



2006

**Proceedings of the
5th International Modelica Conference**

September 4th–5th, 2006
arsenal research
Vienna, Austria

Dr. Christian Kral (Conference Chair)
Anton Haumer (Program Chair)

Volume 2

organized by
The Modelica Association
and arsenal research

All papers of this conference can be downloaded from
<http://www.modelica.org/events/modelica2006>

Proceedings of Modelica2006

arsenal research

Vienna, Austria, September 4th-5th, 2006

Conference Chair: Dr. Christian Kral

Program Chair: Anton Haumer

Published by:

The Modelica Association (<http://www.modelica.org/>) and
arsenal research (<http://www.arsenal.ac.at/>)

Preface

The first International Modelica Conference took place October 2000, in Lund, Sweden. Since then, Modelica has increasingly become the preferred modeling language for complex multi-domain systems. During this time, the community of Modelica users has grown continuously. This is also reflected in the great response to the Call for Papers of the 5th International Modelica Conference. This year's conference will be held on September 4th-5th, 2006 in Vienna. From the excellent papers submitted to the program committee, it was finally decided to include 66 oral and 15 poster presentations in the technical program. The technical papers cover thermodynamic and automotive applications, mechanical and electrical systems and the latest developments in modelling and simulation products. Before the conference, there will be five parallel tutorials. These tutorials include an introduction to Modelica, mathematical aspects of modeling, as well as the modeling of electric drives, vehicle and thermodynamic systems.

Due to the special features of the Modelica language, such as object-oriented modeling and the ability to reuse and exchange models, Modelica strongly supports an integrated engineering design process. This fact is emphasized by the keynote of Dominique Florack, Executive Vice President R&D of Dassault Systemes, "About the strategic decision of Dassault Systemes to select Modelica to be at the core of Dassault Systemes' open strategy for CATIA Systems". In various fields Modelica is being used as a standard platform for model exchange between suppliers and OEMs.

A key issue for the success of Modelica is the continuous development of the Modelica language as well as the Modelica Standard Library by the Modelica Association under strict observance of backward compatibility with 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 5th International Modelica Conference was organized by the Modelica Association and arsenal research, Vienna, Austria. We would like to thank the local organizing committee, the technical program committee and the reviewers for offering their time and expertise throughout the organization of the conference. We would also like to wish all participants an excellent and interesting conference and hope you will have a memorable experience in Vienna.

Vienna, September 1st, 2006

Dr. Christian Kral
Conference Chair

Anton Haumer
Program Chair

Program Committee

- Conference Chair: Dr. Christian Kral, arsenal research, Vienna, Austria
- Program Chair: Anton Haumer, arsenal research, Vienna, Austria
- Program Board: Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
- Program Board: Prof. Peter Fritzson, Linköping University, Sweden
- Program Board: Dr. Hilding Elmqvist, Dynasim AB, Lund, Sweden
- Program Board: Dr. Michael Tiller, Emmeskay Inc., Michigan, USA
- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Germany
- Dr. Ingrid Bausch-Gall, Bausch-Gall GmbH, Munich, Germany
- Daniel Bouskela, Electricite de France, Chatou Cedex, France
- Prof. Felix Breitenecker, Technical University Vienna, Austria
- Dr. Francesco Casella, Politecnico di Milano, Cremona, Italy
- Thomas Christ / Marco Bross, BMW, Munich, Germany
- Dr. Ruediger Franke, ABB, Heidelberg, Germany
- Dirk Limperich, DaimlerChrysler AG, Sindelfingen, Germany
- Prof. Karin Lunde, University of Applied Sciences Ulm, Germany
- Ludwig Marvan, DRIVEScom, Vienna, Austria
- Dr. Jakob Mauss, DaimlerChrysler AG, Berlin, Germany
- Gert Pascoli, arsenal research, Vienna, Austria
- Franz Pirker, arsenal research, Vienna, Austria
- Markus Plainer, arsenal research, Vienna, Austria
- Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Germany
- Dr. Hubertus Tummescheit, Modelon AB, Lund, Sweden

