this figure and the ones shown in Figure 8 is that the dynamics in the predictive model are non-linear. As a result, the positions of the poles vary as a function of the states in the predictive model. For this reason, Figure 9 overlays the values of the poles (computed via linearization) at various times during the predictive model response shown in Figure 7 to demonstrate the range of the dynamics.

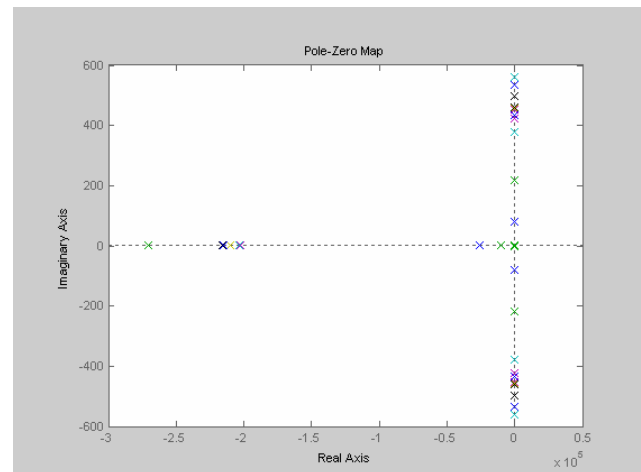


Figure 9: Poles in the Predictive Model

The lowest natural frequency shown in Figure 9 has a value of 18.7 Hz and it exists prior to the first change in solenoid pressure. The highest natural frequency in the system is 90.7 Hz and it appears during the drop in solenoid pressure at approximately 0.41 seconds.

A more intangible quality to these models is the underlying robustness. While Table 1 compares the performance of simulations that were run using these two models, what it does not show is the fact that the detailed model is less robust numerically than the functional model. Singularities associated with vanishing volumes, ill-posed Jacobians, *etc.* can not only have an even greater detrimental impact on the simulation time, they can prevent the simulation from completing at all.

4 Conclusions

A common issue in modeling applications is including the appropriate level of detail for the task at hand. During the initial design phase of a circuit like the one shown in Figure 2, it is important to quickly verify the functional performance of the circuit and/or the control strategy behind it. Simple models can quickly confirm the steady state clutch pressure achieved for a given solenoid force. Then, as the design process focuses on finer details (orifice diameters, spool masses, *etc.*) additional geometric information can be added that allows additional dynamics to be considered.

This paper highlights several advantages of using Modelica for hydraulic system modeling. The first advantage is the ability to leverage the `Modelica.Fluid` and `Modelica.Media` libraries. Careful thought has gone into the formulation of these libraries to leverage as much of the potential of the Modelica language as possible while still providing a relatively straightforward framework for developing components. These libraries can now serve as the foundation for the development and exchange of hydraulic component models.

The other advantage of using Modelica for hydraulic systems is the ability to express idealizations that fall outside the typical formulations. It is no longer necessary to build models from alternating “flow-volume” pairs or to consider only behavioral models that lead to explicit differential equations. Instead, Modelica allows the expression of a broader class of behavioral models which, through symbolic manipulation, can be simplified down to relatively simple and efficient algebraic relationships.

Acknowledgments

The central idea for this paper came after reviewing various models used within Ford to model transmission hydraulics. As such, I would like to thank Chad Griffin and James McCallum for their willingness to share and discuss their models. I would also like to thank James for the time he graciously spent explaining many of the nuances of hydraulic control circuits. Greg Pietron, Bill Tobler and James McCallum also provided valuable feedback while reviewing this paper. Finally, I would also like to thank Bill Tobler and Steve Frait for taking the time to discuss and explain the behavior of specific components discussed in this paper.

References

- [1] Elmqvist H., Tummescheit H., and Otter M., “Object-Oriented Modeling of Thermo-Fluid Systems”, *Proceedings of the 3rd International Modelica Conference*, Linköping, Sweden, 2003.
- [2] “*Modelica – A Unified Object-Oriented Language for Physical Systems Modeling*”, Version 2.1, January 30th, 2004. Modelica Association, <http://www.modelica.org/documents/ModelicaSpec21.pdf>
- [3] Mattsson, S. E. and Söderlind G., “*Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives*”, *SIAM J. Sci. Comput.*, 14(3):677-692, May 1993.
- [4] Brück D., Elmqvist H., Olsson H., Mattsson S.E., “*Dymola for Multi-Engineering Modeling and Simulation*”, *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, 2002.
- [5] Tiller M. M., “*Introduction to Physical Modeling with Modelica*”, *Kluwer Academic Publishers*, 2001. ISBN 0-7923-7367-7.

Session 3c

Methods II

Probabilistic Analysis and Design Optimization of Modelica Models

Björn Johansson Petter Krus
 Department of Mechanical Engineering
 Linköping University, Sweden

Abstract

In this paper, the system simulation model is discussed from an engineering design perspective. Special emphasis will be given Modelica models, and it is exemplified how computational design methods operate on the simulation model in order to evaluate different concepts. Model based design optimization and probabilistic analysis are discussed as examples of such computational methods.

An XML-based information system for representation and management of design data for use together with the Modelica model is further proposed in order to simplify the use of computational design methods.

Finally, an example is presented, where probabilistic analysis is carried out on a Modelica model of an aircraft actuation system using the proposed and implemented tools and methods.

1 Introduction

In the area of engineering design, a substantial part of the process consists of manual design work involving the inspiration and creativity of the designer. However, a large part of the design process can be formalized, and by applying formal design methods, these can be implemented in computer software as *computational design methods*. By employing computational methods in early stages of the design process, it is possible to acquire valuable informa-

tion. Such methods could for example include *model based design optimization* or *probabilistic analysis*. These computational methods will be described in more detail throughout the paper, but common for the methods is that they operate on *simulation models* in an automatic, iterative way. This implies new requirements on the simulation tools as well as on the representation and management of data related to the computational methods.

2 Computational design methods

As indicated in the introduction, a computational design method the uses the simulation model as the primary source of information.

The principal similarities between different computational design methods and how they operate on the simulation model are illustrated in Figure 1. With this view, the computational methods either operate on the inputs to the model (design synthesis), or on the outputs from the model (design evaluation). Both probabilistic analysis and design optimization can be seen as automatic methods that repeatedly execute and evaluate the simulation model.

This way of automatic execution adds specific demands to the simulation environment. From the design perspective, it is not of interest exactly how the model is executed, but it must be valid and must not ‘fail’ or get ‘stuck’. It also calls for separation between the actual simulation model and information related to perform a design task using computational

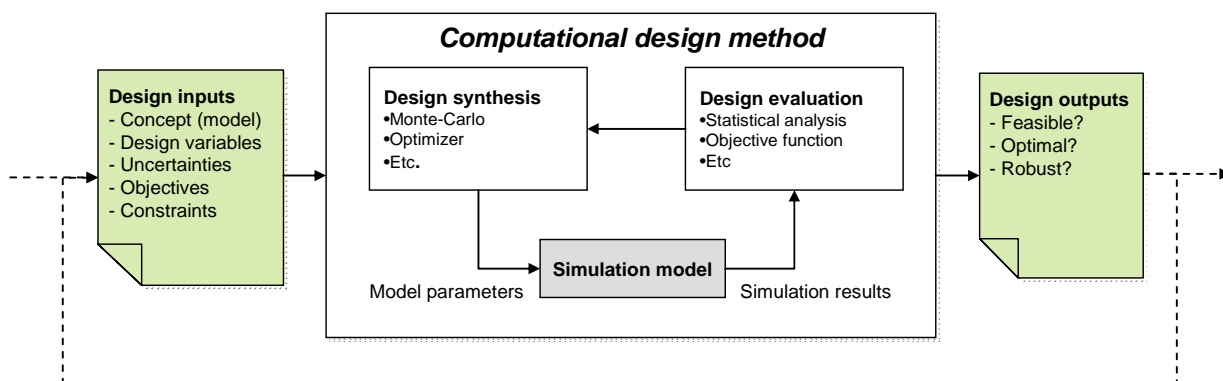


Figure 1. Computational design methods operating on a system simulation model.

methods. This is because the same simulation model could be used in a wide range of design tasks.

2.1 Model based design optimization

A typical example of a computational design method is *design optimization based on system simulation*, as described by Krus et al. [4].

By formulating requirements and desirables as a mathematical objective function, design optimization can be employed. Parameterized simulation models of the system enable an optimization algorithm to be used to find the system parameters that maximize the objective function while meeting the constraints. The optimization algorithm repeatedly modifies specific design variables (model parameters), executes the model, and evaluates an objective function, see Figure 2.

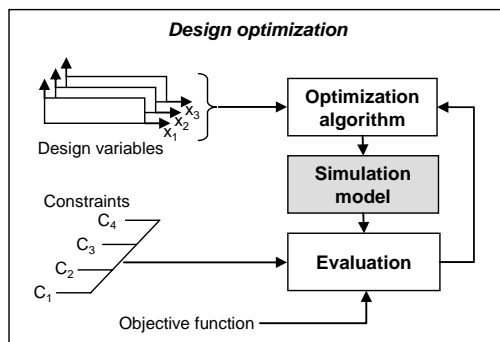


Figure 2. Process for model based design optimization.

A non-gradient method is specifically appropriate for optimization of simulation models since the objective function is defined from simulation results and derivatives of the objective function can not be defined. One example is the *Complex* optimization algorithm, presented by Box [6], which has been used very successfully over a wide range of problems and is characterized by simplicity and robustness.

2.2 Model based probabilistic analysis

Other important examples of computational design methods are based on *probabilistic analysis*. These methods are used not only to assure a technically feasible concept, but also to find a robust design point by including uncertainty in the models.

In all stages of the engineering design process, and especially in early stages, most available information suffers from uncertainty. By using methods for probabilistic analysis, this uncertainty is brought into the design process through the use of simulation models. This is highly desired since important knowledge about the uncertainty is otherwise omitted.

For example, by taking uncertainty into account, the following information can be extracted:

- The probability of meeting a set of constraints and achieving a technically feasible design with in the ranges of the design variables, the *probability of feasibility*.
- How much it will be necessary to relax a specific constraint in order to have a sufficiently high probability of feasibility.
- The effect of uncertainty in system parameter values, i.e. the robustness of the design

The information above can not be achieved using deterministic simulation models with fixed parameter values. Therefore, it is necessary to use *probability distributions* to represent uncertain values on model parameters.

A *feasible design* is defined as a design that satisfies all imposed technical constraints [5]. The examination of the concept's feasibility could be seen as a probabilistic methodology where the probability of finding feasible design alternatives within the design space is investigated. This so-called probability of feasibility, P_{feas} , is an important figure of merit in the early phases of design since it indicates whether the concept is promising for further analysis such as design optimization.

Figure 3 illustrates the process of *concept feasibility assessment*. By assigning normal distributions for the design variables and using a sampling-based method such as the Monte Carlo simulation together with the simulation model, the P_{feas} can be calculated given the settings of the design variables and the constraints.

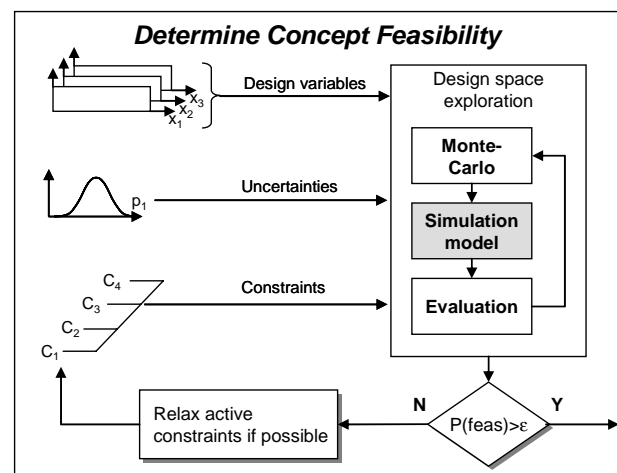


Figure 3. The process of concept feasibility assessment [5]. The model code is evaluated repeatedly where the design variables are varied within the design range using a sampling based method such as Monte-Carlo simulation.

If the total probability of feasibility is too low, the constraints must be investigated individually and either the active constraints relaxed or the concept modified, for example by infusing new technologies to the concept and thereby improving its characteristics. Mathematically, the probability of feasibility P_{feas} for a system with m constraints is defined as [5]:

$$P_{feas} = \prod_{i=1}^m P_i \quad (1)$$

$$P_i = P(C_i \leq 0) \quad (2)$$

where P_i is the probability that one specific constraint C_i is met. For another formulation using information content as the figure of merit, see the theory of Axiomatic Design [8]

The Monte-Carlo simulation used to simulate uncertainty or variability is a rather simple algorithm that randomly samples values according to a probability distribution. However, more sophisticated methods with improved search efficiency can be used as well such as Adaptive Importance Sampling (AIS) as described by Wu in [11].

2.3 Computational design data

As indicated in the previous sections, computational design methods include a wide range of data that is not primarily associated the model of the system. As can be concluded from Figure 2 and Figure 3, a wide range of *design related* data is required such as

- *Design variables* – A subset of the system parameters that are modified during the design iteration.
- *Uncertainties* – Many model parameters are uncertain, which must be handled.
- *Constraints* – Measures that must be met in order for the design to be feasible.
- *Objective functions* – A mathematical function used by an optimization algorithm in order to define a figure of merit.
- *Process model* – In order to accomplish full system simulation and optimization involving several types of models and codes, it is necessary to be able to represent and execute a computational sequence.

The data above is normally not possible to represent inside simulation models. It is also the fact that a computational design task often includes more than one model represented using one specific approach. In order to accomplish for example system optimization, it is often necessary to include several types of models, such as CAD, CFD, financial models, etc.

Typical is also that integration of already existing, so-called legacy codes is necessary.

3 Modelica and computational data

The Modelica modelling language is developed in an international effort by the Modelica Association [6] consisting of members from both industry and the academic world with the intention of establishing a de-facto standard for system simulation. The Modelica language contains a large number of features with extensive support for advanced modelling of systems from different engineering domains. The modelling principle is object-oriented and equation based where different types of equations are supported. Modelica also enables representation of general data as so-called annotations.

It has been shown several times that Modelica is very well suited for modelling of physical systems. However, representation of design related data as exemplified in previous section is not directly supported. Even if it would be possible to represent design data as annotations this is not an attractive solution since it still not would be generally supported in tools available for Modelica.

One important argument why a separate representation of design data would be necessary is:

A design project often contains several models, and several types of models. In order to fully assess the properties of a certain design, this could include both technical domains and others, such as financial models. A general representation of design data that is simple to use together with different model implementations is therefore necessary.

The approach taken in this work is to represent the data as XML outside the simulation model as illustrated in Figure 4. This approach will be further described in the next section of this paper.

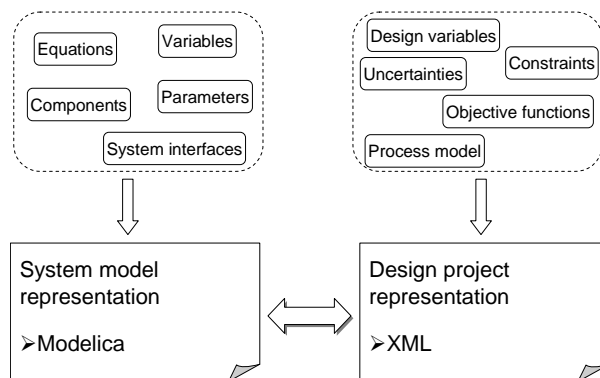


Figure 4. The system simulation model is represented in Modelica, while data regarding the design task is represented in XML.

3.1 XML-based data repository

In order to facilitate the use of computational design methods using models implemented for example as Modelica, a design data repository has been created where the system data can be represented in a general way using XML. An XML document is however not very usable without an accompanying XML schema [10]. Just as the XML can effectively describe data, the XML schema defines the structure of the XML document. It defines each allowed element in a document, the allowed attributes and possibly the acceptable attribute values for each element. It also defines the occurrences, sequence, and nesting of each element.

The information model developed for this purpose has a hierarchical and object-oriented structure in order to organize the data in a way that is close to the physical system. In order for the information model to be as general as possible, generic elements are defined such as *system*, *subsystem*, *variables* and *native data*. A top level structure of the data can be seen in Figure 5, and the different parts of the data model are described in more detail below.

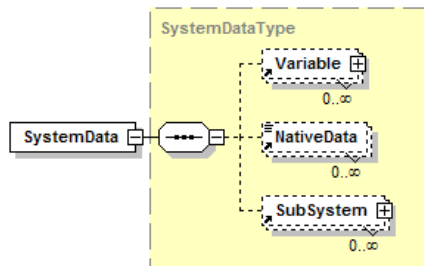


Figure 5. An object oriented and hierarchical structure in order to organize the design data.

The *variable* element is the important building block in the repository. This element is used as a neutral representation of both system parameters and design variables, see Figure 6. Besides name and default value, which are required attributes, the variable contains optional information such as unit, description, and data type. With a *variable type* attribute, it is also possible to define whether the variable is controllable, non controllable, or a so-called technology factor (described in more detail in [3]). As illustrated in Figure 6, the variable element also has sub-elements that contain additional information such as probability distribution and settings if the variable is generated by a design algorithm such as Design of Experiment (DOE) or is a design variable in an optimization algorithm. It is possible to attach these sub-elements to all variables in a generic way.

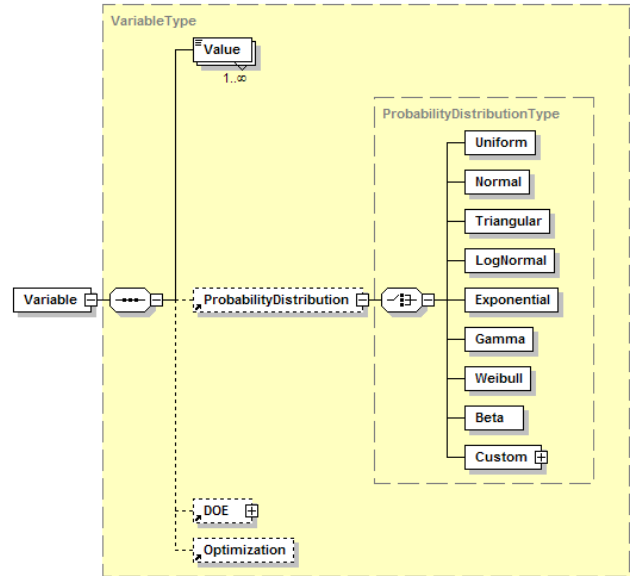


Figure 6. XML Schema representation of the variable element used to represent various kinds of system parameters with extensive information such as probability distributions.

The idea is that probability distributions are defined and stored parametrically. It is possible to select from typical standard distributions such as uniform distribution, normal distribution, triangular distribution, etc. Custom distributions could also be defined as interval values or single values. This means that no mathematical functions for the distributions are stored in the repository. For example, in the case of a normal distribution, the mean value and the standard deviation are stored and not the mathematical function describing the relation between these metrics and the probability density function, PDF.

In Figure 7, some example XML code is visualized as represented using the XML editor XML Spy. For visualization of the actual XML code, see the example in section 5.

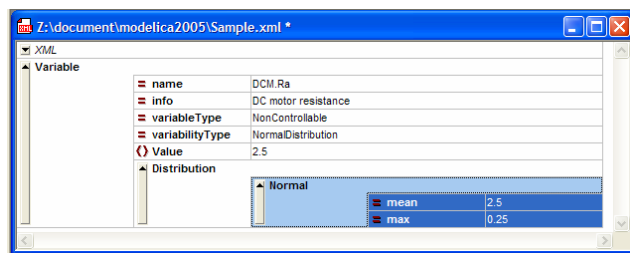


Figure 7. Design variable visualized in the XML editor *XMLSpy*.

4 Integration framework

A software prototype for collaborative system simulation and computational design has been developed

in projects prior to the work presented in this paper; see for example [2].

The framework is based on a *Service Oriented Architecture* [7] which means that models and methods communicate using so-called *web service* standards such as SOAP and WSDL, see [9]. The standards are used to define interfaces between the models and to represent the data being exchanged between the models, methods and users clients. The framework enables different kinds of models to be encapsulated as simulation modules without exposing the actual content of the model. Only a published interface is visible to the outside. The models can also be executed in a distributed fashion which enables models and methods to be executed from their original location. With this approach, both models and methods are managed as generic simulation modules which are integrated and executed as illustrated in Figure 8.

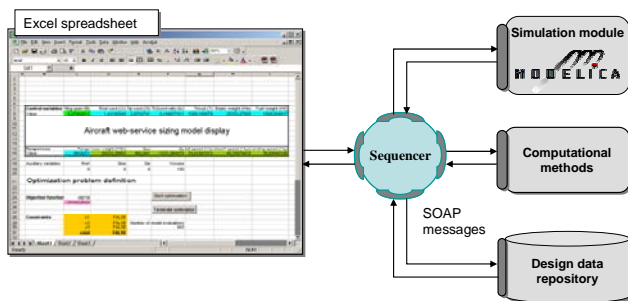


Figure 8. Integration framework where a simulation model implemented in Modelica is integrated with computational methods and a design data repository. Inputs and outputs are here managed using an Excel spreadsheet.

A wrapper is created around the simulation model in order to publish the model as a simulation module as illustrated in Figure 9.

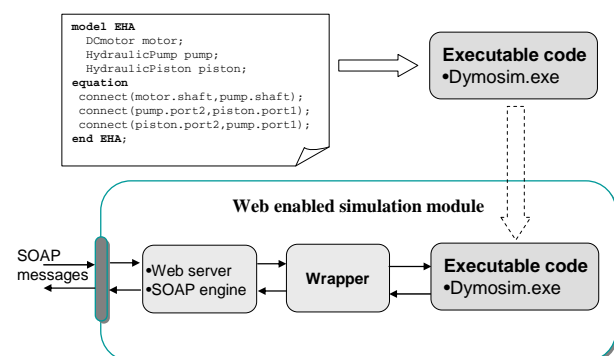


Figure 9. The Modelica system model is translated and compiled using Dymola. The executable code is wrapped as web service simulation module.

In the work presented here, a prototype has been implemented where Matlab constitutes the wrapper that communicates with both the simulation model, and the web service interface. A more permanent solution is however intended where XML technology is

used to dynamically create and parse the input and output files to and from the Modelica simulation directly. This is a very flexible approach which has been implemented in previous projects, see [3].

Important to note is that this for model integration is not intended for high-speed data exchange between tightly coupled models. Rather, it is intended for automation of sequential (or parallel) computational design tasks involving several distributed model. An XML-based process model has also been developed which can be automatically executed by a so-called sequencer. Further details about this framework are presented in [2].

5 Example – Probabilistic analysis of aircraft actuation system

In this section an example will be presented where a probabilistic analysis is carried out using the presented framework and a simulation model developed in Modelica.

5.1 Electro-hydrostatic actuation system

The system is an electro-hydraulic system, principally illustrated in Figure 10. The intention is to mount the system inside the aircraft wing in order to move the control surfaces of the aircraft.

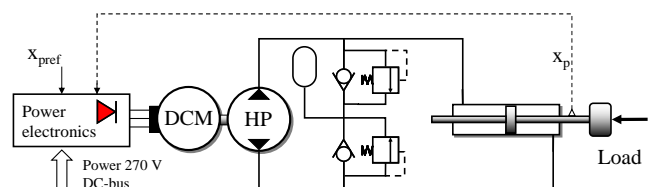


Figure 10. A schematic model of an electro-hydrostatic Actuation system (EHA) implemented in Modelica.

Due to the compact design of the system and the high power density, the system generates heat that can lead to high temperatures and cause damage to the system. It is therefore of interest to analyze the thermal behaviour of the system during missions of the aircraft. In order to accomplish this, a model of both the dynamic performance and the thermal properties of the EHA as well as load forces from authentic missions have been modelled in the Modelica language.

5.2 Simulation model in Modelica

There are different aspects that are of interest when studying actuation systems such as dynamic per-

formance, how the system responds to a control signal, or how sensitive the system is to disturbances.

The models of the system were designed in an object-oriented way where all the components were modelled using the Modelica modelling language. In each component, equations for both dynamic behaviour and thermal properties are included and thermal properties such as temperature and heat flow are represented in the connectors.

The electric motor and the power electronics are also designed to include dynamic as well as thermal properties. Both hydraulic and electric components have equations for thermal properties. Pure thermal components have also been added to the model. In Figure 11, a graphical representation of the model as implemented in Dymola is visualized.

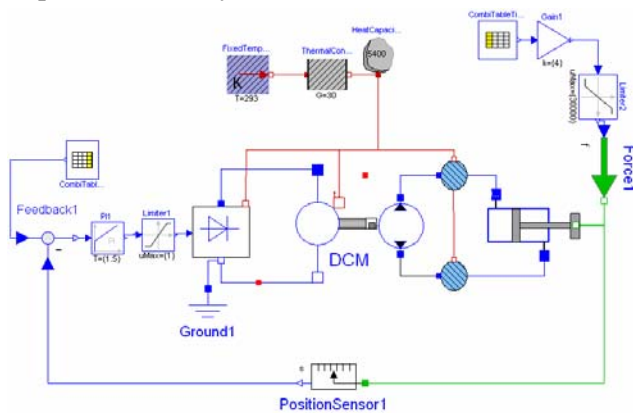


Figure 11. The simulation model as implemented in the Dymola simulation tool.

The system has been simulated in mission of 50 minutes. In Figure 12, results from simulation can be seen. The system was simulated with load and control signals from authentic mission data. The simulation show that high temperatures will occur both in the hydraulic fluid as well as in the motor windings during a so-called extreme mission.

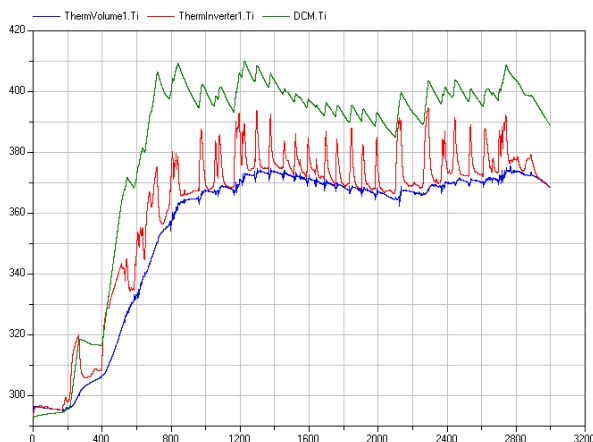


Figure 12. Temperatures [K] in the system during a heavy mission. Simulation of the Modelica model using Dymola. The mission is simulated for 50 minutes (3000 sec).

5.3 The uncertainties

From a design point of view, the system includes several uncertain parameters that could affect the thermal properties in the components. In order to keep the example simple, only three parameters in the model is selected to illustrate uncertainties in the system.

Normal distributions are selected for the resistance in the DC motor and in the power electronics. A normal distribution is also set for at speed dependent thermal parameter in the motor.

Table 1. Definition of uncertain parameters.

System parameter	Mean value	Standard dev
Inverter resistance [Ω]	0.35	0.1
Speed dependent thermal constant [rad^{-1}]	0.5	0.1
Motor resistance [Ω]	2.5	0.25

As an example, the representation of the motor resistance is visualized below. Both graphically, and as XML code.

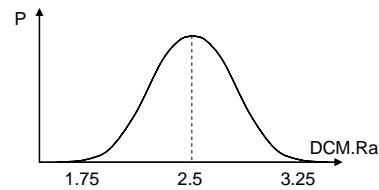


Figure 13. A normal probability distribution defines the resistance of the electric DC motor.

```
<Variable name="DCM.Ra" info="DC motor resistance"
  variableType="NonControllable"
  variabilityType="NormalDistribution">
  <Value>2.5</Value>
  <Distribution>
    <Normal mean="2.5" stdDev="0.25"/>
  </Distribution>
</Variable>
```

5.4 The constraints

A few example constraints are here presented regarding the temperatures in different parts of the system.

- The temperature of the hydraulic oil should not exceed 90°C ,
 - $C_1 = \text{Oil.Ti} \leq 90^{\circ}\text{C}$
- The temperature of the DC motor windings should not exceed 100°C
 - $C_2 = \text{DCM.Ti} \leq 100^{\circ}\text{C}$

The constraints are evaluated in each simulation in order to evaluate the probability of feasibility described below.

5.5 Evaluating probability of feasibility

In the application example, probabilistic analysis is employed on the system in order to investigate the probabilities of meeting the constraints.

The framework illustrated in Figure 8 is here used for the simulations. The simulations are controlled from an Excel document, where inputs to the model can be entered as well as results from the model monitored.

In each execution of the model, the max temperature in the different parts of the system at each simulation is stored. By modifying the inputs according to the probability distributions of the uncertain parameters, variability in the responses is obtained as well.

The results are investigated by computing a Cumulative Density Function (CDF) for the response of interest. By fitting a standard distribution to the values of the responses, the probability of achieving responses that meet the constraints can be computed, see Figure 14.

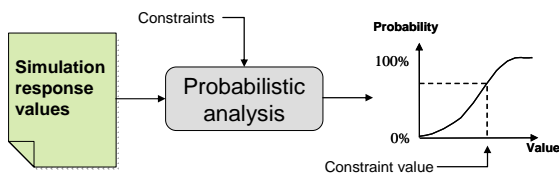


Figure 14. The simulation results are extracted from the XML repository for analysis.

Below, the results for the temperatures of the hydraulic fluid as well as the DC motor temperature are visualized.

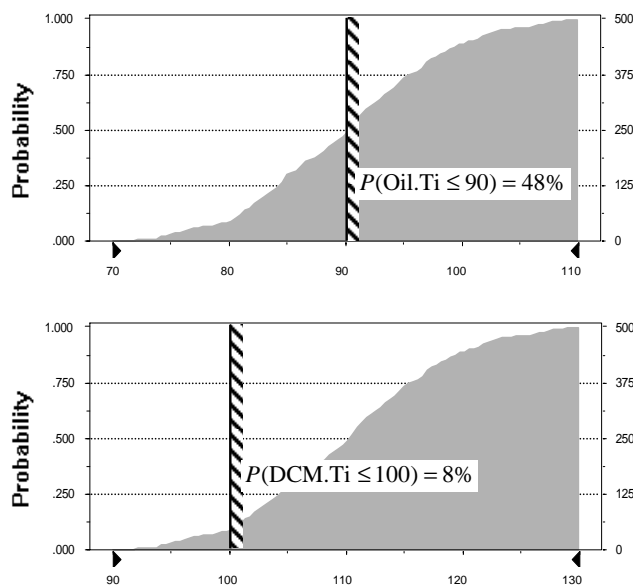


Figure 15. The probability of meeting constraints on oil temperature and DC motor temperature with uncertainty in some system parameters.

For the uncertainties and constraints used in this example, the results are the following probabilities:

- The temperature of the hydraulic oil should not exceed 90°C,
 - $P(C_1) = 48\%$
- The temperature of the DC motor windings should not exceed 100°C
 - $P(C_2) = 8\%$

This implies that the total probability of meeting the constraints (probability of feasibility) is:

$$P(\text{feas}) = P\{(\text{Oil.Ti} \leq 90) \cup (\text{DCM.Ti} \leq 100)\} = 4\%$$

It is obvious that this is too low probability for the system to be robust and we must investigate if the constraint can be relaxed or else we make some change to our design. For the purpose of this example, we now assume that the constraints cannot be relaxed.

Now assume that we infuse technologies to our concept that increases the ventilation of the EHA mounting area and the increases the transportation of heat from the EHA surface. This means that we can assume a *technology factor* that should affect the probability of meeting the constraints.

By modifying our model we can now re-evaluate the probabilistic analysis in the same way as above.

The results in Figure 16 show that the probability of meeting the constraints has increased.

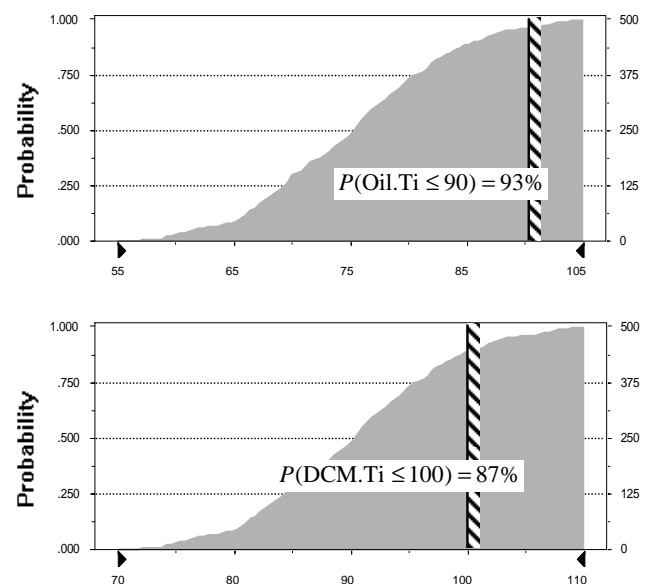


Figure 16. The probability of meeting constraints on oil temperature and DC motor temperature with a modified concept.

The total probability of feasibility for the modified concept is now:

$$P(\text{feas}) = P\{(\text{Oil.Ti} \leq 90) \cup (\text{DCM.Ti} \leq 100)\} \\ = 81\%$$

We can now accept the current concept and move on to the next step in the design process, which includes further simulation and optimization to achieve an optimal design point with respect to both performance and robustness. This is however beyond the scope of this paper.

6 Discussion and conclusions

It is important to realize that in a computational design task, the system simulation model is not the top-level integrator that accesses and integrates different types of data. It is rather a component that is *being accessed* from a design framework at a higher level including some computational method. The information that the simulation model delivers is then evaluated and integrated with results from several types of models.

Simulation models in industry exist in a wide range of representations ranging from old legacy codes represented in Fortran code to modern object-oriented modelling languages such as the Modelica language implemented in simulation tools such as Dymola. It is important that the computational design methods can interact with the models regardless of implementation. With a design data repository implemented in a format that is simple to access by a wide range of tools, this interaction is highly facilitated.

The approach presented in this paper uses XML for representation of the design data in a format that is general and not associated with existing representations of system simulation models. The advantage is that XML is widely supported by a wide range of software tools, and that it is simple to access and manage the XML data.

The example presented in this paper is only one simple illustration of how the simulation model can be used in a computational design task. Increased demands for the product developing industry regarding faster time to market will make design automation more and more important. It is therefore very important to continue to define interfaces between the area of engineering design and the area of system modelling and simulation.

References

- [1] BOX M. J., "A new method of constrained optimization and a comparison with other methods," *Computer Journal*, 8:42-52, 1965.
- [2] JOHANSSON B. AND KRUS P., "A Web Service Approach for Model Integration in Computational Design", in *Proceedings of ASME Computers and Information in Engineering Conference*, Chicago, USA, September 2-6, 2003.
- [3] JOHANSSON B., DELAURENTIS D. A. AND MAVRIS D. N., "Managing Design Data for Probabilistic Evaluation of Aircraft Concepts", in *Proceedings of International Conference on Engineering Design*, ICED'03, Stockholm, Sweden, August 19-21, 2003.
- [4] KRUS P., JANSSON A. AND PALMBERG J-O, "Optimisation for Concept Selection in Hydraulic Systems", in *Proceedings of 4th Bath International Fluid Power Workshop*, Bath, UK, 1991.
- [5] MAVRIS D. N. AND DELAURENTIS D. A., "A probabilistic approach for examining aircraft concept feasibility and viability", *Aircraft_Design*, 3 pp.79-101, 2000.
- [6] MODELICA ASSOCIATION, "Modelica A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, <http://www.modelica.org>, 1999.
- [7] PAPAZOGLU, M.P.: "Service-Oriented Computing: Concepts, Characteristics and Directions" in *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE 2003)*, December 10-12, 2003
- [8] SUH N. P., *Axiomatic Design: Advances and Applications*, Oxford University Press, New York; Oxford, 2001.
- [9] TSALGATIDOU A. AND PILIOURA T., "An Overview of Standards and Related Technology in Web Services," *Distributed and Parallel Databases*, 12, pp. 135-162, 2002.
- [10] WORLD WIDE WEB CONSORTIUM, <http://www.w3c.org>.
- [11] WU Y.-T., "Computational methods for efficient structural reliability and reliability sensitivity analysis" *AIAA Journal*, Vol. 32(8), pp 1717-23, 1994.

Optimization for Design and Parameter Estimation

Hilding Elmqvist¹, Hans Olsson¹, Sven Erik Mattsson¹, Dag Brück¹,
Christian Schweiger², Dieter Joos², Martin Otter²

¹Dynasim AB, Lund, Sweden ({Elmqvist, Hans.Olsson, SvenErik, Dag}@Dynasim.se)

²DLR Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany
(Christian.Schweiger@DLR.de, Dieter.Joos@DLR.de, Martin.Otter@DLR.de)

Abstract

This paper describes new features of the Modelica environment Dymola to perform integrated computer experiments with Modelica models, in particular for model calibration, design optimization and robustness or sensitivity assessment based on multiple criteria and multiple simulation runs. The environment and especially the problem setup are demonstrated by several application examples.

1 Introduction

Recently, new features have been added to the Modelica simulation environment Dymola [2] to simplify experimentation with Modelica models significantly. Some basic ideas are from the optimization environment MOPS [3]. The central part is a Modelica model of a physical system with Modelica parameters that are not yet fixed. Several problem classes can now be conveniently solved:

- **Model Calibration (Parameter Estimation):**
Some Modelica parameters of the model are not known. Several simulation runs are performed and compared with measurement data that is available from equivalent dynamic behavior of the real device. Via optimization, the selected unknown parameters and initial conditions are determined such that the simulations and the measurement data are in good agreement. Also standard tasks such as fitting of functions to measurement data is supported.
- **Design Optimization (Parameter Optimization):**
Selected Modelica parameters of the model are tuned to improve the system dynamics, e.g., by changing the parameters of a controller or some parameters of the physical device. This is performed by multi-criteria parameter optimization using one or several simulation runs to compute the desired criteria.

- **Assessment (Parameter Variation):**
Selected Modelica parameters of the model are systematically changed within a given grid and for every fixed set of parameters, simulations are performed. This might be used, e.g., to evaluate a finished design by varying the operating points. For Monte Carlo simulations, the parameter values are chosen statistically. Simulations with small variations to the parameters can be used to determine how sensitive a design is.

All these experiment tasks utilize the same basic functionality that is defined once:

- **Tuner Parameters:**
Modelica parameters that remain constant for a particular simulation but are varied by the experimentation environment to search for satisfactory solutions are called “tuners”. E.g. via parameter estimation or optimization the tuners shall be determined such that criteria are minimized.
- **Criteria:**
Criteria are used to compute quantitative values of achieved performance of a simulation run. Criteria are assumed to be positive and smaller values reflect better performance. Usually, several criteria are needed to express the desired behavior. For example, typical criteria of a system step response are over-shoot or settling time. By weighting each criterion individually by a fixed demand value ($\text{criterion}_i/\text{demand}_i$), where the demand value expresses the designer’s notion of expected system performance, a clear preference list of the criteria is defined. For example, the demand value for the settling time might be 0.1 s, indicating that a value of 0.1 s is satisfactory from a users point of view.
To solve the multi-criteria problem by means of standard numerical optimization an overall criterion to be minimized has to be formulated. This so called aggregation function is by default the maximum function, yielding a min-max optimization problem over the weighted criteria (= the

largest weighted criterion is minimized). Of course the maximum weighted criterion may change during optimization depending on the tuner values found. Note that an aggregation function value less than 1 implies that all criteria satisfy the demands.

There is an option to choose other aggregation function types like weighted sum. In any case the weights are formed by the reciprocal demand values.

- **Constraints:**
Design specifications are often given as constraints such as actuator limits. If a certain level of compliance is achieved for a criterion, this level must be kept, and smaller (better) criterion values are not necessary. Constraints are formulated as criteria, which are requested to be smaller than the demand value. Optimization procedures account for constraints in their optimization strategy explicitly.
- **Indicator plots:**
A criteria value results in one number to express the performance. This is necessary in order that an optimizer can be used. A human would like to evaluate a design by visual comparison of result plots of different designs, e.g., by viewing a whole step response curve and not only the overshoot value. Indicator plots for a model can be defined once and then reused, e.g., for online visualization of the optimization or parameter estimation process.
- **Model Cases:**
In many applications, several simulation runs are necessary to evaluate a design or to estimate parameters. In the Dymola environment, several simulation runs are collected together to model cases: Exactly the same tuners, criteria and indicator plots are used for each model case. The model cases are distinguished only by a set of fixed Modelica parameters that define the different simulation runs. Often, these case parameters describe different operating conditions, e.g., different road or load conditions of a vehicle.

The paper shows for two application examples how to define the problem setup of tuners, criteria, cases and indicator plots by means of the graphical user interface. The defined problem setups are used to solve the calibration task of an under-actuated two joint Furuta Pendulum and then the parameter optimization for robot control laws. The solution of the multi-criteria optimization problem is discussed briefly in Section 4.

2 Application Examples

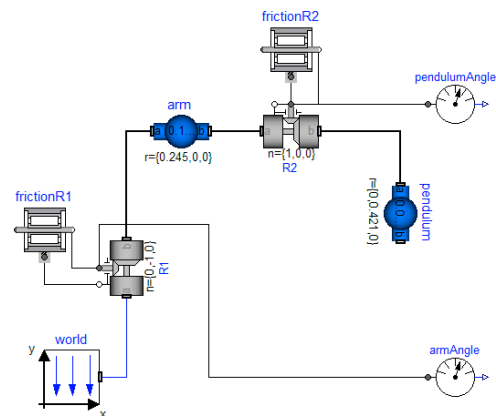
In this section, details of the experimentation environment are described elaborately by several examples.

2.1 Parameter Estimation of an under-actuated two joint Furuta Pendulum

Consider the Furuta pendulum demonstration of the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden. The



pendulum, consisting of 2 revolute joints and 2 moving bodies, is shown in the figure. Only the first joint (vertical axis) is driven by a DC motor. Experiments and controller have been designed and evaluated in [1].



Within Dymola, a model of the Furuta pendulum is easily constructed by dragging, dropping and connecting body and joint components from the Multi-Body library. However, it is also necessary to set physical parameters in the model. Some of these parameters such as the length of the arm or the length of the pendulum are easily measured on the system. Direct measurements of the weights of the parts would require the system to be dismantled (in other cases it is easy to measure the mass or determine it from a CAD system). Moreover, it is not simple to measure the inertia of the parts or the friction characteristics of the two joints.

The new Dymola experimentation environment has been used for parameter identification. For this, the movement of the 2 body pendulum has been recorded by sensors in the joints. The same movements are performed with a simulation model and the friction parameters are optimized such that the measured and the simulated movements closely agree.