Local Organizing Committee

- Anton Haumer
- Dr. Christian Kral
- Franz Pirker
- Veronika Roscher
- Silke Schrödl
- WEBSTRACTS on-line Conference Management
- procon Conference, Incentive & Event Management GmbH

Table of Contents

Volume 1

Session 1a: Thermodynamic Systems for Power Plant Applications 1	1
Fast Start-up of a Combined-Cycle Power Plant: A Simulation Study with Modelica	3
F. Casella ^[1] , F. Pretolani ^[2]	
^[1] Politecnico di Milano, Italy, ^[2] CESI S.p.A., Italy	
Modelling of a Water/Steam Cycle of the Combined Cycle Power Plant “Rio Bravo 2” with Modelica	11
B. El Hefni, D. Bouskela	
EDF R&D, France	
Modeling and Dynamic Analysis of CO ₂ -Emission Free Power Processes in Modelica using the CombiPlant Library	17
J. Eborn ^[1] , F. Selimovic ^[2] , B. Sundén ^[2]	
^[1] Modelon AB, Sweden, ^[2] Lund Institute of Technology, Sweden	
Session 1b: Automotive Applications 1	23
Simulation of Hybrid Electric Vehicles	25
D. Simic, H. Giuliani, C. Kral, J.V. Gragger	
arsenal research, Austria	
Coordinated Automotive Libraries for Vehicle System Modelling	33
M. Dempsey ^[1] , M. Gäfvert ^[2] , P. Harman ^[3] , C. Kral ^[4] , M. Otter ^[5] , P. Treffinger ^[6]	
^[1] Claytex Services Ltd., UK, ^[2] Modelon AB, Sweden, ^[3] Ricardo UK Ltd., UK,	
^[4] arsenal research, Austria, ^[5] DLR Oberpfaffenhofen, Germany, ^[6] DLR Stuttgart, Germany	
The VehicleDynamics Library - Overview and Applications	43
J. Andreasson, M. Gäfvert	
Modelon AB, Sweden	
Session 1c: Language, Tools and Algorithms 1	53
Modelica CVD - A Tool for Visualizing the Structure of Modelica Libraries	55
M. Loeffler ^[1] , M. Huhn ^[1] , C.C. Richter ^[1] , R. Kossel ^[2]	
^[1] TU Braunschweig, Germany, ^[2] TLK-Thermo GmbH, Germany	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
C. Nytsch-Geusen ^[1] , T. Ernst ^[1] , A. Nordwig ^[1] , P. Schwarz ^[2] , P. Schneider ^[2] , M. Vetter ^[3] ,	
C. Wittwer ^[3] , A. Holm ^[4] , T. Nouidui ^[4] , J. Leopold ^[5] , G. Schmidt ^[5] , A. Mattes ^[6]	
^[1] Fraunhofer FIRST, Germany, ^[2] Fraunhofer IIS/EAS, Germany, ^[3] Fraunhofer ISE, Germany,	
^[4] Fraunhofer IBP, Germany, ^[5] Fraunhofer IWU, Germany, ^[6] Fraunhofer IPK, Germany	
Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism	73
T. Beltrame ^[1] , F.E. Cellier ^[2]	
^[1] VTI, Finland, ^[2] ETH Zurich, Switzerland	