Selection of parameters to be tuned

When estimating parameters from measurements, a basic question is “Which parameters can be estimated from the measurements?” Changing a parameter to be estimated must of course influence the output. However, this is not enough if several parameters are to be estimated. Consider the arm of the Furuta pendulum. It rotates around a vertical axis. The model of the arm uses

Modelica.MultiBody.Parts.BodyShape

It has the following parameters: mass, position of center of mass and inertia tensor with respect to center of mass. Since the arm is rotating around a fix axis, it is only the inertia with respect to this axis that influences the behavior of the system. Inertia with respect to an axis being orthogonal to the vertical axis does not influence the motion at all. The criterion is independent of its value. We will discuss a more general case. Let J_c denote the inertia with respect to a vertical axis through the center of mass. Let m denote the mass of the arm and r_c denote the distance between the point of rotation and the center of mass. The inertia of the arm with respect to the point of rotation, J_a , is then

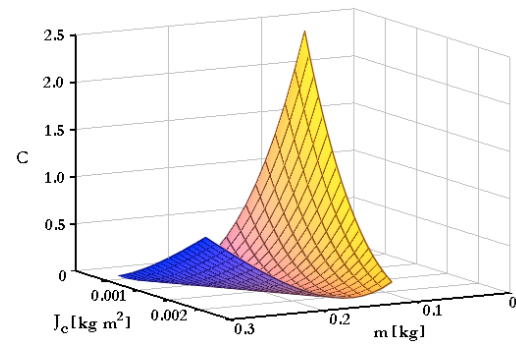
$$J_a = J_c + m \cdot r_c^2$$

It is this inertia that is of importance for the rotation of the arm. It is this parameter that can be estimated from measurements. It is not possible to estimate m , J_c or r_c uniquely from measurements of positions or velocities.

What will happen if we try to estimate J_c and m ? In the best case we will get a good estimate of J_a , but J_c and m may be really different from their real physical values. One approach to investigate if we are about to estimate too many parameters is to make estimation experiments on the model. First one could use the model with its nominal parameter values to produce simulated “measurement data”. Then one would set the model parameters that are going to be estimated to other values, and run the calibration procedure using the previously produced “measurement data”. If the estimated parameters give the same simulation results but are significantly different from the “true” nominal ones, then this indicates overparametrization, because the nominal behavior can be reproduced by other parameter values than the nominal ones used to produce the “measurement data”.

Assume $r_c = 0.1225$ and being known. Assume the nominal values $J_c = 0.0014$ and $m = 0.165$. Assume the criterion to be the integrated squared error of the pendulum angle and arm angle. Let the pendulum start in horizontal position and use the simulation

time 5 s. It is illustrative to plot the criterion versus J_c and m as shown below.



We see a valley where the criterion is unchanged. Along the valley the effective inertia, J_a , is unchanged with peaks on either side where it has been changed. The error is not symmetric with respect to large variations in J_a and the error increases more when J_a decreases.

To compute this map we used a gridding function that takes the calibration task (as described below) and additionally the gridding parameters as inputs, i.e. we do not have to define the calibration task twice.

A further possibility is to investigate the sensitivity of the criterion with respect to the parameters estimated. In particular we can calculate the sensitivity matrix (the Hessian of the minimization problem). Dymola calculates the sensitivity matrix by simulating for disturbed parameters and taking differences of the resulting criterion values obtained. The sensitivity matrix was found to be

$$\begin{bmatrix} 431064, & 6456.3; \\ 6456.3, & 97.0671 \end{bmatrix}$$

The eigenvalues being 431161 and 0.37 are very different in magnitudes. Considering the numerical accuracy, the small eigenvalue may be considered as being zero. The eigenvector having the large eigenvalue is $\{0.99989, 0.01498\}$. It means a large sensitivity in the direction

$$\{0.99989, 0.01498\} * \{J_c, m\}$$

Recall $J_a = J_c + 0.01501 * m$ which shows that the criterion is sensitive for variations in J_a . The second eigenvector $\{-0.01498, 0.99989\}$ is orthogonal. (For a symmetric matrix, all eigenvalues are real and eigenvectors are orthogonal.) The small eigenvalue, being very close to zero, indicates that the measured behavior is insensitive to variations in the direction of the second eigenvector as is also shown in plot above.

Since the optimization problem is a non-linear least-squares problem, $\sum f_i^2(p)$, we can alternative com-

pute the insensitive direction (in a more numerically robust way) as the approximate null-space of $\sum \partial f_i(p) / \partial p$.

It may be remarked that for the case where the criterion is independent of a parameter, there will be a valley parallel to the axis representing the parameter and the sensitivity matrix the vector in the direction of the parameter will be an eigenvector with zero eigenvalue.

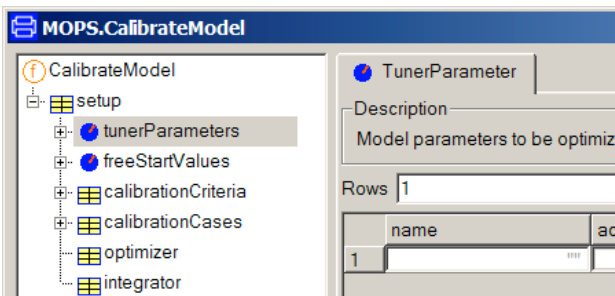
Thus for the arm we set $r_c = 0$ and interpret the estimated inertia as being J_a . It means that we introduce a top-level parameter I_{arm} and set $arm.r_{CM}$ to $\{0,0,0\}$, and $arm.I_{22}$ to I_{arm} .

The pendulum consists of a cylinder having a small mass at the end. It is natural to assume its inertia with respect to all axes perpendicular to its length axis to be equal, call it $I_{pendulum}$. Set $pendulum.I_{11} = I_{pendulum}$ and set $pendulum.I_{33} = I_{pendulum}$. The inertia sensed by the rotating arm depends on the angle of the pendulum, which means that for the pendulum, we can estimate also mass ($pendulum.m$) and position of the center of mass, $r_{CM_pendulum}$. We set $pendulum.r_{CM}$ to $\{0, r_{CM_pendulum}, 0\}$.

Parameters in the friction model for the joints can also be estimated. We assume Coulomb friction with a linear dependence on velocity. For the arm joint we introduce τ_{11} and τ_{12} and set $frictionR1.\tau_{pos}$ to $[0, \tau_{11}; 1, \tau_{12}]$ and similarly for the pendulum joint.

Setting up the calibration task

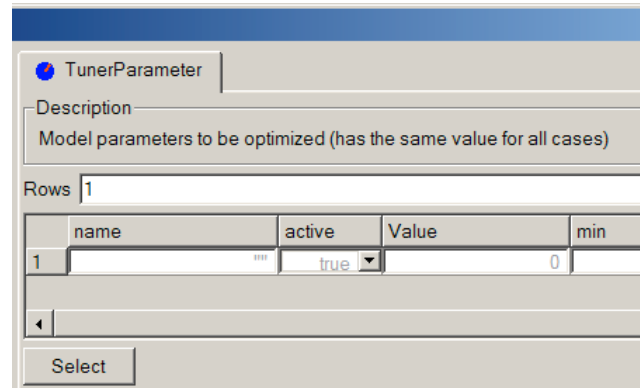
To set up the calibration task, select CalibrateModel from optimization package. Click the right mouse button. Select Call Function. A dialog is shown:



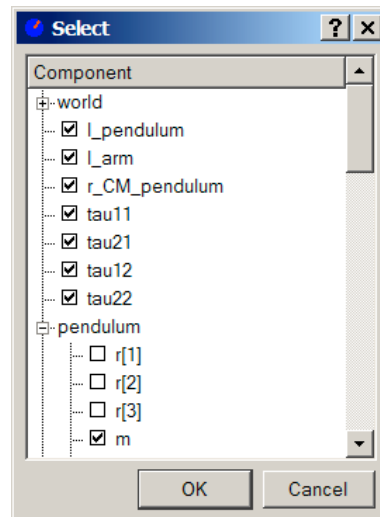
It shows that the calibrate function has one argument, setup. Clicking on the “+” opens it and shows that it is a record with five elements, which describes various aspects of the calibration task including which parameters to tune, criterion and which measurement data to use. We will discuss the specification of these elements in turn. The calibrate function assumes the current model (last translated

model). This allows easy reference to parameters when selecting tuners as described below.

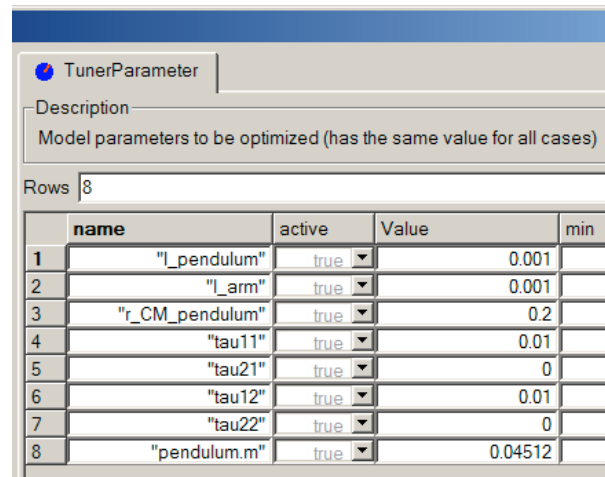
To specify which parameters to tune, click on the tunerParameters and the right pane of the dialog shows



Clicking on Select gives a browser for simple selection where we tick the parameters as decided.



Clicking OK fills the tunerParameter dialog as shown in the following image.

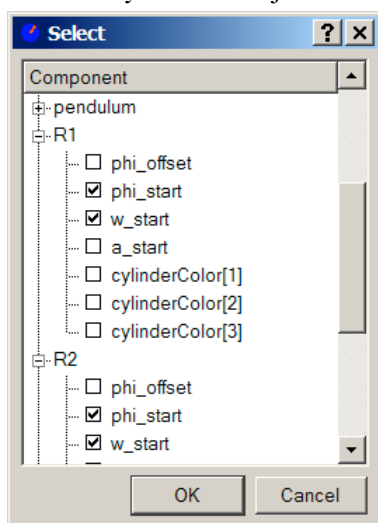


Estimating initial conditions

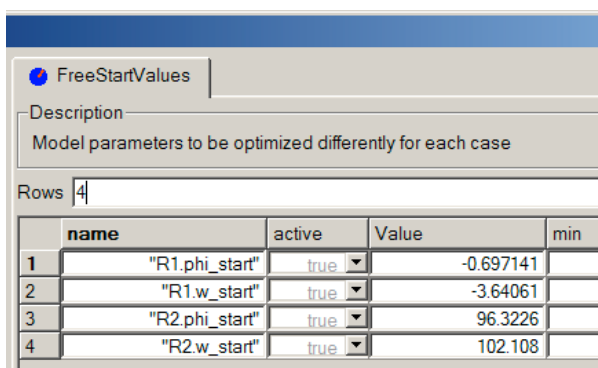
The initial conditions will be different for different experiments, for example, if we let the pendulum start at different angles. To get good fits it is advisable to estimate the initial conditions. We can give reasonable guesses, because we try to start the system in a well-defined state. For some experiments the initial conditions are known accurately and they should then be given as part of the setup of the calibration cases, but this is not the case here.

The joint models allow specification of initial conditions in terms of parameters. These parameters can be tuned. For each of the parameter discussed previously we would like to have a common tuned value for all cases.

However, for the initial conditions we need individual estimation for each case. Moreover, we would also need them to be tuned for the evaluation where the parameters tuned are kept fixed. The initial conditions to be estimated for each measurement case are specified by the element `freeStartValues`. Select the element `freeStartValues` in the tree browser. It is specified in the same way as done for `tunerParameters`. In the select dialog we tick the start values for the angle and velocity of the two joints.

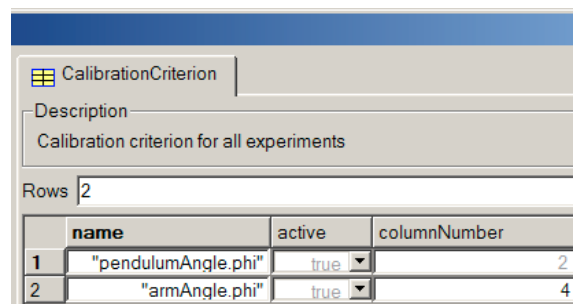


Clicking OK fills in the variable names in the first column and start values in the “Value” column.



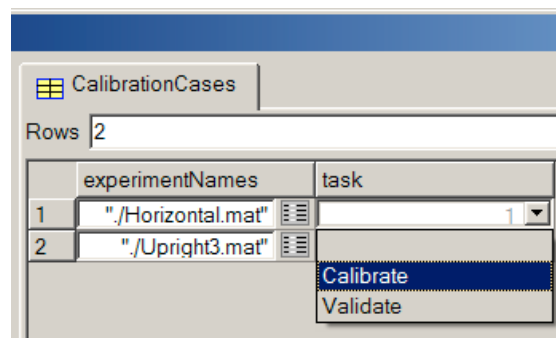
To specify the criterion, select `calibrationCriteria`.

Clicking on Select displays a Select browser where we check `pendulumAngle.phi` and `armAngle.phi`. This fills the “name” column.



The criterion is a weighted sum of the integrated square difference with respect to measured value for each variable.

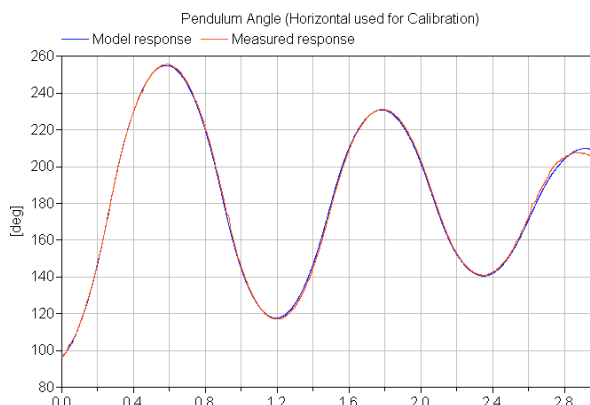
The measurement files to be used are specified by the element `calibrationCases`. It also specifies whether a file is to be used for calibration or validation.

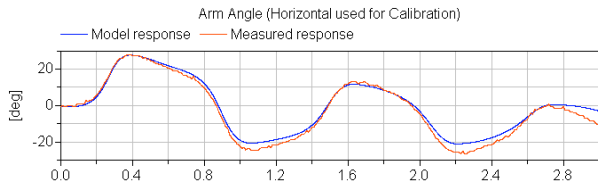


The elements `optimizer` and `integrator` allows advanced setting of optimization and simulation parameters. We use their default settings, except for simulation time that we need to specify.

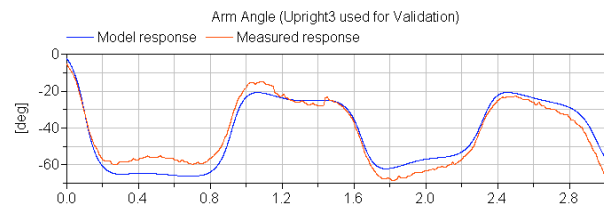
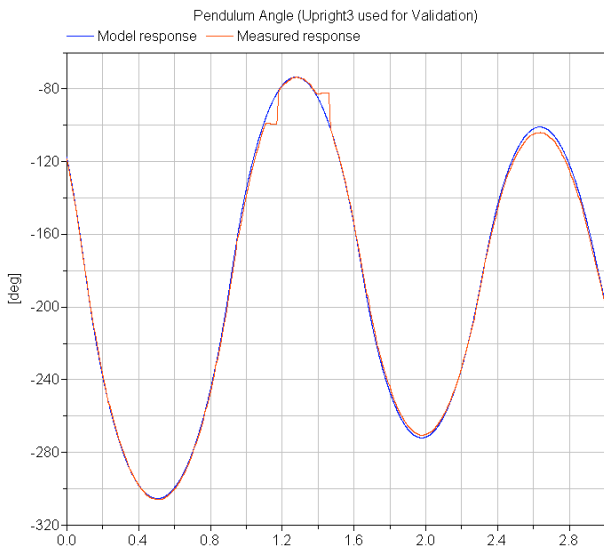
Running the calibration

When all the input data have been entered, the calibration is started by clicking “Execute” on the dialog. The two next plots compare the simulation result with the obtained tuned parameters and measured data where the pendulum starts in a horizontal





position. The result shows a good agreement. The tuned parameters were validated against another experiment where the pendulum starts in a more upright position. The system is nonlinear and it is of interest to validate the model for cases where the amplitudes of the pendulum are large.



The jump in the pendulum angle in the validation case between -100° and -80° is due to problems of handling wrap-around in the measurement device. The agreement is good, in particular for the pendulum motions. The modeling of the arm may be improved by a more elaborate modeling of the arm friction. However, that is out of the scope of this paper.

2.2 Robot optimization

In this section it is demonstrated how to optimize the controller parameters of a robot for a set of cases consisting of different loads and reference motions.

Description of the robot

The model is chosen as the r3-robot from the Modelica standard library. As described in its documenta-

tion this was originally a Manutec r3-robot. It was then updated with incorporating CAD-data from a KUKA-robot, and the geometry was modified to fit the CAD-data.

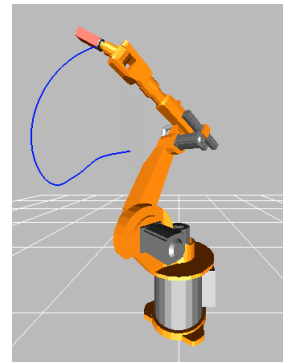


Figure 1 Animation of the robot

The robot consists of a 6 degree-of-freedom mechanical structure modelled as the multibody system “mechanics” (= bodies connected together by revolute joints). The calculation of the animation can be optionally switched off to increase simulation speed, which is especially important during an optimization run. Every joint is driven by a drive train called “axis1”, “axis2”, ..., in the next figure:

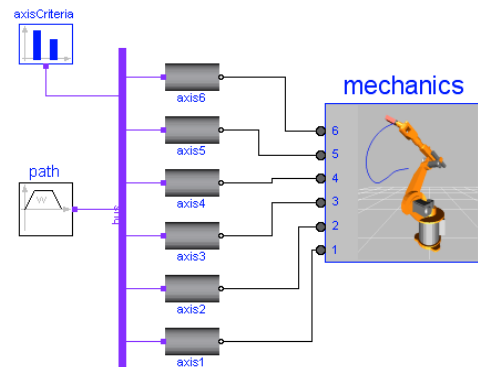


Figure 2 Composition diagram of robot

The desired reference motion is generated in component “path” and as input it has the start and end angle of each axis. All signal data is communicated via a “data bus”.

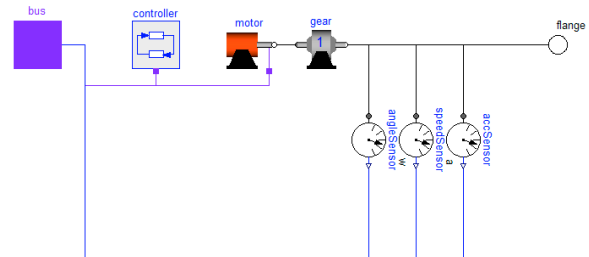


Figure 3 Drive train of one axis

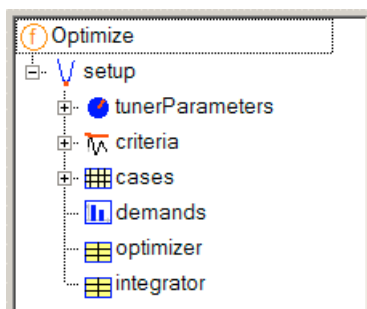
One drive train consists of a controller, an electrical motor, a gearbox (which includes the bearing friction). The controller of one axis is a P-PI cascade controller, i.e., has the simplest form useful for a servo drive (in most industrial robots, more sophisticated controllers are used). The goal of the optimization is to tune the values of the 3 controller parameters of every axis:

- k_p – gain of position controller
- k_s – gain of speed controller
- T_s – integrator time constant of speed controller

The difficulty is that this controller should work well with a fixed set of values for all paths and operating conditions encountered by the robot. This makes it impossible to use standard, linear control design methods.

Optimization setup

The structure of the optimization setup is similar to the calibration setup:



The tuner parameters are the 3 controller parameters of each axis as discussed above. They are easily entered in the tunerParameter dialog by using the Select dialog. The snapshot below shows the dialog when the controller parameters of axis 2 have been selected. Numeric values have also been entered for “Value” to be used as the start of the optimization and minimum and maximum values that are used as box constraints during optimization and optionally also for scaling:

TunerParameter						
Description						
Model parameters to be optimized (same value for all cases)						
Rows 3						
	name	active	Value	min	max	autoScale
1	"kp2"	true	2	0.01	20	true
2	"ks2"	true	1	0.01	10	true
3	"Ts2"	true	0.1	0.001	1	true

In order to specify the different cases of the optimization, the case parameters are first selected by using

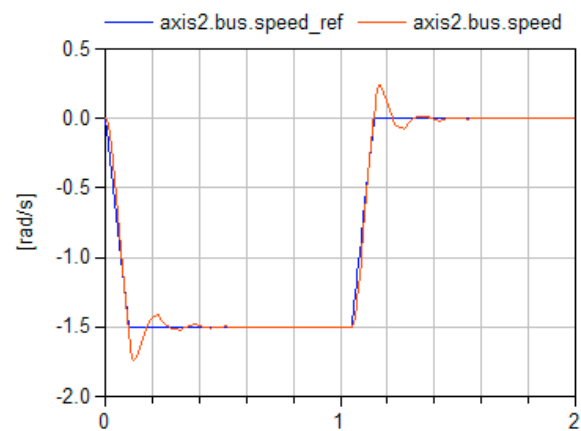
the Select dialog to browse the model parameters. The parameter values for the different cases used to optimize the design of the controller are then given by filling out the cases form as below (the selected parameter names appear as column headings):

Cases							
Rows 4							
	Case	Active	startAngle2	startAngle3	endAngle2	endAngle3	mLoad
1	"High 1"	true	90	0	0	0	15
2	"Low 1"	true	90	-90	0	-90	0
3	"High 2"	true	0	90	30	90	15
4	"Low 2"	true	0	0	30	0	0

Simulation cases are defined by specific settings of model parameters that are given as labels “startAngle2”, “startAngle3” etc. in the figure above. Note that in the reference trajectory axis 2 goes through different movements in the different cases, whereas axis 3 is fixed in different positions (all other axes are fixed to the default reference angle in all cases). In practice, robots are optimized for a larger number of cases.