Session 1d: Mechanical Systems and Applications 1	83
The DLR FlexibleBodies Library to Model Large Motions of Beams and of Flexible Bodies Exported from Finite Element Programs	85
A. Heckmann ^[1] , M. Otter ^[1] , S. Dietz ^[2] , J.D. Lopez ^[3] ^[1] German Aerospace Center (DLR), Germany, ^[2] INTEC GmbH, Germany, ^[3] Dynasim AB, Sweden	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method.....	97
X. Murua, F. Martinez, A. Pujana, J. Basurko, J.M. Pagalday IKERLAN Research Centre, Spain	
A Modelica Library for Space Flight Dynamics	107
T. Pulecchi, F. Casella, M. Lovera Politecnico di Milano, Italy	
Session 2a: Thermodynamic Systems for Power Plant Applications 2	117
Simulation of Components of a Thermal Power Plant	119
R. Schimon, D. Simic, A. Haumer, C. Kral, M. Plainier arsenal research, Austria	
Pressurized Water Reactor Modelling with Modelica	127
A. Souyri ^[1] , D. Bouskela ^[1] , B. Pentori ^[2] , N. Kerkar ^[2] ^[1] Electricité de France EDF/R&D, France, ^[2] Electricité de France EDF/SEPTEN, France	
Simulation of the Start-Up Procedure of a Parabolic Trough Collector Field with Direct Solar Steam Generation	135
T. Hirsch, M. Eck German Aerospace Center, Institute of Technical Thermodynamics, Germany	
Session 2b: Automotive Applications 2	145
Modeling the Dynamics of Vehicle Fuel Systems.....	147
J.J. Batteh, P.J. Kenny Ford Motor Company, USA	
Motorcycle Dynamics Library in Modelica.....	157
F. Donida , G. Ferretti, S.M. Savaresi, F. Schiavo, M. Tanelli Politecnico di Milano, Italy	
Development and Verification of a Series Car Modelica/Dymola Multi-body Model to Investigate Vehicle Dynamics Systems	167
C. Knobel ^[1] , G. Janin ^[2] , A. Woodruff ^[3] ^[1] BMW Group Research and Technology, Germany, ^[2] École Nationale Supérieure de Techniques Avancées, France, ^[3] Modelon AB, Sweden	
Session 2c: Language, Tools and Algorithms 2	175
Modeling and Simulation of Differential Equations in Scicos	177
M. Najafi, R. Nikoukhah INRIA-Rocquencourt, France	
How to Dissolve Complex Dynamic Systems for Wanted Unknowns with Dymola / Modelica.....	187
J. Koehler ZF Friedrichshafen AG, Germany	

Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel	193
F.H.A. Claeys ^[1] , P. Fritzson ^[2] , P.A. Vanrolleghem ^[3]	
^[1] BIOMATH, Ghent University, Belgium, ^[2] PELAB, Linköping University, Sweden,	
^[3] modelEAU, Université Laval, Canada	
Session 2d: Mechanical Systems and Applications 2	203
Leaf Spring Modeling.....	205
N. Philipson	
Modelon AB, Sweden	
Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation	213
I.I. Kosenko ^[1] , M.S. Loginova ^[2] , YA.P. Obraztsov ^[2] , M.S. Stavrovskaya ^[1]	
^[1] Moscow State University of Service, Russian Federation,	
^[2] Moscow State Academy of Instrument Making and Computer Science, Russian Federation	
NowaitTransit Concept Assessment. Modeling of Trains on Complex Track Geometry	225
J. Tuszynski ^[1] , N. Philipson ^[2] , J. Andreasson ^[2] , M. Gäfvert ^[2]	
^[1] Nowaittransit AB, Sweden, ^[2] Modelon AB, Sweden	
Session 3a: Thermodynamic Systems for Energy Storage and Conversion	233
Analysis of Steam Storage Systems using Modelica.....	235
J. Buschle, W.D. Steinmann, R. Tamme	
German Aerospace Center (DLR), Germany	
An Enhanced Discretisation Method for Storage Tank Models within Energy Systems	243
S. Wischhusen	
XRG Simulation GmbH, Germany	
HydroPlant – a Modelica Library for Dynamic Simulation of Hydro Power Plants.....	251
K. Tuszynski ^[1] , J. Tuszynski ^[2] , K. Slättorp ^[3]	
^[1] Modelon AB, Sweden, ^[2] Datavoice HB, Sweden, ^[3] Tactel AB, Sweden	
Session 3b: Hardware in the Loop	259
Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation	261
A. Ebner, A. Haumer, D. Simic, F. Pirker	
arsenal research, Austria	
Parameterisation of Modelica Models on PC and Real Time Platforms	267
M. Kellner ^[1] , M. Neumann ^[1] , A. Banerjee ^[1] , P. Doshi ^[2]	
^[1] ZF Friedrichshafen AG, Germany, ^[2] Universität Duisburg-Essen, Germany	
Synchronising a Modelica Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System	275
D. Winkler, C. Gühmann	
Technische Universität Berlin, Germany	
Session 3c: Language, Tools and Algorithms 3	283
A Numeric Library for Use in Modelica Simulations with Lapack, SuperLU, Interpolation, and MatrixIO.....	285
A. Sandholm ^[1,2] , P. Bunus ^[1] , P. Fritzson ^[1]	
^[1] Linköping University, Sweden, ^[2] Kalmar University, Sweden	
Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA...293	
R. Franke ^[1] , J. Doppelhamer ^[2]	
^[1] ABB AG, Power Technology Systems, Germany, ^[2] ABB Corporate Research, Germany	

Types in the Modelica Language.....	303
D. Broman ^[1] , P. Fritzson ^[1] , S. Furic ^[2]	
^[1] Linköping University, Sweden, ^[2] Imagine, France	
Session 3d: Electric Systems and Applications 1	317
Modeling and Simulation of Generator Circuit Breaker Performance	319
O. Fritz ^[1] , M. Lakner ^[2]	
^[1] ABB Switzerland Ltd., Corporate Research, Switzerland,	
^[2] ABB Switzerland Ltd., High-Current Systems, Switzerland	
Parallel Simulation with Transmission Lines in Modelica	325
K. Nyström, P. Fritzson	
Linköping University, Sweden	

Volume 2

Session 4: Poster Session	333
GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms	335
M.A. Rubio ^[1] , A. Urquia ^[2] , L. González ^[1] , D. Guinéa ^[1] , S. Dormido ^[2]	
^[1] Instituto de Automática Industrial (IAI), CSIC, Spain,	
^[2] ETS de Ingeniería Informática, UNED, Spain	
Ascola: A Tool for Importing Dymola Code into Ascet.....	343
C. Schlegel ^[1] , R. Finsterwalder ^[2]	
^[1] Schlegel Simulation GmbH, Germany,	
^[2] University of the Federal Armed Forces Munich, Germany	
An Analyzer for Declarative Equation Based Models.....	349
J.-W. Ding ^[1] , L.-P. Chen ^[1] , F.-L. Zhou ^[1] , Y.-Z. Wu ^[1] , G.B. Wang ^[2]	
^[1] Huazhong University of Science and Technology, China,	
^[2] National Natural Science Foundation of China, China	
Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools	359
O. Johansson, A. Pop, P. Fritzson	
Linköping University, Sweden	
On the Noise Modelling and Simulation	369
D. Aiordachioaie, V. Nicolau, M. Munteanu, G. Sirbu	
Dunarea de Jos Galati University, Romania	
Acausal Modelling of Helicopter Dynamics for Automatic Flight Control Applications	377
L. Viganò, G. Magnani	
Politecnico di Milano, Italy	
Dynamic Modeling and Control of a 6 DOF Parallel Kinematics	385
M. Krabbes, Ch. Meissner	
Leipzig University of Applied Sciences, Germany	
Modelling of Alternative Propulsion Concepts of Railway Vehicles	391
H. Dittus, J. Ungethüm	
German Aerospace Center, Institute of Vehicle Concepts, Germany	
Modelling Automotive Hydraulic Systems using the Modelica ActuationHydraulics Library	399
P.A. Harman	
Ricardo UK Ltd., UK	