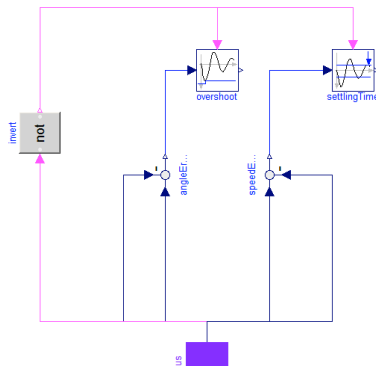
Optimization criteria

If we compare the actual axis speed with the reference speed we normally get the following:



For design optimization the goal is in general *not* to simply minimize these errors, as it would be for a calibration, but something more advanced.

For the design optimization we thus have to introduce additional blocks to measure the performance of the axis, these are introduced by extending the robot-model with an additional performance component for the reference signals (containing different performance indicator blocks).

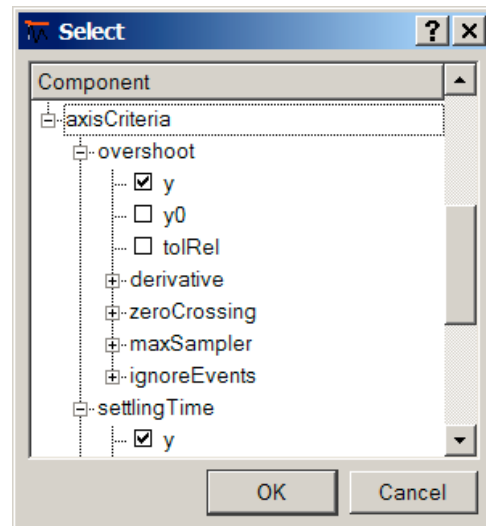


In the performance Modelica model, the following performance indicator blocks are present (these blocks are from a criteria library that contains several predefined useful criteria blocks):

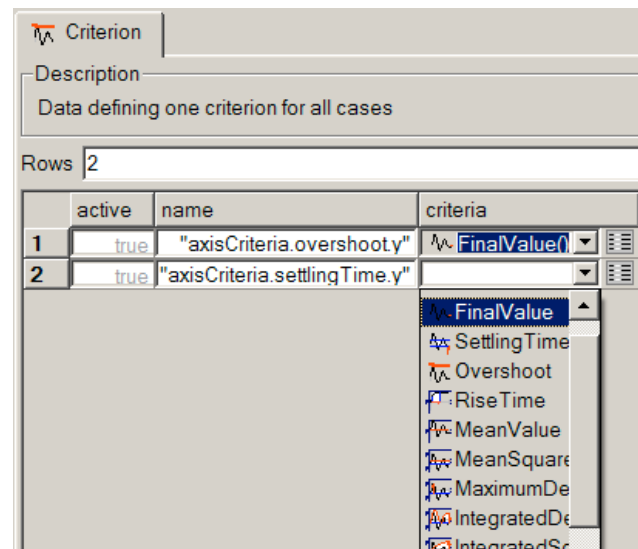
- Block “Overshoot” computes the overshoot (here: the maximum deviation from the reference angle) after the referenced motion has come to a rest. This is performed very precisely by triggering an event whenever the derivative of the input signal is zero (here: whenever the speed is zero), i.e., a minimum or maximum of the signal is reached and storing the corresponding signal, if its absolute value is larger than the previously stored value. The last stored value is the overshoot which shall be minimized.
- Block “SettlingTime” computes the time until a signal stays completely within a tolerance band around zero, after the reference motion is in rest. This is performed by triggering an event whenever the input signal passes through this band and storing the corresponding time instants. The last stored value is the settling time which shall be minimized.

The usage of these two blocks is shown above. As can be seen the angle error is used as input to the “overshoot” block whereas the speed error is used as input in to the “settlingTime” block.

This is connected to the bus to get the reference signals and the actual values for the specific axis. We then select two reference indicators, the overshoot and the settling time, in the optimizer:



As an alternative to using criteria blocks in the model one can compute the criteria in a post-processing function that operates on the simulation results and which is selected in the drop down menu shown below. In this case we select the final value for these two criteria:



We also have to set the demand values. Based on first simulations and specifications we set the overshoot demand to 3/1000 [rad/s] and the settling time to 0.3 [s]:

	axisCriteria.overshoot.y	axisCriteria.settlingTime.y
High 1	0.003	0.3
Low 1	0.003	0.3
High 2	0.003	0.3
Low 2	0.003	0.3

Applying these values as demand values results in the following weighted criteria:

case	overshoot	settling time
High 1	0.6854	0.7588
Low 1	1.5106	0.9855
High 2	0.8332	0.5579
Low 2	1.7887	1.0468

The values indicate satisfactory behavior in cases with load (cases “High 1” and “High 2”), but bad overshoot performance in unloaded cases (cases “Low 1” and “Low 2”).

After finalizing the optimization setup, a click on “Execute” starts the design optimization. As a result of the first optimization run, which converges after 30 function evaluations, we obtain the following tuner and corresponding weighted criteria values:

kp2	3.0776
ks2	2.5089
Ts2	0.08896

	overshoot	settling time
High 1	0.5929	0.8998
Low 1	1.0315	1.0902
High 2	0.6510	0.7157
Low 2	1.1956	1.1955

Overshoot has been improved but settling time is slower. The equal and largest criteria values in case “Low 2” indicate a conflict between the 2 criteria which usually can only be solved when one criterion is eased off.

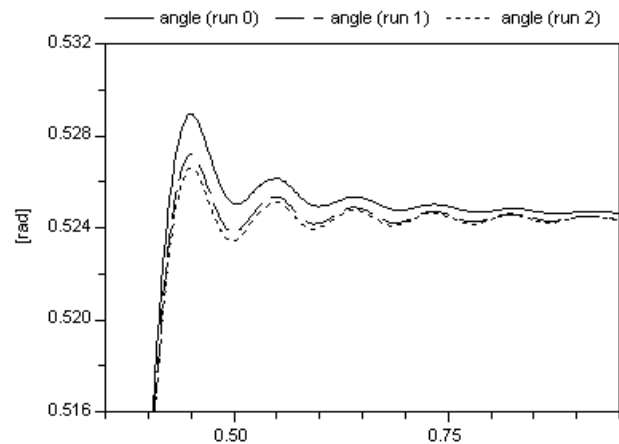
We decide to force the overshoot criteria that they reach their demand values and to put lower emphasis on settling time. This is accomplished by applying all overshoot criteria as *inequality constraints* during the next optimization run forcing the optimizer to perform improvements in these criteria until the demand value is reached. Criteria to be minimized (here settling time) may increase. The next run results in:

kp2	4.0722
ks2	4.8897
Ts2	0.070565

	overshoot	settling time
High 1	0.7623	0.8965
Low 1	0.8845	1.1855

High 2	0.6175	0.7161
Low 2	0.9985	1.2027

The overshoot demand is satisfied, The settling time slightly increases to 1.2027 for the worst case “Low 2”. We might stop the design optimization here. In other cases, one might change demands, select other simulations cases and criteria. In the next figure, results for the 3 runs (initial, first and second optimization run) are shown for case “Low 2”:



3 Customizable user interfaces

Experimentation includes operations that require rich interfaces to supply all the information needed to perform the task.

For model components, parameters have been visually split into groups and tabbed pages, representing logical grouping of primary and secondary parameters. However, the individual data items comprise a relatively “flat” structure.

For calibration and optimization the interface contains a much deeper hierarchical structure, and the complexity at each level is also greater. For example, it is common that subitems contain variable amount of data, typically represented by arrays of records.

To handle the increased complexity, the graphical user interface of Dymola has been extended in two dimensions:

- The nested structure is visualized by a tree, which makes relationships easier to understand and allows easy navigation between data items.
- Specialized GUI elements, for example, for file and color selection, can be enabled by annotations, which facilitate common input operations.

Several of these improvements are useful also for simpler data structures, and the specialized GUI elements can also be used for parameters of models. In

other words, all the features discussed below can be easily utilized by every user. It is even possible to make a copy of the calibration and/or optimization setup and adapt the user interface to the specific needs of an end-user with more specialized menus.

3.1 Nested structures

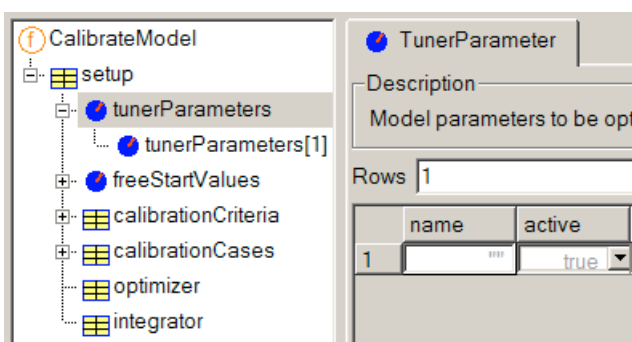
Model calibration will be used as an example. The function CalibrateModel takes a record (setup) as its input parameter. The structure of the record contains among other elements an array of type TunerParameter, which in turn contains several attributes.

```
function CalibrateModel
  "Calibrate model to measured data"
  input ModelCalibrationSetup setup;
  ...
end CalibrateModel;

record ModelCalibrationSetup
  String Model;
  TunerParameter tunerParameters[:];
  ...
  Optimizer optimizer;
  Integrator integrator;
end ModelCalibrationSetup;

record TunerParameter
  "Model parameter to be optimized ..."
  String name="" "Full name of ...";
  Boolean active=true "true, if ...";
  ...
end TunerParameter;
```

The GUI is automatically built from the data structure declarations. The nested structure of the input to CalibrateModel is evident in the tree view at the left.

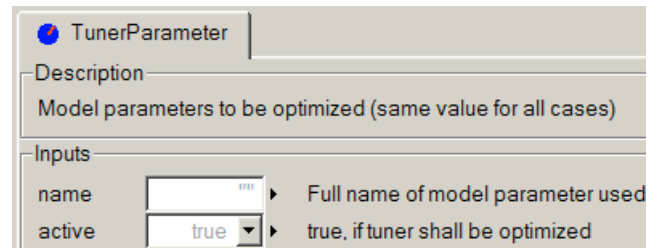


The tree serves two purposes. First it shows the structure and makes it easier to understand what information must be provided. The tree view corresponds exactly to the data structure. Second, it is used to navigate between multiple “pages” (input forms) that are swapped into the space at the right.

The component tunerParameters is an array of records, each containing several variables. The user can choose a combined tabular view of the array (as

shown above), which offers maximum overview in a compact format.

Alternatively it is possible to inspect and edit individual array elements, which has the advantage of displaying descriptions for each input field and that data can be grouped and put under different tabs. Each page corresponds to one row in the combined view.



The implementation of the tree view also ensures that data filled out by the user is propagated. For example, changes to an individual tuner parameter must be visible when the user switches to the tabular view of all tuner parameters. Changes in a modifier at a high level is propagated down to more detailed views.

3.2 New GUI elements

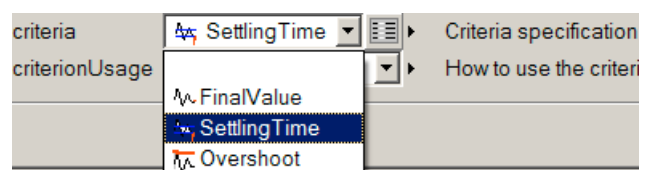
The graphical user interface can greatly simplify certain input tasks with some additional support. Although it is always difficult to strike a balance between features and complexity, the following operations have been found useful in the experimentation environment.

The deployment of these GUI elements is controlled by model annotations, either at the class level or on individual variables.

Predefined choices. A list of values suitable to a particular type or application are presented. A simple example is “true” and “false” for Boolean. Selection of a criteria function is specified by this annotation:

```
CriterionSpecification criteria
  "Criteria specification"
  annotation (choices(
    choice=FinalValue(),
    choice=SettlingTime(),
    choice=Overshoot()));
```

To the user the choices are presented in a drop-down combobox:



Because parameters which depend on the selected function must also be specified, pressing the “edit” button will display a dialog for the parameters to the chosen function.

Color selection. Colors can typically be represented by RGB (red, green, blue) or HSV (hue, saturation, value) tuples. Although they could be specified numerically by the user, a colorful dialog makes selection much easier.

File selection. Standard dialogs for selecting files either for reading or writing data. The filename is stored in the corresponding variable.

Variable selection. Several operations in the experimentation environment involves the selection of variables from the model, for example, parameters to optimize. A specialized selection dialog simplifies the task considerably. Furthermore, additional data, such as, start/min/max values can be extracted.

In this case a detailed specification (in the form of an annotation) is needed to move data into the right elements of a table, and if needed resize the table.

User-defined labels. The default labels used in the tree view or in the combined tabular view are constructed from variable names found in the data structure. By use of annotations, other labels can be specified or even extracted from actual data in the structure.

4 Solving the Multi-Criteria Optimization problem

In a multi-criteria optimisation problem setup all weighted criteria $q_{ij} = c_{ij} / d_{ij}$, $ij \in S_{min}$ can be combined to a vector \mathbf{q} , where S_{min} denotes the set of all criteria (i) to be minimised defined in all simulation cases (j). In order to decide whether a solution \mathbf{q}^I is better than a solution \mathbf{q}^{II} , these vectors should be completely comparable. However, comparing each vector component individually, some components can be better, others can be worse. To make criteria vectors completely comparable a vector norm has to be introduced.

We prefer to use the max-norm, because weighted criteria with positive ‘the smaller the better’ values and quality limiting demands as upper bounds yield a most visible comparative satisfaction assessment of design alternatives in case of the max-norm. Define for all weighted criteria

$$\alpha := \max_{ij} \{q_{ij}\}, \quad ij \in S_{min},$$

then requirements' satisfaction of a design alternative (II) is said to be *better* than of a design alternative (I) if $\alpha^{(II)} < \alpha^{(I)}$. If $\alpha \leq 1$, the design alternative is called a satisfactory solution, because in that case each criterion is less than the respective demand value. In particular, a *best possible* design alternative is characterised by $\alpha^* = \min \{\alpha\}$, yielding the overall constraint optimization problem which can be solved by standard optimization methods:

$$\begin{aligned} \min_T \max_{ij \in S_{min}} \{c_{ij}(T) / d_{ij}\} \\ c_{ij}(T) \leq d_{ij}, \quad ij \in S_{inequality} \\ c_{ij}(T) = d_{ij}, \quad ij \in S_{equality} \\ T_{min,k} \leq T_k \leq T_{max,k} \end{aligned} \quad (1)$$

The disadvantage of this approach is the lack of differentiability of the aggregation function. Thus, methods relying on gradients (like SQP methods) can encounter difficulties in this case.

To overcome the problem of differentiability we provide two mechanisms: an exponential approximation of the max-function yielding a smooth overall criteria and a reformulation by ‘equivalent constraints’. In the latter case the min-max problem can be reformulated by an equivalent constrained problem. Let γ be a new variable for which we impose that $\gamma \geq \max\{q_{ij}(T), ij \in S_{min}\}$. Instead of (1), we can solve an equivalent optimization problem with the extended parameter vector $x = [T, \gamma]$. The aggregation function to be minimized is simply $\alpha(x) = \gamma$.

Defined inequality and equality constraints are applied as in (1) while the components to be minimized are added as additional constraints as

$$q_{ij}(x) \leq \gamma, \quad ij \in S_{min}$$

The main advantage of this formulation is that the functions are differentiable provided the defined problem criteria are differentiable. The disadvantage is that a problem of higher dimension is solved and additional constraints are added. However, the application of this formulation of the min-max problem is recommended whenever a gradient based optimization method is used. There is an option to choose other aggregation functions like weighted sum:

$$\bar{\alpha} := \sum_{ij} |q_{ij}|, \quad ij \in S_{min}$$

Optimization Methods

At the moment 5 different methods to solve the optimization problem (1) are implemented for use with Modelica:

1. Sequential quadratic programming (SQP)
2. Quasi Newton (Bounds)
3. Pattern search (Pattern)
4. Simplex method (Simplex)
5. Genetic algorithm (GA)

The *Sequential Quadratic Programming* (SQP) approach can be used to solve the general optimization problem and has usually a super-linear convergence (= faster than linear, and slower than quadratic convergence). Bounds on tuners and linear equality and inequality constraints are met exactly during the iterations. The SQP approach needs gradients of functions and constraints. SQP in combination with the reformulation of the min-max optimization problem as an equivalent constraint problem is the method of choice for general optimization or calibration problems.

The *Quasi Newton method* (Bounds) is an algorithm intended to solve large optimization problems but can only handle simple bounds constraints on the tuners. "Bounds" needs also gradient information of the aggregation function.

The *Pattern Search* approach is a derivative free search method. It is numerically more robust in tackling with non-smooth criteria than other methods.

The *Simplex* approach is also a derivative-free algorithm and employs linear approximations to the objective and constraint functions. The main advantage of SIMPLEX over many of its competitors is that it treats each constraint individually when calculating a change to the variables, instead of lumping the constraints together into a single penalty function. A drawback of this method is that even bound constraints can be violated during computation.

The *genetic algorithm* (GA) is a global optimization technique. The basic algorithm allows only simple bounds on the variables. Thus, to address more general constraints, penalty function techniques are employed. The genetic algorithm search method is based on evolution principles which guarantee the survival of the fittest individual. The use of GA for optimization is normally quite costly in terms of function evaluations.

5 Conclusions

An environment was presented to optimize Modelica models in Dymola, especially with regards to design optimizations and calibration of unknown model parameters.

It is possible to prepare a customized GUI in Dymola for specific tasks such as optimization. This makes it possible to customize the optimization menus. Since the customization is performed in Modelica (annotations) it is possible for an end-user to also adapt this to his/her particular needs.

Acknowledgements

Measurements and calibration of the Furuta pendulum was performed by Marco Bracci as a part of his master-thesis project at the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

References

- [1] Åkesson J. (2000): **Safe Manual Control of Unstable Systems**. Master Thesis, ISRN LUTFD2/TFRT--5646—SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.
- [2] Dynasim (2005): **Dymola - Users' Manual**
- [3] Joos H.-D., Bals J., Looye G., Schnepper K., Varga A. (2002): **A multi-objective optimisation based software environment for control system design**. Proc. IEEE International Conference on Control Applications, Glasgow, Scotland, Sept. 18-20, pp. 7-14.

Nonlinear Inverse Models for Control

Gertjan Looye, Michael Thümmel, Matthias Kurze, Martin Otter, Johann Bals
 DLR Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany
 {Getjan.Looye, Michael.Thuemm, Matthias.Kurze, Martin.Otter, Johann.Bals}@DLR.de

Abstract

A general technique to design advanced controllers for non-linear systems is described, using component oriented modeling and symbolic algorithms as used for Modelica models. Starting point are linear design techniques that use linear inverse models as a core part of the controller structure. Starting from such a structure, the approach is to replace the linear inverse model with a nonlinear one, resulting in a controller that is applicable over the full operating range of the (nonlinear) plant. It is shown that nonlinear inverse models may be automatically generated from the plant model in Modelica.

1 Introduction

The subject of this article is the systematic design of controllers for *nonlinear* systems, based on inversion of the plant model. Traditional design techniques require the nonlinear plant model to be linearised around a stationary operating point, after which linear methods may be applied to synthesize a controller. In order to make this controller work over the full operating range of the plant, robust design techniques and/or gain scheduling are applied. The first approach may considerably reduce achievable performance if the plant dynamics vary strongly over the operating range, whereas the latter may involve designing many controllers at a grid of operating points and finding an interpolation scheme in between them.

In linear design, inversion of plant dynamics is sometimes used to compensate for coupled input / output responses, or as an easy way to impose specific dynamic behavior of the closed-loop system [7]. Provision is that the linear plant model is minimum phase and, for some structures, stable. In a nonlinear context, the application of model inversion additionally provides compensation of nonlinear dynamic

behavior of the plant. This is exploited in design techniques such as feedback-linearization [19].

The design approach in this article starts from any controller structure that is based on a linear inverse model of the plant. This model is replaced with a nonlinear inverse one, resulting in a controller that is valid for the full operating range of the plant. In case the plant model is available in Modelica, it will be demonstrated that inversion can be performed automatically, exploiting symbolic algorithms and code generation features of a Modelica simulation environment. This allows for a highly automated design process that directly results in nonlinear controllers that work in all operating conditions of the plant, avoiding the need for gain scheduling.

This article is structured as follows. First general aspects of nonlinear inverse models are reviewed, as well as the possibility to derive these automatically from Modelica. In section 4, a number of common controller structures are discussed, for which the described design approach is applied. Next, a design example will be discussed. In section 6 a number of common problems in deriving and applying nonlinear inverse models will be described, as well as possible solutions or workarounds.

2 Inversion of Nonlinear Models

The goal is to use a nonlinear plant model in a controller in order that the nonlinearities of the plant are directly taken care of in the control system. For linear systems, several control structures are known where an inverse plant model is part of the controller. A single-input-single-output plant might be described as transfer function

$$y = \frac{n(s)}{d(s)}u \quad (1)$$

where “u” is the plant input, “y” is the plant output, “n(s)” is the numerator and “d(s)” is the denominator

of the transfer function. In section 4 several control structures will be investigated where an *inverse model*

$$u = \frac{d(s)}{n(s)} y \quad (2)$$

is part of the controller. Basically, the plant and the inverse plant model “cancel” each other due to the connection structure and by additional control blocks a desired transfer function of the closed loop system can be achieved. Although, this seems to be quite a “brute” force method, it will be shown that by appropriate adaptations practically useful control systems can be designed.

The essential idea is as follows:

1. Take any control structure for linear systems that utilizes a *linear inverse* plant model.
2. Replace the linear inverse plant model by a more detailed *nonlinear inverse* plant model.
3. Determine the remaining part of the control system by appropriate techniques, e.g., by tuning controller coefficients via parameter optimization.

Several different controller structures according to this technique will be discussed in section 4. The difficult part is issue (2): The nonlinear plant model should be constructed in a convenient way and the inverse model should be directly derived from the plant model. It turns out that Modelica is very well suited for this approach because Modelica is designed to model complex systems, and since Modelica tools, like Dymola [5], can generate nonlinear inverse models *automatically*:

A continuous Modelica model is primarily mapped to a DAE (= set of Differential Algebraic Equations) of the form:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, \mathbf{u}) \quad (3)$$

where $\mathbf{x}(t)$ are variables that appear differentiated in the model, $\mathbf{y}(t)$ are algebraic and $\mathbf{u}(t)$ are known input functions of time t . It is possible to transform system (3) to the following state space form, at least numerically:

$$\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \mathbf{x}_2 \\ \mathbf{y} \\ \mathbf{w} \end{bmatrix} = \mathbf{f}_1(\mathbf{x}_1, \mathbf{u}) \quad (4)$$

where \mathbf{x}_1 and \mathbf{x}_2 form vector \mathbf{x} such that the subset vector \mathbf{x}_1 is the *state vector* and contains the independent variables of \mathbf{x} . The new vector \mathbf{w} contains higher order derivatives of $\dot{\mathbf{x}}$ and of \mathbf{y} that appear

when differentiating equations of $\mathbf{f}(\cdot)$ and that are treated as algebraic variables. For the computation of $\mathbf{f}_1(\cdot)$, it might be necessary to solve linear and/or non-linear algebraic systems of equations. The equations to be differentiated can be determined with the algorithm of Pantelides [15]. The selection of the state variables \mathbf{x}_1 can be performed with the “dummy derivative method” of Mattsson and Söderlind [16]. Both algorithms are, for example, available in the Modelica simulation environment Dymola [5].

An *inverse* model of the DAE (3) is constructed by exchanging the meaning of variables: A subset of the *input* vector, \mathbf{u}_{inv} , with dimension n_{inv} , is treated no longer as known but as *unknown*, and n_{inv} previously unknown variables from the vectors \mathbf{x} and/ or \mathbf{y} are treated as *known* inputs. The result is still a DAE which can be handled with the same methods as any other DAE. Examples are given in the following sections. This technique of constructing non-linear inverse models has been first applied in [17][18]. An inverse model can only be used in a controller if the DAE of the inverse model has a *unique solution* and if it is *stable*. For linear systems the latter requirement means that the plant must be a minimum phase system. In section 6 it is discussed how to proceed if these requirements are not fulfilled.

Since the transformation from (3) to (4) might differentiate equations, the known inputs of the *inverse* model may be differentiated too, i.e., the derivatives of these inputs must exist and must be provided analytically up to a certain order. These derivatives can be provided if, e.g., the inputs are available as analytic functions that can be differentiated sufficiently often, or by a desired reference model that in combination with the inverse DAE results in a DAE that does not require derivatives of inputs. Often, the reference model is selected as a filter such that a combination of the filter states yields the needed derivatives. For linear systems, this approach is well known. Take for example the following linear system with one zero and two poles:

$$y = \frac{s+1}{(s-2) \cdot (s+3)} u \quad (5)$$

The inverse model is constructed as

$$u = \frac{(s-2) \cdot (s+3)}{(s+1)} \cdot \frac{1}{(Ts+1)} y \quad (6)$$

In order that the inverse model is causal (i.e., can be implemented as an algorithm), additional poles have to be added until the degree of the denominator is larger or, at least, as large as the degree of the numerator. For this reason, a filter $1/(Ts+1)$ has been connected in series, making the combined transfer

function causal. Another possibility is not to control y , but one of its derivatives instead:

$$\dot{y} = s \cdot y = \frac{s(s+1)}{(s-2) \cdot (s+3)} \cdot u$$

The transfer function is proper now and may be inverted:

$$u = \frac{(s-2) \cdot (s+3)}{s(s+1)} \dot{y}$$

Connecting this controller with the plant (5) in series results in integrator behavior. A simple feedback loop may be added to place the integrator pole at a desired location.

3 Constructing Inverse Models

With a Modelica simulation environment, such as Dymola, the practical derivation of inverse models is straightforward, even for complex systems:

1. Define the plant as Modelica model and include input and output signals of the plant over which the inversion shall take place.
2. If necessary, provide a reference model or input filter of appropriate relative degree. The relative degree may be known from physical knowledge of the plant dynamics, or can be automatically derived by Dymola as described below.
3. Connect the “u1” inputs of a “Modelica.Blocks.Math.TwoInputs” block to the plant outputs, the “u2” inputs of this block to the outputs of the reference model, and the input of this model to an input signal connector (Modelica.Blocks.Interfaces.RealInput) that defines the desired plant outputs.

A typical example is shown in Fig. 1. On the left side the plant model with one input and one output is

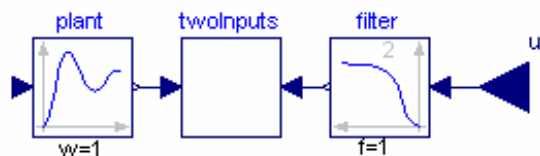


Fig. 1. Definition of inverse model with Modelica

present. On the right side, a filter is used as reference model. The output of the filter should be connected to the output of the plant. This is not directly possible, because signal connectors can only be connected according to block diagram semantic and in block diagrams it is not allowed to connect two output signals with each other. For this reason the “TwoInputs” block is used. It has two inputs $u1$ and $u2$ and

is described by the equation “ $u1 = u2$ ”. If the filter order is too low the DAE is not causal and Dymola prints an error message of the following form (Dymola version 5.3b and later):

Error: The model requires derivatives of some inputs as listed below:

Order of input	derivative
4	u1
2	u2
3	u3

Error: Failed to reduce the DAE index

In the second column the Modelica names of the input signals are listed that need to be differentiated according to the differentiation order of the first column. The numbers in the first column are therefore the minimum order of the corresponding filters.

If the inversion is to be based on a time derivative of the output, a sufficient number of integrators needs to be added, instead of increasing the filter order.

There is always a filter order / number of integrators for which the system will translate. The higher the filter order, the more problems will occur when applying it in a control system. In such cases, one might remove dynamic elements from the plant and try it again. One might even use a *stationary plant* model.

4 Example Controller Structures

In this section different controller structures will be discussed that follow the general approach outlined in section 2.

4.1 Inverse Model in Feedforward Path

Different variants of linear controllers with two structural degrees of freedom are known. The most general form for linear, single-input/single-output systems has been proposed and analyzed by Kreiselmeier [12]. According to the approach sketched in section 2, the generalization using nonlinear inverse models is shown in Fig. 2. This structure has been applied in [22] to the control of robots and has been successfully validated with hardware experiments. In flight control, the “model following approach”, see for example [2], is a special case of this structure whereby the reference model is known as the “command block” providing state references for the inverse model as well as the feedback controller.

In Fig. 2 the multi-input/multi-output plant has inputs u , measured signals y_m and outputs y_c that are primarily controlled. In many cases $y_c \in y_m$. For this controller structure the number of inputs must be

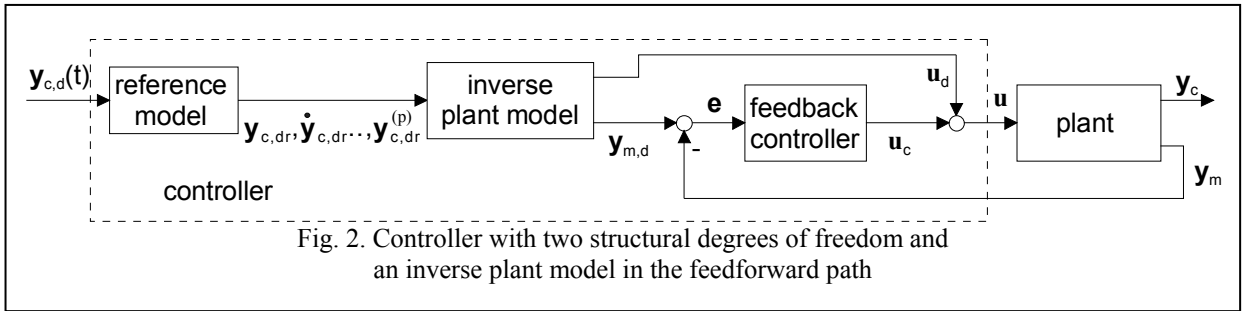


Fig. 2. Controller with two structural degrees of freedom and an inverse plant model in the feedforward path

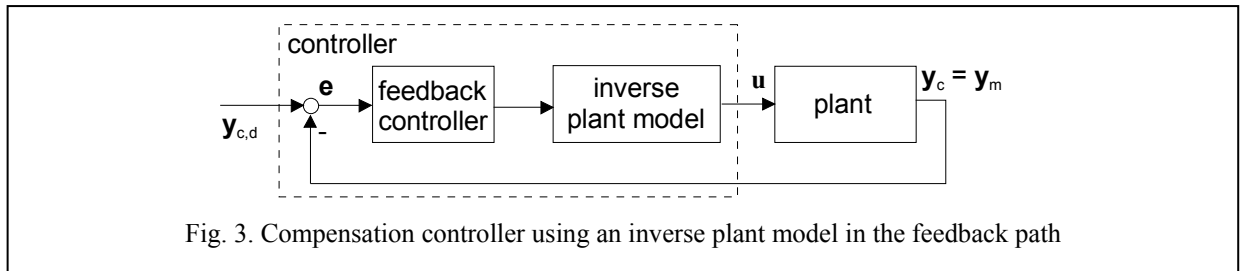


Fig. 3. Compensation controller using an inverse plant model in the feedback path

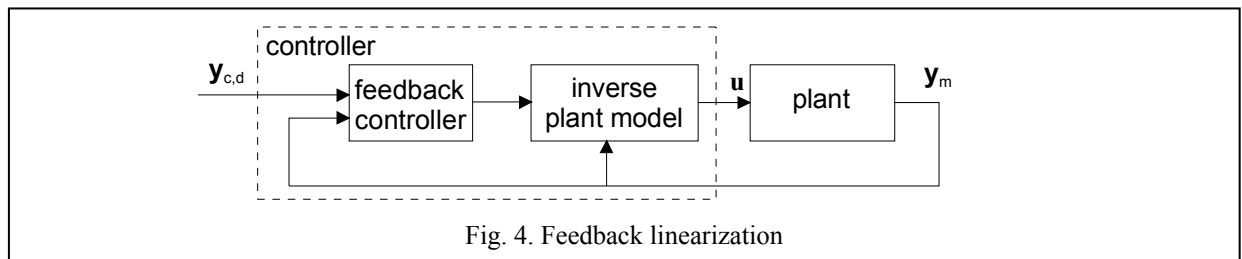


Fig. 4. Feedback linearization

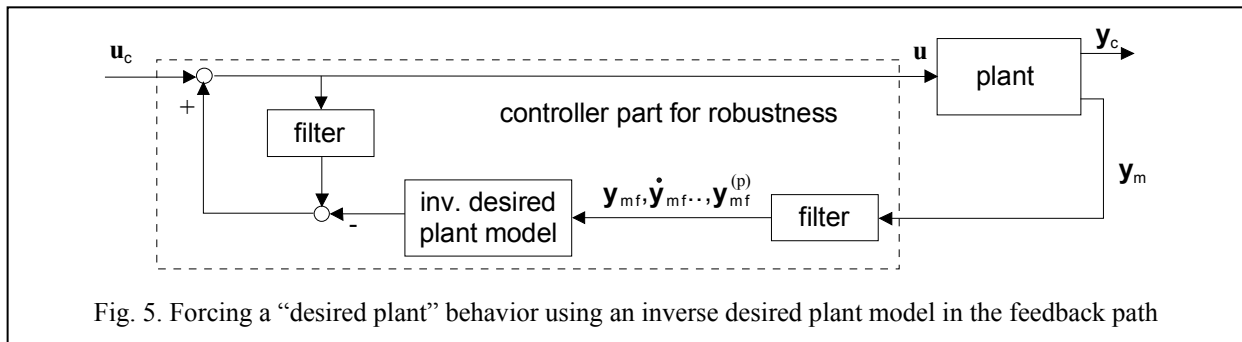


Fig. 5. Forcing a “desired plant” behavior using an inverse desired plant model in the feedback path

identical to the number of controlled variables: $\dim(\mathbf{y}_c) = \dim(\mathbf{u})$. In this case the inverse plant model with (known) inputs \mathbf{y}_c and unknown outputs \mathbf{u} is used in the feedforward path of the controller to compute the desired actuator inputs \mathbf{u}_d to the plant.

A “reference model” defines the desired dynamic behavior of the closed loop system. It is often most convenient to use a filter, since the filter is parameterized by just the cut-off frequency, once the filter order and the filter type is fixed, and because a filter provides the “optimal” reference model with transfer function “1” below the cut-off frequency. There are also other useful choices of the reference model, see for example [2] for in-flight simulation.

The outputs $\mathbf{y}_{c,dr}$ of the reference model are the inputs to the inverse plant model. By solving a DAE system (3) or the symbolically transformed system (4), the

inverse plant model computes the desired measurement signals $\mathbf{y}_{m,d}$ and the desired plant inputs \mathbf{u}_d . A feedback controller is used to stabilize the overall system and to improve robustness. This might be a simple PID like controller.

It can be shown that the feedback controller has no effect, as long as the plant and the inverse plant models are identical, the plant and the inverse plant models are stable and both start at the same initial conditions. In this case the “reference model” determines the input/output behavior, i.e., it is the transfer function of the closed loop system. If these assumptions are not fulfilled, a control error occurs and the controller has to stabilize the system and cope with the imprecise inverse plant model and its initial conditions.

The structure in Fig. 2 has several advantages:

- The two controller parts (inverse plant model with reference model and feedback controller) can be designed independently from each other.
- The controller structure can be applied to unstable plants provided the inverse model is stable, see section 6.1.
- Since the inverse plant model is in the feedforward path, the calculation of \mathbf{u}_d and of $\mathbf{y}_{m,d}$ might be performed *offline* if possible, so that hard real-time requirements for the solution of the inverse plant model are not present.

The disadvantage of this structure is that for some applications the feedback controller may still have to be scheduled as a function of the operating conditions.

Inverse model-based feedforward control will be demonstrated at hand of an example in section 5.

4.2 Compensation Controller

The disadvantage of the inverse feedforward controller can be avoided by moving the inverse model into the feedback path. This is shown in Fig 3. The feedback controller now only “sees” the combined inverse and plant model. The structure is a generalization of the *linear compensation controller* described in Föllinger [10], page 266. For linear plant models the “feedback controller”, see Fig. 3, must have a relative degree that is equal or larger than the relative degree of the plant, in order that the system is proper. For single-input/single-output systems, a useful “feedback controller” is

$$u_c = \frac{1}{r(s)-1} \cdot e \quad (7)$$

Under the assumption that the desired and the actual plant behavior is identical, the inverse and the actual plant model “cancel” each other and the transfer function from $y_{c,d}$ to y_c is identical to $1/r(s)$, i.e., $r(s)$ of the feedback controller defines the “desired” closed loop behavior. Note that it is assumed that y_c is measurable (in this case, $y_c = y_m$). Alternatively, the procedure as described in section 2 may be applied: integrators are added to the inverse model input before designing the feedback controller.

This structure has the disadvantage that it can be applied to stable plants only. Also the inverse plant model needs to be stable. For a linear plant model this is obvious, since otherwise an unstable pole/zero cancellation occurs, resulting in an internally unstable system. For multi-input/multi-output systems it is nearly always possible (also for unstable plants) to construct the inverse of a *stationary* desired plant model. Once the control error \mathbf{e} has reached a sta-

tionary value, the inverse plant model leads to a decoupled control loop. In other words, the different outputs might be controlled independently from each other by simple PID-like single-input/single-output controllers and the stationary inverse plant model is used to decouple the control loops from each other.

4.3 Feedback linearization

A complete theory to use nonlinear plant models as the controller kernel is “feedback linearization” (in aerospace applications also known as Nonlinear Dynamic Inversion, NDI), see for example Isidori [11] and Enns et. al. [7]. The basic structure is given in Fig 4. The principal difference compared with the compensation and feedforward controllers (Fig. 2,3) is that the states in the inverse model are obtained from the actual plant, via measurement and estimation. Contrary to the compensation controller, the methodology can also be applied to unstable plants.

When deriving feedback linearizing control laws manually, the outputs to be controlled are differentiated until an analytical relation with a control input is found [19]. To this end “Lie” algebra is used. The number of required differentiations is the so-called relative degree of the specific output. If the system model is available in Modelica, the derivation of the control laws can be automated using a similar procedure as described in section 2. However, instead of a filter of appropriate relative degree, a set of integrators is added (see section 2):

$$y_i = \frac{1}{s^{p_i}} v_i$$

where v_i is the i^{th} new model input, corresponding with the i^{th} output (with relative degree p_i). The desired dynamic behavior of the closed-loop system is then imposed by application of an additional feedback law, like for example:

$$v_i = k_{0,i}(y_{md,i} - y_{m,i}) - k_{1,i}(y_{m,i}^{(1)}) \dots - k_{(p_i-1),i}(y_{m,i}^{(p_i-1)}) \quad (8)$$

Note that this feedback law requires availability of the $(p_i-1)^{\text{th}}$ derivative of the controlled output. This derivative may be obtained from measurements or, less favorably, from the computed value in the inverse model. In aerospace applications first or no time derivatives are usually required, since relative degrees of controlled variables tend to be low (1 or 2). One reason for this is that control laws are designed in the form of multiple cascaded loops [7]. In case the inverted model exactly represents the true system, the closed loop system becomes:

$$y_{m,i} = \frac{k_{0,i}}{s^p + k_{(p-1),i}s^{p-1} + \dots + k_{1,i}s + k_{0,i}} y_{md,i}$$

Note that the coefficients may be selected to match the reference model in Fig. 2.

In case time derivatives of the desired output $y_{c,d,i}$ are available, the relative degree (i.e. phase lag of the response) of this linear closed loop system may be reduced, provided that these are not too fast as to require too large control inputs.

An important disadvantage of feedback linearization is that the state vector of the plant must be fully available from measurement and/or estimation.

Automatic generation of feedback linearization control laws in Modelica will be illustrated in section 5. This procedure has been applied for an automatic landing system, see [13], and manual control laws for a fighter aircraft, see [21]. The software code for the automatic landing system was automatically generated with Dymola and successfully flight tested on a small passenger jet [3], see the figure below that shows one of the automatic landing tests.



4.4 Robust Controllers

All previous controller structures require that the plant model used as inverse system in the controller match the real plant “sufficiently” accurate. The controller structure in Fig. 5 uses an inverse model to achieve a more robust design. It was developed for linear systems with the goal to enhance robustness against disturbances and model errors, see [14][23] [1]. This structure is called “disturbance observer” in the literature although the name is misleading since it is actually an additional structural degree of freedom for a controller. It can be designed independently from the main control loop. In Fig. 5 the generalization for nonlinear systems is shown: One important part is an inverse model of a desired plant behavior in the feedback path. Additionally, the same filter is present at two places. The standard disturbance observer uses a linear model for the inverse plant model. A nonlinear desired plant model provides more freedoms, since it might be impossible that a physical system can be forced to have the same

linear behavior in its whole operating range. Note, there is the requirement that the number of measurement signals is identical to the number of plant inputs: $\dim(\mathbf{y}_m) = \dim(\mathbf{u})$.

For a single-input/single-output system where all parts are linear, the transfer function from u_c to y_m is given by:

$$y_m = \frac{1}{\frac{1-F(s)}{P(s)} + \frac{F(s)}{P_{des}(s)}} \cdot u_c \quad (9)$$

where $F(s)$ is the filter, $P(s)$ is the plant and $P_{des}(s)$ is the desired plant transfer function in the feedback loop. For low frequencies, $F(s) \approx 1$ and therefore $y_m \approx P_{des}(s) \cdot u_c$. For high frequencies, $F(s) \approx 0$ and then $y_m \approx P(s) \cdot u_c$. The effect of the disturbance observer is therefore, that it enforces the desired plant behavior for low frequencies. In other words, if there are modeling errors or disturbances then the disturbance observer enforces a desired plant behavior below the cut-off frequency of the filter, i.e., the controller designed for the desired plant will usually work considerably better.

The disturbance controller is usually combined with other controller structures. For example, by combining it with the structure from section 4.1, a controller with 3 structural degrees of freedom is obtained:

- An inverse plant model from \mathbf{y}_c to \mathbf{u} in the feedforward path is used for *command following* and for providing the desired measurements $\mathbf{y}_{m,d}$.
- An inverse plant model from \mathbf{y}_m to \mathbf{u} in the feedback path is used to make the closed loop system *robust* against model errors and disturbances.
- The feedback controller in the feedback loop is used to *stabilize* the system.

5 Example application

In this section the feedforward and feedback linearization controller structures as discussed in the previous section will be illustrated on the following example (the plant description is from Föllinger [9], page 279):

A substance A is flowing continuously into a mixing reactor. Due to a catalyst, the substance reacts and splits into several base substances that are continuously removed. The reaction generates energy and therefore the reactor is cooled with a cooling medium. The cooling temperature $T_c(t)$ in [K] is the primary actuation signal. Substance A is described

by its concentration $c(t)$ in [mol/l] and its temperature $T(t)$ in [K] according to the following DAE:

$$\begin{aligned}\gamma &= c \cdot k_0 \cdot e^{-\varepsilon/T} \\ \dot{c} &= -a_{11} \cdot c - a_{12} \cdot \gamma + a_{13} \\ \dot{T} &= -a_{21} \cdot T + a_{22} \cdot \gamma + a_{23} + b \cdot T_c\end{aligned}\quad (10)$$

with

$$\begin{aligned}k_0 &= 1.24 \cdot 10^{14} & a_{11} &= 0.00446 & a_{21} &= 0.0303 \\ \varepsilon &= 10578 & a_{12} &= 0.0141 & a_{22} &= 2.41 \\ b &= 0.0258 & a_{13} &= 0.00378 & a_{23} &= 1.37\end{aligned}$$

For the given input $T_c(t)$ these are 1 algebraic equation for the reaction speed $\gamma(t)$ and two differential equations for $c(t)$ and $T(t)$. The concentration $c(t)$ is the signal to be primarily *controlled* ($= y_c$) and the temperature $T(t)$ is the signal that is measured ($= y_m$).

5.1 Inverse Model in Feedforward Path

The *inverse* plant model is constructed from (10) by assuming that the variable to be controlled, i.e., the concentration $c(t)$, is a known time function. By inspection or by using the Pantelides algorithm [15] it turns out that the first two equations of (10) have to be differentiated:

$$\begin{aligned}\dot{\gamma} &= \left(\dot{c} + \frac{\varepsilon \cdot c}{T^2} \dot{T} \right) \cdot k_0 \cdot e^{-\varepsilon/T} \\ \ddot{c} &= -a_{11} \cdot \dot{c} - a_{12} \cdot \dot{\gamma}\end{aligned}\quad (11)$$

(10) and (11) are the *inverse model* of (10). A filter with an n^{th} order pole on the negative real axis is used as “reference model”. Since the second derivative of the input appears ($= \ddot{c}$), at least a filter of order 2 is needed, such as:

$$c = \frac{1}{(s/\omega + 1)^2} \cdot c_{des}\quad (12)$$

with c_{des} the desired concentration, $\omega = 2\pi f$ and f the cut-off frequency of the filter. A state space description of the filter is given by:

$$\begin{aligned}\dot{x} &= (c_{des} - x) \cdot \omega \\ \dot{c} &= (x - c) \cdot \omega\end{aligned}\quad (13)$$

The needed second derivative of c is obtained by differentiating the second equation of (13):

$$\ddot{c} = (\dot{x} - \dot{c}) \cdot \omega\quad (14)$$

Equations (10), (11), (13), (14) are the DAE of the *inverse model* of (10) with a *prefilter* of order 2, i.e., these are the connected blocks labeled as “inverse plant model” and as “reference model” in Fig. 2. It

turns out that this DAE has two states. One possibility is to use the filter states $\{x, c\}$ as state vector \mathbf{x}_1 of the overall system. Here, the original plant states $\{c, T\}$ are used as state vector \mathbf{x}_1 . Transforming the equations to the state space form (4) results in the following *sequence of assignment* statements to compute the derivative $\{\dot{c}, \dot{T}\}$ of the state vector and of the output T_c as function of $\{c, T\}$

$$\begin{aligned}\gamma &:= c \cdot k_0 \cdot e^{-\varepsilon/T} \\ \dot{c} &:= -a_{11} \cdot c - a_{12} \cdot \gamma + a_{13} \\ x &:= c + \dot{c} / \omega \\ x' &:= (c_{des} - x) \cdot \omega \\ c'' &:= (x' - \dot{c}) \cdot \omega \\ \gamma' &:= (c'' + a_{11} \cdot \dot{c}) / a_{12} \\ \dot{T} &:= \frac{T^2}{\varepsilon \cdot c} \cdot \left(\frac{\gamma'}{k_0 \cdot e^{-\varepsilon/T}} - \dot{c} \right) \\ T_c &:= (\dot{T} + a_{21} \cdot T - a_{22} \cdot \gamma - a_{23}) / b\end{aligned}\quad (15)$$

For notational clarity, the time derivatives of variables that are treated as purely *algebraic* variables ($=$ “dummy derivative method”) are denoted with an apostrophe, such as γ' . Equations (15) are a set of differential equations in state space form: Given the desired concentrations c_{des} , it is possible by numerical integration to compute the desired cooling temperature T_c ($= \mathbf{u}_d$ in Fig. 2) and the desired substance temperature T ($= \mathbf{y}_{m,d}$ in Fig. 2). The latter is compared with the measured substance temperature forming the control error \mathbf{e} as input to the feedback controller.

Even for this rather simple system, the derivation of the nonlinear feedforward controller is not so easy. Such a manual derivation becomes impractical if the plant model consists of hundreds or of thousands of equations as it is usual in complex Modelica models. It is now demonstrated how to derive this nonlinear feedforward controller in an *automatic* way:

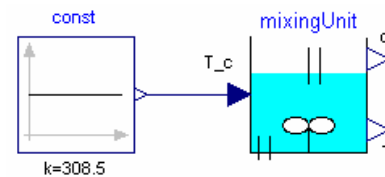


Fig 6. Modelica model of mixing unit with constant cooling temperature T_c

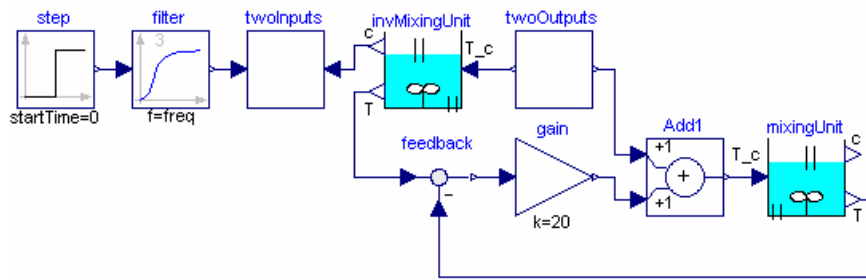


Fig 9. Control system with nonlinear feedforward path for mixing unit

In Fig. 6 a Modelica model of the mixing unit is shown. The constant input is the cooling temperature; the outputs are the concentration c and the temperature T of the substance. This model contains just the equations (10). Simulation results of this model are shown in Fig. 7. As can be seen, the system is unstable at this operating point

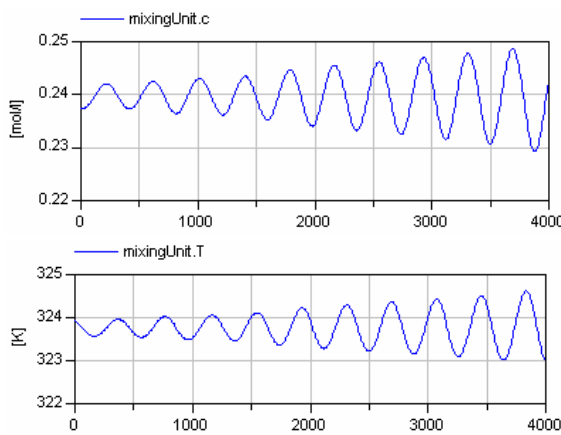


Fig. 7. Simulation results of mixing unit for $c(t_0) = 0.237$ mol/l, $T(t_0) = 323.9$ K, $T_c(t) = 308.5$ K

In Fig. 8 the inverse model of the mixing unit is constructed by connecting the input “ c_{des} ” via a filter to the “ c ” output of the mixing unit, i.e., the concentration c is treated as known input signal.

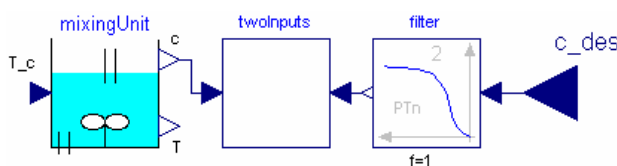


Fig 8. Inverse model of mixing unit

When this system is translated without the filter, Dymola reports that the second derivative of c_{des} is needed. In a second step, the filter is included with order = 2 and Dymola translates without an error. Afterwards, the inverse model is connected with the plant model according to Fig. 2. The result is shown in Fig. 9. In order to not have a jump in the cooling temperature, a filter order of 3 instead of 2 is actually used. The cut-off frequency of the filter is set to 1/300 Hz. It turns out that a simple P controller is

sufficient to stabilize the system. A controller gain of 20 is selected.

Simulation results are shown in Fig. 10 for a jump of $c_{des} = 0.492$ to 0.237 . The straight lines correspond to the nominal case, where the plant and the inverse plant model have the same parameters. The result is a good control behavior. The dashed lines correspond to the case where the parameters of the plant are 50 % higher as the parameters of the inverse plant model to check the robustness of the design (only parameter ε was not changed because the result is very sensitive to it). As can be seen, the result is still satisfactory. For an actual design, it is useful to perform a Monte Carlo simulation by varying all model parameters and initial conditions of the plant systematically in order to determine how robust the control system is

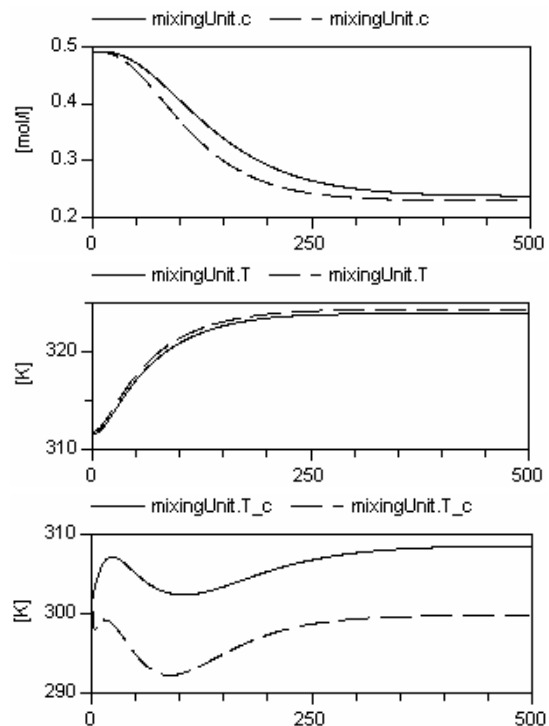


Fig. 10. Simulation results of mixing unit of Fig. 8 Straight line: same model parameters for plant and inverse plant model. Dashed line: model parameters of plant are enlarged by 50 %

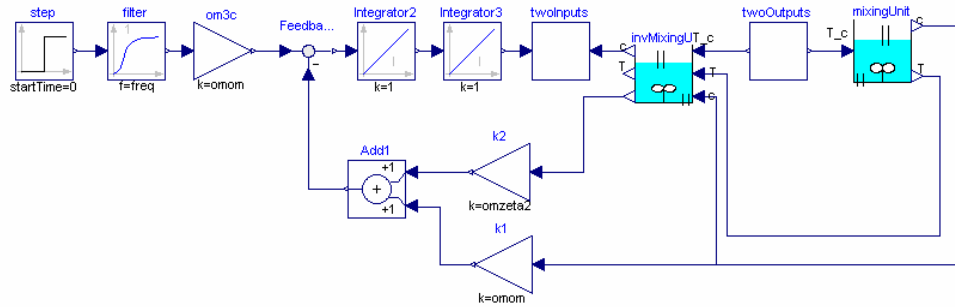


Fig 12. Closed loop system of mixing reactor and feedback linearization controller

5.2 Feedback linearization

The compensation control scheme and the feedback linearization cannot be applied directly to the example plant, since the concentration c is not measurable. One possibility is to use model knowledge in combination with estimation (e.g. a Kalman filter), but this is beyond the scope of this example. For this reason it is assumed that the concentration is measurable. In Modelica, design of feedback linearization and the compensation controllers start in the same way. This time two integrators are added, instead of an input filter, see Fig. 11.

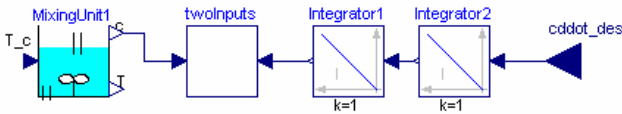


Fig 11. Inverse model of mixing unit for feedback linearization (compare with Fig. 8)

In the case of the compensation controller, the inversion work is done. For feedback linearization, the states in the inverse plant model must be replaced with measured ones. This can be performed by setting the flag

`Advanced.TurnStatesIntoInputs = true`

before translation to transform all states into inputs in the generated code. This code can be incorporated with the export feature of Dymola in another environment, such as Simulink from Mathworks. Currently, it is not possible to import this transformed system in Modelica again. Dynasim plans to support this in the future. For the example, the differentiated equations (11) are added manually to the model, and \ddot{c} and the plant states (c, T) are selected as input variables (in case of complex models this manual derivation is not practical). The design is finished by adding the feedback controller. In case of feedback linearization, a usual choice is:

$$c'' = k_1(c_{des} - c) - k_2\dot{c}$$

whereby c is available from measurement and \dot{c} is computed or obtained from differentiation. By choosing

$$k_1 = (2\pi f)^2, \quad k_2 = 2(2\pi f),$$

with $f = 1/300$ Hz, exactly the same closed loop dynamics is obtained as with the input filter of second order in section 5.1. Starting from the ideal response

$$\frac{1}{r(s)} = \frac{k_1}{s^2 + k_2s + k_1}$$

the feedback controller may also be shaped as:

$$T_c = \frac{1}{r(s) - 1} s^2 = \frac{k_1 s^2}{s^2 + k_2 s}$$

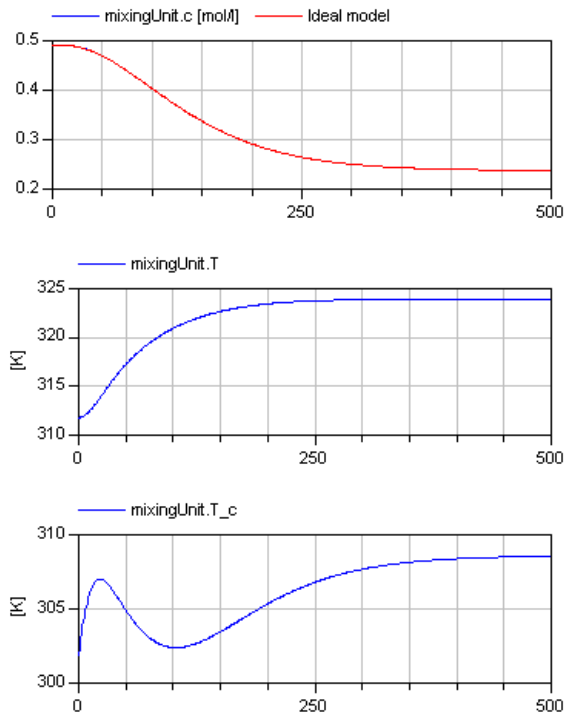


Fig 13. Closed-loop and ideal step response of the mixing reactor

whereby in the numerator s^2 has been added, since the input of the inverse model is effectively the second time derivative of c_{des} . Fig. 13 shows the response of the closed loop system to the same command as in Fig. 10. The command input has been smoothed with a first order filter (as in Fig. 8). The over-all closed-loop system is depicted in Fig. 12.

6 Difficulties with Inverse Models

When constructing inverse models for industrial systems, it is often the case that the generated inverse models do not work as expected. In this section, the major reasons are discussed and it is explained how to circumvent such problems.

6.1 Unstable inverse models

Usually, it is required that the inverse model is a *stable system*. For example, in the structure of Fig. 2, the inverse model is in the feedforward path and if it would not be stable, the overall system would be unstable as well. For linear single-input/single-output systems this situation is well known and can be easily analyzed. For example, take the following linear plant model:

$$y = \frac{s-1}{(s-2) \cdot (s+3)} u \quad (16)$$

The inverse model together with a reference model is

$$u = \frac{(s-2) \cdot (s+3)}{(s-1) \cdot (Ts+1)} y \quad (17)$$

As can be seen, the inverse model is unstable, because the plant has an unstable zero. In other words, for linear systems the plant must be a *minimum phase system* in order that the inverse model is stable. For a general DAE no stability proof exists. Therefore many simulations have to be performed with the inverse DAE to check whether it is stable in the desired operation region. For certain classes of DAEs, it might be possible to prove that the inverse model is stable. An alternative is to linearize the plant model around several stationary operating points and check whether the transmission zeros are stable. Of course, none of these checks can guarantee that the inverse DAE is stable for simulations or stationary points that have not been analyzed.

If the inverse plant is unstable, only *approximate* inverse plant models can be used for the design. For linear single-input/single-output systems this can be achieved by removing unstable zeros before invert-

ing the plant. E.g., in the example above, the approximate inverse plant model would be:

$$u = \frac{(s-2) \cdot (s+3)}{(Ts+1)^2} y \quad (18)$$

For a non-linear plant, one might choose other outputs of the plant as inputs to the inverse model, since this might change the stability behavior of the inverse plant, see for example [20]. Alternatively, the plant might be modified before inversion. These advices are demonstrated by the crane example in Fig. 14.

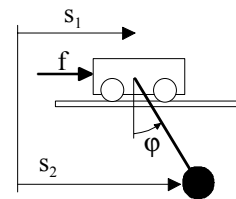


Fig 14. Crane consisting of horizontal moving crab and a load on a rope

The crane consists of a horizontally moving crab and a rope on which the load is attached. For simplicity, the load is modeled as a mass point. The crab is driven by the external force “f”. The horizontal position of the crab “s1” and its derivative “v1” are measured. The goal is to move the load to a specified horizontal position “s2”.

For a non-linear disturbance observer, the inverse model from s1 to f is needed, since s1 is measured. The system is first linearized around the stationary position where the rope hangs vertically down ($\varphi = 0$). The transfer function from f to s1 has 2 conjugate complex zeros on the imaginary axis, signaling an undamped oscillation of the inverse model. This can be improved by including linear damping ($= d \cdot \dot{\varphi}$) in the revolute joint for the inverse plant used in the controller. If the damping constant d is large enough, the two zeros on the imaginary axis are moved to the negative real axis. The disturbance observer is able to force the plant (that does have low damping) moving in such a way as if there would be high damping.

The major goal is to position the load, i.e., to control the horizontal position “s2” of the load. Therefore, the feedforward control should use the inverse model from s2 to f. The transfer function from f to s2 of the linearized model has no zeros and a relative degree of 4. Constructing the inverse model from the non-linear plant model requires, however, a filter of order 2 instead of 4 as suggested by the linearized model. Simulating the inverse model results in a division by zero if $\varphi = 0^\circ$ or $\varphi = 180^\circ$. To summarize, the structure of the inverse model equations is different

at these two points and at $\varphi \neq 0^\circ$ and $\varphi \neq 180^\circ$ (the DAE index is 5 for $\varphi = 0^\circ$ and $\varphi = 180^\circ$ and the DAE index is 3 otherwise). Since the division by zero occurs when computing $\ddot{\varphi}$, the plant model should be changed to compute $\ddot{\varphi}$ in a different way. This can be accomplished by taking the inertia of the load into consideration (previously it was neglected). With a non-zero inertia, the transfer function from f to s_2 of the linearized plant has 2 conjugate complex zeros on the imaginary axis and a relative degree of 2. Again, by introducing damping in the revolute joint, these two zeros are moved to the negative real axis. Note, that the inverse model is very insensitive with respect to the newly introduced load inertia.

To summarize, for the crane example the inverse plant models from s_1 to f and from s_2 to f can be constructed by inverting a modified plant that has a load inertia and additionally damping in the revolute joint. A simpler alternative is also available: Before inversion, the angle φ is fixed to 0° and therefore $s_1 = s_2$, and the plant to be inverted is described by the equations (m_{crab} is the mass of the crab and m_{load} is the mass of the load):

$$(m_{\text{crab}} + m_{\text{load}}) \cdot \ddot{s}_1 = f \quad (19)$$

which can be easily inverted. This example demonstrates that it might be necessary to slightly modify the original plant model in order that the inverse model of the plant can be used in a controller.

6.2 Equations that cannot be inverted

A plant may have equations that cannot be inverted. Examples are time delays, backlash, friction, hysteresis. This can be fixed by *approximating* the problematic elements in such a way that the resulting equation leads to a unique inverse.

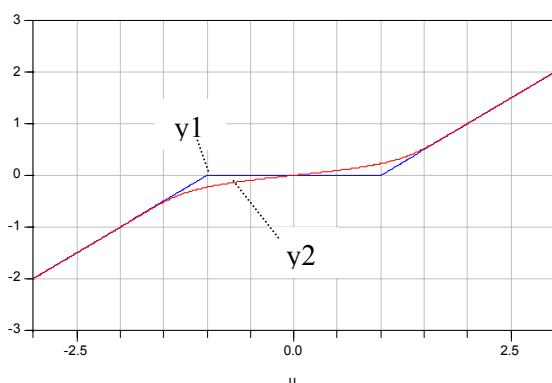


Fig. 15. Backlash (y_1) and approximate backlash (y_2)

A typical example is shown in Fig. 15. The original backlash characteristic $y_1 = f_1(u)$ is not invertible

because for $y_1 = 0$, there are an infinite number of solutions ($u = -1 \dots +1$). In Fig. 15 an approximation $y_2 = f_2(u)$ is shown that is strict monotonic and therefore the inverse function has a unique solution.

It might also occur that tables have to be inverted. Formally, a table in one dimension is defined as a function $y = f(u)$. Inversion of this function means to solve a non-linear equation. This can be often quite easily avoided by providing already the inverse tabulated values $u = g(y)$ in the plant before inverting the plant model. The advantage is that the solution is faster and more robust. This problem was, e.g., encountered in [21], where control surface effectiveness of a military jet tended to have a local maximum as a function of the deflection. This was solved by adapting tables and internal limitations of control commands.

The inverse plant model may have also other singularities at particular operating points or regions that prevent an inversion, e.g., due to divisions by zero, singular linear or singular non-linear systems. The reason is that the corresponding inverse model has no or infinitely many solutions in particular points or in particular regions of the state space. Again, one remedy is to change the plant model before the inversion, e.g., by neglecting dynamic elements or by approximating components with functions that are less problematic to invert.

6.3 Actuator limits

Every control system is inherently limited by constraints in the actuator or other parts of the plant and therefore the question arises how to cope with these restrictions. When inverting a plant model, such constraints have to be removed before the inversion. Otherwise no unique solution of the inverse exists anymore, because there are infinitely many solutions when an actuator is in one of its limits. As a result, usually only the trivial action is possible to add appropriate limiters to the outputs of inverse models. This will only help for short-time violations of the constraints because the control system is effectively switched off when the actuators are in their limits.

The most effective way to cope with actuator constraints in any control system is to adapt the desired control signals, such as $y_{c,d}(t)$ in Fig. 2. In the most general case this means to solve a trajectory optimization problem, i.e., to determine actuator signals $u(t)$ such that the plant outputs $y_c(t)$ have a desired behavior, e.g., reach the desired position in minimum time or with minimum energy, without violating the plant constraints. The result is used as $y_{c,d}(t)$. A typical example can be found in [8]. Note, if the plant is

unstable and the inverse plant model is stable, it might be considerably simpler to solve the trajectory optimization problem with the inverse plant instead with the plant model. Usually, trajectory optimization problems are difficult to solve and therefore highly simplified plant models are used.

Take for example the crane model from section 6.1. The basic requirement is to move the crab from position $s_1=a$ to $s_1=b$ in a short time. The plant model is simplified by fixing the angle to $\varphi = 0^\circ$ resulting in equation (19). Based on (19), the actuator limit $|f| \leq f_{\max}$ can be directly transformed into a limit of the acceleration:

$$|\ddot{s}_1| \leq f_{\max} / (m_{\text{crab}} + m_{\text{load,max}}) \quad (20)$$

Together with limits on the maximum speed, due to the maximum speed of the motor, $|\dot{s}_1| \leq \dot{s}_{1,\max}$, and the requirement to move in minimum time from a to b it is straightforward to construct the analytic solution of the desired movement $s_{1,d}(t)$. This solution is, e.g., available via the block `Modelica.Blocks.Sources.KinematicPTP`. Note, the plant model used for the trajectory optimization problem and for the inverse plant model in the feedforward path according to Fig. 2 are identical here. In such a case, the feedforward controller can be removed and can be replaced by the result of the trajectory optimization: $s_{1,d}$ and $f_{1,d} = (m_{\text{crab}} + m_{\text{load,max}}) \cdot \ddot{s}_{1,d}$. For the trajectory optimization problem an f_{\max} should be used that is, say, 10 % - 20 % smaller as the actual limit in order to provide some margin for the feedback controller.

If the desired control variables $y_{c,d}(t)$ are not known in advance but generated online, e.g., by an operator, online optimization techniques have to be used: The operator request is reduced such that the plant constraints are fulfilled in the next sample time instant. A well known measure in flight control is the so-called daisy-chain. In case a control input saturates, a secondary, redundant control input is brought in that provides the remaining required control power. In [21] for example, lateral deflection of the thrust vector is used to yaw the aircraft in case the rudder saturates.

6.4 Real-time implementation

If inverse plant models are part of the controller, linear and non-linear systems of equations as well as non-linear differential equations might have to be solved in every sampling interval of the controller. The techniques developed for hardware-in-the-loop

simulations can be also applied for such an application. The methods described in [6] are available in Dymola [5] with the Dymola real-time option and can be applied by selecting the appropriate options when translating the inverse model (Simulation / Setup / Realtime / Inline integration method). Only fixed step integrators can be used for a real-time application. Via simulations, the appropriate step size of the integrator has to be determined.

6.5 Robustness

As already mentioned in section 5, the use of inverse model equations gives rise to robustness issues, since any mismatch between the inverted model equations and the actual plant will leave part of the nonlinearities and couplings uncompensated. The usual approach is to provide robustness to model uncertainty via the (linear) feedback controller (Fig. 2,3,4) or the filter (Fig. 5). This can be done by application of a robust control synthesis technique [2], or by robust parameter tuning in a classical structure, e.g. using multi-model techniques and enforcing sufficient stability margins [13].

Tolerances on parameters in the model also appear in the inverse model equations. In [13] it has been shown that these parameters may be very effectively used as additional tuning parameters in multi-objective optimization. The result is a model that is basically inverted at a location in the parameter space that provides the highest level of robustness.

7 Summary

Several control structures have been discussed that are based on non-linear inverse plant models. These structures are attractive since it is possible to cope directly with operating point dependencies. The difficult part to construct an inverse model can be performed automatically even for complex systems: The plant is modeled with Modelica, inputs and outputs are exchanged and a Modelica simulation environment, such as Dymola, generates automatically the appropriate C code for the inverse plant model, including real-time integration algorithms. The generated code can be easily embedded into Simulink from Mathworks using the corresponding Dymola export option. Via Mathworks Realtime-Workshop, the code can be finally downloaded to different target processors.

The presented controller structures can be used in all types of areas such as control of robots, vehicles, aircrafts, satellites, ships, motors, air conditioning

systems. The most important requirement is that an appropriate plant model is available. Then, the inverse modeling approach is in principle fully automatic, although the practical application is usually more difficult. The essential issues have been discussed in section 6 and also possible remedies

References

- [1] Ackermann J., Blue P., Bünte T., Güvenc L., Kaesbauer D., Kordt M., Muhler M., and Odenthal D. (2002): **Robust Control: The Parameter Space Approach**. Springer-Verlag.
- [2] Adams, R.J., Banda, S. **Robust Flight Control Design Using Dynamic Inversion and Structured Singular Value Synthesis**. IEEE Transactions on Control Systems Technology, 1(2):80-92, June 1993.
- [3] Bauschat, M., Mönnich, W., Willemsen, D., and Looye, G. **Flight testing Robust Autoland Control Laws**. In Proceedings of the AIAA Guidance, Navigation and Control Conference, Montreal CA, 2001.
- [4] Duda H., Bouwer G., Bauschat J.M., Hahn K.-U. (1997): **A Model Following Control Approach**. In "Robust Flight Control: A Design Challenge" by J.-F. Magni, S. Bennani and J. Terlouw (editors), Springer Verlag, pp. 116 – 124.
- [5] Dynasim (1994): **Dymola – Users Manual** (<http://www.dynasim.com>)
- [6] Elmqvist H., Mattsson S.E., Olsson H. (2002): **New Methods for Hardware-in-the-Loop Simulation of Stiff Models**. 2nd International Modelica Conference, March 18-19, DLR Oberpfaffenhofen, Germany, pp. 59-64. Download: <http://www.Modelica.org/-Conference2002/papers.shtml>.
- [7] Enns, D., Bugajski, D., Hendrick, R., and Stein, G.. **Dynamic Inversion: An Evolving Methodology for Flight Control Design**. In AGARD Conference Proceedings 560: Active Control Technology: Applications and Lessons Learned, pages 7-1 – 7-12, Turin, Italy, May 1994.
- [8] Franke R., Rode M., and Krüger K. (2003): **On-line Optimization of Drum Boiler Startup**. 3rd Int. Modelica Conference, Linköping, Nov. 3-4, pp. 287 – 296. Download: <http://www.Modelica.org/-Conference2003/papers.shtml>.
- [9] Föllinger O. (1998): **Nichtlineare Regelungen I**, Oldenbourg Verlag, 8. Auflage.
- [10] Föllinger O. (1994): **Regelungstechnik**. Hüthig Verlag, 8. Auflage.
- [11] Isidori A. (1995): **Nonlinear Control Systems**. 3rd Edition, Springer Verlag.
- [12] Kreisselmeier G. (1999): **Struktur mit zwei Freiheitsgraden**. Automatisierungstechnik at 6, pages 266-269.
- [13] Looye G. (2001): **Design of Robust Autopilot Control Laws with Nonlinear Dynamic Inversion**. Automatisierungstechnik at 49-12, p. 523-531.
- [14] Ohnishi K. (1987): **A new servo method in mechatronics**. Trans. Japanese Society of Electrical Engineering, vol 107-D, pp. 83-86.
- [15] Pantelides C.C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, pp. 213-231.
- [16] Mattsson S.E., Söderlind G. (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, pp. 677-692.
- [17] Mugica F., Cellier F.E. (1994): **Automated synthesis of a fuzzy controller for cargo ship steering by means of qualitative simulation**. Proceedings of the European Simulation MultiConference (ESM'94), Barcelona, Spain, pp. 523-528,
- [18] Otter M., Cellier F.E. (1996): **Software for Modeling and Simulating Control Systems**. The Control Handbook, by W.S. Levine (editor), CRC Press, pp. 415 – 428.
- [19] Slotine, J.E, Li, W. **Applied Nonlinear Control**. Prentice Hall, Englewood Cliffs, N.J., 1991.
- [20] Snell, A. **Decoupling of Nonminimum Phase Plants and Application to Flight Control**, AIAA-2002-4760 AIAA Guidance, Navigation, and Control Conference and Exhibit, Monterey, California, 2002.
- [21] Steinhäuser R., Looye G., Brieger O. (2004): **Design and Evaluation of a Dynamic Inversion Control Law for X-31A**. Proc. 6th ONERA-DLR Aerospace Symposium, Berlin, June 22-23, pp. 25-33.
- [22] Thümmel M., Otter M., Bals J. (2001): **Control of Robots with Elastic Joints based on Automatic Generation of Inverse Dynamics Models**. IEEE/RSJ Conference on Intelligent Robots and Systems, Oct. 29- Nov. 3rd, Hawaii, U.S.A.
- [23] Umeno T., Hori Y. (1991): **Robust speed control of dc servomotors using modern two degrees-of-freedom controller design**. IEEE Trans. Ind. Electron, 38-5, pp. 363-368.

An Empirical Study on Debugging Equation-Based Simulation Models

Peter Bunus

Department of Computer and Information Science

Linköping University, Sweden

petbu@ida.liu.se

Abstract

A typical problem which often appears in Modelica models is when too many/few equations are specified. This leads to a situation where the simulation model is inconsistent and therefore cannot be compiled and executed. We propose a methodology for detecting and repairing over- and under-constrained situations based on graph theoretical methods. Components and equations that cause the irregularities are automatically isolated, and meaningful error messages for the user are presented. The potentially large number of error fixing alternatives is reduced by applying filtering rules extracted from the modeling language semantics.

The paper illustrates that it is possible to localize and repair a significant number of errors during static analysis of a Modelica model without having to execute the simulation model. In this way certain numerical failures can be avoided later during the execution process. The paper proves that the result of structural static analysis performed on the underlying system of equations can effectively be used to statically debug real models.

Keywords: Modelica, debugging, structural and static analysis, mathematical modeling, structural validation.

1 Introduction

Mathematical modeling and simulation of complex physical systems is emerging as a key technology in engineering. Modern approaches to physical system simulation allow users to specify simulation models with the help of equation-based languages. Such languages have been designed to allow automatic generation of efficient simulation code from declarative specifications. Complex simulation models are created by combining available model components from user-defined libraries. The resulted models are compiled in a simulation environment for efficient execution.

Unfortunately, errors are made and inconsistencies are easily introduced in the simulation models. A significant part of the model development effort is spent on detecting deviations from specifications and subsequently localizing the sources of such errors. A typical problem which often appears in physical system modeling and simulation is

when too many/few equations are specified in a system. This leads to a situation where the simulation model is inconsistent and therefore cannot be compiled and executed. The user should deal with over- and under-constrained situation by identifying the minimal set of equations or variables that should be removed from the system in order to make the remaining set of equations solvable. For example, if there are too many equations in a system of 100 000 equations, which equations should be removed? Currently the only systematic technique is to remove equations one by one until the equation that caused the inconsistency is identified and finally removed from the system. It can easily be imagined that, if a static debugger presents a small subset of over-constraining equations, from which the user can select the equation that needs to be eliminated from the overall model can greatly reduce the amount of time required to get the simulation working.

Currently there are essentially no advanced tools that can handle the debugging of equation-based languages at the source code level and provide useful error fixing solutions. The aim of the research presented in this paper is to considerably improve the situation, especially with respect to debugging the Modelica language. However, powerful graph-theoretic methods can help to pinpoint possible candidates for erroneous equations. A dramatic reduction in the number of erroneous equation candidates can be achieved by applying new methods such as semantic filtering.

In this paper we describe an empirical evaluation of debugging of automated debugging techniques for detecting and repair structural inconsistencies in equation-based simulation models. We focus on performance of debugging tools that use static analysis tools integrated into a Modelica compiler where the main purpose was to reduce the number of debugging alternatives when structural inconsistencies were present in the model. Static analysis techniques only involve statically available information, such as which variables are present in which equations in and equation-based model. No assumptions regarding the inputs and outputs of the simulation models are made. The development of static and dynamic techniques for equation-based languages have been addressed by our previous research (Bunus 2004 [1], Bunus and Fritzson 2003 [2], Bunus and Fritzson 2004 [3]).

The remainder of the paper is organized as follows: Section 2 presents the problem formulation and a motiva-

tional example. Section 3 gives a brief description of the algorithms for detecting and debugging over-constrained situations that arise during the modeling phase with equation-based languages. Section 4 presents an evaluation of our debugging framework based on several benchmarks. Section 5 presents the overall architecture of a prototype debugger developed in the context of a Modelica compiler. Finally Section 6 presents our conclusions and future work.

2 Problem Formulation and Motivational Example

Mathematical modeling proceeds by specifying a set of mathematical equations or functional relations denoted $E = \{e_1, \dots, e_n\}$ involving a set of variables denoted $V = \{v_1, \dots, v_m\}$. In the general case a system of n equation with m variables or unknowns can be described by the following equality:

$$e_i(v_1, \dots, v_m) = c_i \quad (2.1)$$

where c_i are constants and $i = 1 \dots n$. Solving the system of equations E is the problem of finding the set of solutions $S = \{(s_1, \dots, s_m) \in T^m \mid e(s_1, \dots, s_m)\}$ where T is the domain of equations, which fulfill the equality (2.1). The relation (2.1) can be expanded into:

$$\begin{aligned} a_{11}v_1 + \dots + a_{1m}v_m &= c_1 \\ &\vdots \end{aligned} \quad (2.2)$$

$$a_{n1}v_1 + \dots + a_{nm}v_m = c_n$$

where a_{ij} , $i = 1 \dots n$, $j = 1 \dots m$ are real coefficients. In a matrix-vector notation, (2.2) has the form: $\mathbf{A}\mathbf{v} = \mathbf{c}$

$$\text{where } \mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} \text{ and } \mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \quad (2.3)$$

A necessary condition for the existence and uniqueness of a solution S is that matrix \mathbf{A} is a square matrix (the number of equations is equal to the number of variables) and there exists permutations $\mathbf{P}_1\mathbf{P}_2$ such that $\mathbf{P}_1\mathbf{A}\mathbf{P}_2$ has a non-zero diagonal. This condition guarantees the structural singularity of the system of equations. The structural singularity checks whether the system of equations is well-posed or not. It is only a necessary but not sufficient condition for the existence and uniqueness of a solution. The more powerful notion of numerical singularity will guarantee the existence and uniqueness of a solution. However the checking the numerical singularity is as expensive as solving the system of equations. Therefore when analyzing the system of equations in this stage we assume that the structural non-singularity is a sufficient abstraction for implying that the equation system has a unique solution. Further analysis based on numerical values and numerical singularities is delayed until the dynamic analysis stage.

If the system of equations is structurally singular we switch from the problem of finding the set of solutions S to the problem of finding the maximal subset of equations

$E_S = \{e_1, \dots, e_t\}$ where $t < n$ and $E_S \subset E$ if $n > m$ (we have more equations than variables) or to the to the problem of finding the maximal subset of variables $V_S = \{v_1, \dots, v_k\}$ where $k < m$ and $V_S \subset V$ if $n < m$ (we have more variables than equations).

As an example let us consider a Modelica model consisting of a sinusoidal voltage source and a resistor connected together. This model is trivial, but it serves as a straightforward vehicle for introducing several fundamental debugging concepts.

```
connector Pin
  Voltage v;
  Flow Current i;
end Pin;

model TwoPin
  Pin p, n;
  Voltage v;
  Current i;
equation
  v = p.v - n.v; 0 = p.i + n.i; i = p.i
end TwoPin;

model Resistor
  extends TwoPin;
  parameter Real R;
equation
  R*i = v;
end Resistor;

model VsourceAC
  extends TwoPin;
  parameter Real VA=220; parameter Real f=50;
  protected constant Real PI=3.141592;
equation
  v=VA*(sin(2*PI*f*time));
end VsourceAC;

model Ground
  Pin p;
equation
  p.v = 0
end Ground;

model Circuit
  Resistor R1(R=10); VsourceAC AC; Ground G;
equation
  connect (AC.p,R1.p); connect (R1.n,AC.n);
  connect (AC.n,G.p);
end Circuit;
```

We introduce an additional equation ($i=23$) inside the Resistor component in order to over-constrain the simulation model. The flattened equations corresponding to the Circuit model is depicted in Figure 1.

eq1	R1.v = -R1.n.v + R1.p.v	var1	R1.p.v
eq2	0 = R1.n.i + R1.p.i	var2	R1.p.i
eq3	R1.i = R1.p.i	var3	R1.n.v
eq4	R1.i R1.R = R1.v	var4	R1.n.i
eq5	R1.i = 23	var5	R1.v
eq6	AC.v = -AC.n.v + AC.p.v	var6	R1.i
eq7	0 = AC.n.i + AC.p.i	var7	AC.p.v
eq8	AC.i = AC.p.i	var8	AC.p.i
eq9	AC.v = AC.VA*sin[2*time*AC.f*AC.PI]	var9	AC.n.v
eq10	G.p.v = 0	var10	AC.n.i
eq11	AC.p.v = R1.p.v	var11	AC.v
eq12	AC.p.i + R1.p.i = 0	var12	AC.i
eq13	R1.n.v = AC.n.v	var13	G.p.v
eq14	AC.n.v = G.p.v	var14	G.p.i
eq15	AC.n.i + G.p.i + R1.n.i = 0		

Figure 1. Flattened equations and variables corresponding to the Circuit model.

It should be noted that the number of equation is greater than the number of variables and therefore we are facing a structurally nonsingular problem.

3 Debugging Over- and Under-constrained Models

The methods proposed in this section present a strategy to deal with overdeterminacy by identifying the minimal set of equations that should be removed from the system in order to make the remaining set of equations solvable. The idea is to isolate the over-constraining part of the bipartite graph associated to the underlying system of equations and to perform reasoning based on specific properties of the specified subgraph. Efficient graph transformations, based on rules derived from the semantics of the modeling language are also performed on the subgraphs. We are going to show how these rules are automatically derived from the modeling language semantics and how the associated annotations to the equations contribute to the filtering of the combinatorial explosion of possible error fixing solutions. Those interested in more details may wish to consult Bunus and Fritzson 2004 [3] or Bunus 2004 [1].

Step 1: Isolating the over-constraining part.

In step 1, from the flattened intermediate form of the equations the associated bipartite graph is derived and a maximum cardinality matching is found. The Dulmage Mendelsohn canonical decomposition (Dulmage and Mendelsohn 1963 [4]) will lead to two different subgraphs: a well-constrained part W_G and an over-constrained part O_G^{1+} as depicted in Figure 2. The maximum cardinality matching is shown in Figure 2 with bold edges.

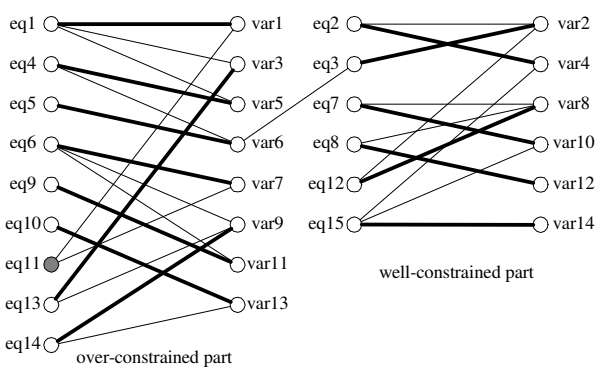


Figure 2. Canonical decomposition of an over-constrained system.

It can be seen that equation $eq11$ is not covered by the found maximum cardinality matching. Therefore equation $eq11$ is a non-saturated or free vertex of the equation set, therefore it is a source for the over-constrained part O_G^{1+} . Next, starting from $eq11$, the directed graph can be derived from the undirected bipartite graph, as illustrated in Figure 3, by exchanging all the matching edges into bidi-

rectional edges and orienting all other edges from equation to variable nodes. The layout of the directed graphs derived from the undirected bipartite graphs has been rearranged into a tree representation for the purpose of increasing understandability for the reader of the paper.

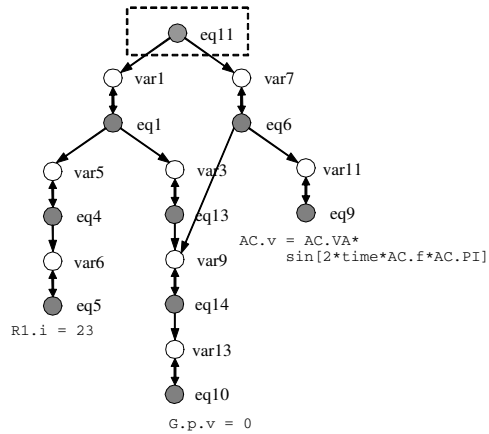


Figure 3. A directed graph associated to the over-constrained part.

Step 2: Reducing the over-constraining equations by using structural information.

The general error fixing strategy in the case of over-constrained equation subsystems is to remove the extra equations. An immediate fix to the over-constrained part is to remove one of the equation nodes, which will lead to a well-constrained part. However, as it can be seen from Figure 4, not all the equation edges can be safely removed.

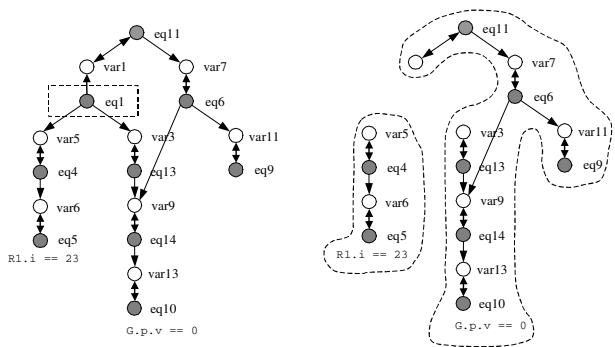


Figure 4. The elimination of an unsafe equation node ($eq1$) from the over-constrained subgraph (on the left) leads to two disconnected components (on the right).

By removing an equation node and the corresponding incident edges from the bipartite graph the remaining undirected graph must remain connected. In our particular example the set of over-constraining equations that satisfy this condition is $\{eq11, eq13, eq10, eq5, eq9\}$. It should be noted that the safe removal of equation nodes only refers to the bipartite graph representation of the intermediate code of the flattened set of equations, and it is influenced by only structural properties of the bipartite graph. If we would like to further reduce this set of equations,

removal criteria derived from the semantics of the modeling language would need to be developed and included in the debugging strategy.

Step 3: Reducing the over-constraining equations by using semantic information

As we have seen in the previous example not all the over-constraining equations are possible to remove without causing further structural failures in the model description. By taking into account simple rules derived from the language semantics we can safely discard some other elimination alternatives as well.

We note that equation *eq11* ($AC.p.v = R1.p.v$) is generated by a `connect` equation from the `Circuit` model and the only way to remove the equation *eq11* is to remove the original `connect(AC.p, R1.p)` equation. However, removing the above-mentioned equation will remove two equations from the flattened model since the `connect` equation expands into two equations. It is obvious that this modification cannot be performed by the user at the original source code level.

In order to provide a mechanism to reason about the erroneous model under consideration based on language semantics rules the equations need to be annotated. We define an annotated equation as a record with the following structure:

```
< Equation,
  Name,
  Description,
  No. of associated equations,
  Class name,
  Flexibility level,
  Connector generated,
  No. of linked equations
>
```

The *Class Name* indicates which class the equation comes from. This annotation is extremely useful in exactly locating the associated class of the equation and therefore providing concise error messages to the user in terms of original source code statements.

The *No. of associated eqs.* field defines the number of equations which are specified together with the annotated equation inside the same model. For an equation that belongs to the `TwoPin` class the number of associated equations is equal to 3. If one associated equation of the class needs to be eliminated the value is decremented by 1. During debugging, if the equation $R1.i * R1.R = R1.v$ is diagnosed to be an over-constraining equation and therefore needs to be eliminated, then the elimination is not possible because the model will be invalidated (the *No. of associated eqs.* cannot be equal to 0) and therefore other solutions need to be investigated.

The *flexibility level*, in a similar way as defined in Flannery and Gonzalez 1997 [5], allows the ranking of the relative importance of the equation in the overall flattened system of equations. The value can be in the range of 0 to 3, with 0 representing the most rigid equation and 3 being

the most flexible equation. In practice, it turns out that the equations generated by connections are more rigid from the constraint relaxation point of view than the equations specified inside the model. This means that preference is given to repair strategies that involve the removal of equations which defines the behavior of a particular component and not to topology changes of the circuit given by the connection equations. We set the flexibility value to 0 for those equations that should not be removed or modified. These equations are *locked* for editing which means that an automatic debugger should not consider any repair strategy that would involve the modification or the removal of the equations associated to such a component. For example the equations of components that come from well tested and trusted libraries can have this value set to zero.

The *Connector generated* is a `Boolean` attribute which tells whether the equation is generated or not by a `connect` equation. Usually these equations have a very low flexibility level.

The *No. of linked equations* attribute specifies how many other equations are linked with the current equations. Equations that come from `connect` equations or from parent objects (such as the `TwoPin` partial component) have this attribute greater than zero. Removing an intermediate equation that has this attribute greater than zero will trigger the removal of other intermediate additional equations equal to the number of linked equations. This is due to the fact that the removal of an intermediate equation is only possible by removing the original source code that generated that equation. By doing this all the generated intermediate equations by the original equation will be removed.

It is worth noting that the annotation attributes are automatically initialized by the static analyzer. These are incorporated in the front-end of the compiler, by using several graph representations of the declarative object-oriented program code. Therefore the user does not need to manually annotate the source code. A debugger pre-processor takes care of the automatic generation and initialization of the annotating code. In this way a mapping between the intermediate code and original declarative code is kept during the translation phases.

The annotations associated to the set of equivalent over-constraining equations $\{eq11, eq13, eq10, eq5, eq9\}$ are shown in Table 1.

Table 1. The associated annotations of the remaining over-constraining equation set

Name	Equation	No. of assoc. eqs.	Class name	Flex. level	Con. gen.	No. of linked eqs.
eq11	$AC.p.v=R1.p.v$	3	Circuit	1	Yes	1
eq13	$R1.n.v= AC.n.v$	3	Circuit	1	Yes	1
eq10	$G.p.v=0$	1	Ground	2	No	0
eq5	$R1.i=23$	2	Resistor	2	No	0
eq9	$AC.v=AC*VA*\sin..$	1	VsourceAC	2	No	0

The equation node *eq11* was already analyzed and can therefore be removed from the set. Equation node *eq13* is

removed as well, for the same reasons as equation *eq11*. By analyzing the remaining equations $\{eq10, eq5, eq9\}$, one should note that they have the same flexibility level and therefore candidates for elimination with equal probability. However, by analyzing the value of the *No. of associated eqs.* annotation, equation *eq10* and *eq9* have this attribute equal to one, which means that they are the only equations that define the behavior of the model. Removing one of these equations will invalidate the corresponding model component, which is probably not the intention of the modeler and therefore not acceptable as an error fixing solution.

By examining the annotations corresponding to equation *eq5* one can see that it can safely be removed because its flexibility level is high. The removal of *eq5* will not trigger the removal of any other equation since it has no linked equations (indicated by the value of *No. of linked eqs.* annotation which is equal to 0). Moreover, removing equation *eq5* will not invalidate the model since there is another equation defined inside the `Resistor` model ($R1.i * R1.R = R1.v$) denoted by the value of *No. of associated eqs.* annotation which is equal to 2.

Step 3: Outputting the debugging alternatives.

After selecting the right equation for elimination the debugger tries to identify the associated class of that equation based on the *Class name* parameter defined in the annotation structure. Having the class name and the intermediate equation form ($R1.i=23$), the original equation can be reconstructed ($i=23$) to exactly indicate to the user the equation that needs to be removed in order to make the simulation model well-constrained. In this case the debugger correctly located the faulty equation previously introduced by us in the simulation model.

When multiple valid error fixing solutions are possible and the debugger cannot decide which one to choose, a ranked list of error fixes is presented to the user for further analysis and decision. In those cases, the user must take the final decision, as the debugger cannot know or does not have enough information to decide which equation is over-constraining. The advantage of this approach is that

the debugger automatically identifies and solves several anomalies in the declarative simulation model specification without having to execute the system.

When debugging under-constrained systems (more variables than equations are present in the system) two distinct strategies can be considered. The first strategy considers the removal of the free variables while the second strategy considers the addition of new equations to the overall system of equations, which must contain the free variables. Additionally, the second strategy takes into account extra variables that can be added to the introduced new equation. New equations can be introduced at different levels in the object hierarchy.

4 Experimental Validation

In this paper we are interested in the quality of structural and semantics filtering rules employed in the proposed static debugging algorithm for correcting over- and under-constrained system of equations extracted from simulation models expressed in the Modelica language.

Firstly, we have modified several working simulation models by inserting additional equations in the model definitions at various places, thereby over-constraining the whole system models. In this first set of experiments we were interested if over-constraining situations are detected and how many repair possibilities are reported by the debugger.

A short description of the benchmark programs and the over-constraining nature for each example is given in Table 2. The measurements in Table 2. were performed as follows. We built several Modelica simulation models that were structurally correct. Then we have modified each example by inserting an extra equation in different components of the simulation model. In this way the models became over-constrained. During the translation phase the system of flattened equation and each equation was annotated. In the next step a canonical decomposition was performed on the structurally singular system of flat equations and the over-constraining graph was isolated. Based on the over-constraining graph the reduced set of

Table 2. Benchmark program description for over constrained systems.

Test model	Description	No. of var.	No of eq.	Over contr. part	Red. over constr. part	Semantic filtering	Debugging alt.
<i>scircuitR¹⁺</i>	A simple electrical circuit model consisting of a resistor connected in parallel with a continuous voltage source. The <code>Resistor</code> component is over-constrained by an extra equation.	14	15	9	5	1	1
<i>scircuitPin³⁺</i>	A simple electrical circuit model consisting two resistors connected in parallel with a direct current source. The <code>TwoPin</code> component is over-constrained by one extra equation.	20	23	19	15	3	1
<i>generatorR²⁺</i>	A generator circuit model similar where the <code>Resistor</code> component is over-constrained by one an extra equation.	49	51	35	22	6	3
<i>dcmotorR¹⁺</i>	A direct current motor circuit model where the <code>Resistor</code> component is over-constrained by one an extra equation.	36	37	31	29	7	7

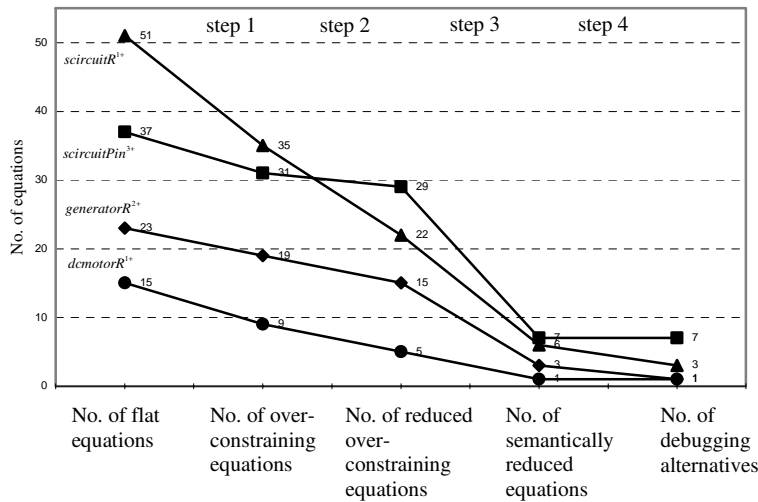


Figure 5. Number of over-constraining equations obtained after each reduction step during structural debugging.

over-constraining equations was computed. This set of equations was further reduced by using semantic filtering rules. Based on this final set of equation the error messages are output to the user. The numbers of debugging alternatives are shown in the last column of Table 2. Figure 5 depicts the number of over-constraining equations obtained after each reduction step.

Secondly, we investigated the detection capabilities of the static debugger when under-constrained situation were purposely introduced in the simulation model by deleting equations or adding extra variables in the system. The modifications performed on each model are described Table 3. The debugging of the under-constrained system was performed by considering only those corrections that imply the removal of a free variable from

Table 3. Benchmark program description for under-constrained systems

Test model	Description	No of eq.	No. of var.	Under contr. part	Red. over constr. part	Semantic filtering	Debugging alt.
scircuitR ⁺	A simple electrical circuit model consisting of a resistor connected in parallel with a continuous voltage source. In the Resistor component an extra variable has been declared (Real s) and the equation $R \cdot i = v \cdot s$ was introduced instead of the correct equation $R \cdot i = v$.	14	15	8	5	1	1
scircuitPin ⁺	A simple electrical circuit model consisting two resistors connected in parallel with a direct current source. The TwoPin component is under-constrained by introducing an extra variable (Real s) and by exchanging equations $0 = p.i + n.i$ with $s = p.i + n.i$.	20	23	7	4	3	1
generatorR ⁺	A generator circuit model. In the Resistor component an extra variable has been declared (Real s) and the equation $R \cdot i = v \cdot s$ was introduced instead of the correct equation $R \cdot i = v$.	49	51	28	21	3	1
dcmotorR ⁺	A direct current motor circuit model. In the Resistor component an extra variable has been declared (Real s) and the equation $R \cdot i = v \cdot s$ was introduced instead of the correct equation $R \cdot i = v$.	37	36	30	28	4	4

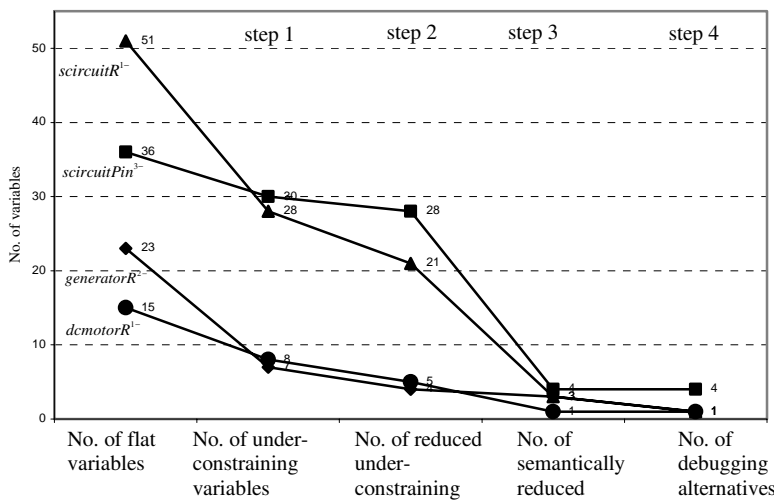


Figure 6. Number of under-constraining variables obtained after each reduction step during structural debugging.

the system. Figure 6 displays the number of under-constraining variables after each reduction step. After each step during the structural debugging the number of free variables that can be removed from the system is dramatically reduced. It should be noticed in Figure 6 that the largest reduction in the number of free variables and implicitly in the number of debugging alternatives presented to the user is achieved by the semantic filtering phase.

We are interested in the quality of structural and semantics filtering rules employed in the proposed static debugging algorithm for correcting over- and under-constrained system of equations extracted from simulation models ex-

pressed in the Modelica language. Table 4 shows the percentage reduction in the number of equations/variables that need to be examined by user after each step in the debugging process.

Table 4. Percentage reduction of the number of equation/variables that need to be examined by the user after each reduction step.

Test model	No. of flat eq./var	Step1	Step 2	Step3	Step 4
<i>scircuitR</i> ¹⁺	15	40.0%	66.7%	93.3%	93.3%
<i>scircuitPin</i> ³⁺	23	17.4%	34.8%	87.0%	95.7%
<i>generatorR</i> ²⁺	51	31.4%	56.9%	88.2%	94.1%
<i>dcmotorR</i> ¹⁺	37	16.2%	21.6%	81.1%	81.1%
<i>scircuitR</i> ¹⁻	15	46.7%	66.7%	93.3%	93.3%
<i>scircuitPin</i> ³⁻	23	69.6%	82.6%	87.0%	95.7%
<i>generatorR</i> ²⁻	51	45.1%	58.8%	94.1%	98.0%
<i>dcmotorR</i> ¹⁻	36	16.7%	22.2%	88.9%	88.9%

As can be seen in Figure 5, Figure 6 and from the percentage reduction Table 4, the proposed algorithm for debugging over- and under-constrained systems is very efficient in reducing the number of debugging alternative shown to the user. On the average, 91% of the irrelevant candidates were eliminated, which allows the user to look for the bug among the few remaining candidates, thus dramatically improving bug localization effectiveness.

5 Implementation

For the previously presented graph decomposition techniques to be useful in practice, we must be able to construct and manage the graph representation of equation-based specifications efficiently and integrate them into an automatic or semi-automatic debugging tool. The use of graph-based tools in structural analysis is of great interest both in displaying properties of systems of equations and also in following and performing symbolic manipulations of variables and equations when modeling with equation-based languages (Harman 2005 [6]). We show how existing graph theoretical decomposition techniques can be adapted and integrated into debugging tools integrated into simulation environments that employs such languages.

At this stage we are able to provide an overview of the proposed framework developed for the Modelica language and Modelica-based simulation environments. Even if we have limited our prototype implementation to the Modelica language, the developed debugging kernels can easily be adapted to handle other object-oriented equation-based languages as well. It is important to note that the proposed debugging framework can easily be integrated into the existing Modelica compilers.

AMOEBAs (Automatic Modelica Equation-Based Analyzer) is the static analysis module that we have designed and implemented in order to attach it to a Modelica

compiler. The tool is able to successfully detect and provide error-fixing solutions for typical over and under-constrained situations, which might appear during the modeling stage using Modelica. Figure 7 show the general architecture of our static debugger.

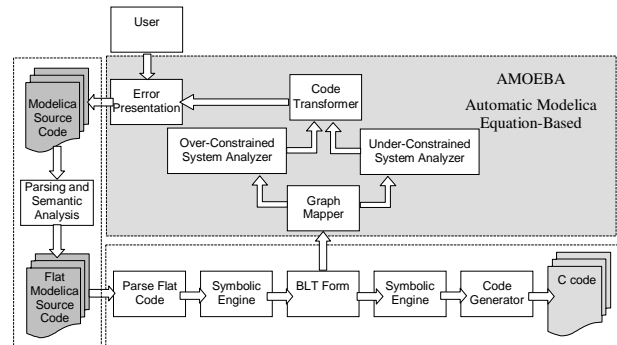


Figure 7. AMOEBAs integration into the compilation framework.

Below we present each phase of the static analysis with the corresponding module:

The flattened equations are transformed into the bipartite graph representation by a *Graph Mapping* module. The canonical decomposition algorithm applied by the *BLT* module in the compiler splits the graph into three distinct subgraphs corresponding to an over-constrained system of equations (too many equations are present), an under-constrained system (too few equations or too many variables are present in the system) and a well-constrained system of equations (the number of variables is equal to the number of equations). A simple heuristic filtering rule assumes that the well-constrained part obtained after decomposition will lead to a solvable system of equations and therefore need not be included in any repair strategy. If under- or over-constrained situations are detected, this means that there are some inconsistencies in the model.

The *Over- and Under-Constrained System Analyzers* applies the algorithms presented in previous sections, in order to transform these graphs into a well-constrained graph and elaborate the necessary program modifications.

The *Code Transformer* module needs to validate the program correction: it must assure that there exists a semantically correct source code program that can be translated into the intermediate program correction. The source code transformations must be performed only by using atomic changes at the original source code level. Finally, the error fixing solution is output by the debugger in terms of atomic changes that need to be performed on the original source code in order to obtain a valid original source code program that will generate the corresponding program modifications at the intermediate code level. When multiple error fixing solutions exist, the annotations attached to the flattened equations are used in the process of eliminating some of the modifications and prioritizing the remaining ones.

The *Error Presentation* module is responsible for presenting error messages to the user based on the previously

obtained valid source code modifications. Before being presented to the user, the output is filtered. For example, all the modifications that would involve atomic changes on locked components are eliminated and the remaining corrections are ranked based on equations annotations. This module handles most of the user interaction necessary for the debugger to complete the missing formal specification of the program. At this level the user can be confronted with several error fixing corrections that will eliminate the symptom of the detected inconsistency at the intermediate code level. The corrections that most closely correspond to the programmer's view of the model structure should be selected.

6 Conclusions

Structural analysis techniques are widely used for assessing the correctness and the credibility of mathematical models expressed with the help of equations. Experience has taught us that pre-processing a system of equations pays high dividends by reducing the time for finding inconsistencies and efficiently correcting them. From the user point of view, such techniques are extremely beneficial because they provide guidance during early stages of the simulation model building process and do not require solving the equations system.

The paper illustrates that it is possible to localize and repair a significant number of errors during static analysis of object-oriented equation-based modeling languages without having to execute the simulation model. In this way certain numerical failures can be avoided later during the execution process. The paper proves that the result of structural static analysis performed on the underlying system of equations can effectively be used to statically debug Modelica simulation models.

This paper describes one of the first experimental studies on how these new static debugging techniques perform on erroneous model examples. We have presented an empirical evaluation of proposed static analysis based debugging paradigm for equation-based languages. Our studies demonstrated that static analysis can dramatically reduce debugging time, suggesting the potential of structural analysis as a highly effective approach.

Currently, the debugger's functionality is limited mostly due to our inability to compile the full Modelica language. Therefore only a limited number of real world examples with limited size and complexity have been tested. The integration of the presented debugging tech-

niques into the Open Source Modelica framework is underway. In order to provide a complete debugging framework for the Modelica language we intent to integrate the proposed structural analysis techniques with the existing debugger for the algorithmic subset of the Modelica language proposed by Pop and Fritzson 2005 [7].

We claim that the techniques developed and proposed in this paper are suitable for a wide range of equation-based languages and not only for the Modelica language. These techniques can be easily adapted to the specifics of a particular simulation environment. Our claim is based on the *close integration of the developed debugging techniques and the compilation process*. Most of the existing compilers for equation based languages share the same principles.

Acknowledgements

This research was supported by Center for Industrial Information Technology (grant CENIIT 05.02) at Linköping University Sweden.

REFERENCES

- [1] Bunus Peter. (2004). Debugging Techniques for Equation-Based Languages. PhD Thesis. Department of Computer and Information Science, Linköping University, 2004.
- [2] Bunus Peter and Peter Fritzson. (2003). "Semi-automatic Fault Localization and Behaviour Verification for Physical System Simulation Models." In Proceedings of the 18th IEEE International Conference on Automated Software Engineering. (Montreal, Canada, October 6-10, 2003).
- [3] Bunus Peter and Peter Fritzson. (2004) "Automated Static Analysis of Equation-Based Components." Simulation: Transactions of the Society for Modeling and Simulation International. Special Issue on Component Based Modeling and Simulation., vol. 80: 8, August 2004.
- [4] Dulmage A.L. and N.S. Mendelsohn. (1963) "Coverings of bipartite graphs." Canadian J. Math, vol. 10, pp. 517-534.
- [5] Flannery L. M. and A. J. Gonzalez. (1997) "Detecting Anomalies in Constraint-based Systems." Engineering Applications of Artificial Intelligence, vol. 10: 3, pp. 257-268.
- [6] Harman Peter. (2005). " Visualisation of Model Transformation Algorithms for a Modelica Translator." In Proceedings of the 4th International Modelica Conference. (Hamburg, Germany, 7-8 March, 2005).
- [7] Pop Adrian and Peter Fritzson. (2005). "A Portable Debugger for Algorithmic Modelica Code." In Proceedings of the 4th International Modelica Conference. (Hamburg, Germany, 7-8 March, 2005).