Vehicle Model for Transient Simulation of a Waste-Heat-Utilisation-Unit Containing Extended PowerTrain and Fluid Library Components.....	405
M. Eschenbach, J. Ungethüm, P. Treffinger German Aerospace Center, Germany	
Modeling, Calibration and Control of a Paper Machine Dryer Section.....	411
J. Åkesson ^[1] , O. Slättke ^[2] ^[1] Lund University, Sweden, ^[2] ABB Ltd., Ireland	
System and Component Design of Directly Driven Reciprocating Compressors with Modelica	421
T. Bödrich Dresden University of Technology, Germany	
Multizone Airflow Model in Modelica.....	431
M. Wetter United Technologies Research Center, USA	
Modelling of a Solar Thermal Reactor for Hydrogen Generation	441
J. Dersch, A. Mathijssen, M. Roeb, C. Sattler Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Germany	
Object Oriented Modelling of DISS Solar Thermal Power Plant.....	449
L.J. Yebra ^[1] , M. Berenguel ^[2] , E. Zarza ^[1] , S. Dormido ^[3] ^[1] C.I.E.M.A.T., Spain, ^[2] Universidad de Almería, Spain, ^[3] U.N.E.D., Spain	
Session 5a: Language, Tools and Algorithms 4.....	457
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging	459
A. Pop, P. Fritzson, A. Remar, E. Jagudin, D. Akhvlediani Linköping University, Sweden	
A Modelica Based Format for Flexible Modelica Code Generation and Causal Model Transformations.....	467
J. Larsson, P. Fritzson Linköping University, Sweden	
Dymola interface to Java - A Case Study: Distributed Simulations	477
J.D. Lopez, H. Olsson Dynasim AB, Sweden	
Simulation of Complex Systems using Modelica and Tool Coupling.....	485
R. Kossel, W. Tegethoff, M. Bodmann, N. Lemke TLK-Thermo GmbH, Germany	
Session 5b: Thermodynamic Systems for Cooling Applications.....	491
Optimization of a Cooling Circuit with a Parameterized Water Pump Model	493
D. Simic, C. Kral, H. Lacher arsenal research, Austria	
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
C.C. Richter, C. Tischendorf, R. Fiorenzano, P. Cavalcante, W. Tegethoff, J. Köhler TU Braunschweig, Germany	
Modeling of Frost Growth on Heat Exchanger Surfaces.....	509
K. Proelss, G. Schmitz Hamburg University of Technology, Germany	

Multizone Building Model for Thermal Building Simulation in Modelica.....	517
M. Wetter	
United Technologies Research Center, USA	
Session 5c: Free and Commercial Libraries 1	527
The LinearSystems Library for Continuous and Discrete Control Systems.....	529
M. Otter	
German Aerospace Center (DLR), Germany	
ARENALib: A Modelica Library for Discrete-Event System Simulation	539
V.S. Prat, A. Urquia, S. Dormido	
ETS de Ingeniería Informática, UNED, Spain	
Neural Network Library in Modelica	549
F. Codecà, F. Casella	
Politecnico di Milano, Italy	
The Modelica Multi-bond Graph Library	559
D. Zimmer, F.E. Cellier	
ETH Zürich, Switzerland	
Session 5d: Electric Systems and Applications 2	569
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
J.V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, F. Pirker	
arsenal research, Austria	
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
O. Enge ^[1] , C. Clauß ^[1] , P. Schneider ^[1] , P. Schwarz ^[1] , M. Vetter ^[2] , S. Schwunk ^[2]	
^[1] Fraunhofer Institute Integrated Circuits, Germany,	
^[2] Fraunhofer Institute Solar Energy Systems, Germany	
Identification and Controls of Electrically Excited Synchronous Machines	589
H. Kapeller, A. Haumer, C. Kral, F. Pirker, G. Pascoli	
arsenal research, Austria	
Session 6a Language, Tools and Algorithms 5	597
Dynamic Optimization of Energy Supply Systems with Modelica Models	599
C. Hoffmann, H. Puta	
Technische Universität Ilmenau, Germany	
Robust Initialization of Differential Algebraic Equations	607
B. Bachmann ^[1] , P. Aronsson ^[2] , P. Fritzson ^[2]	
^[1] University of Applied Sciences, Germany, ^[2] Linköping University, Sweden	
Calibration of Static Models using Dymola	615
H. Olsson ^[1] , J. Eborn ^[2] , S.E. Mattsson ^[1] , H. Elmqvist ^[1]	
^[1] Dynasim AB, Sweden, ^[2] Modelon AB, Sweden	
Automatic Fixed-point Code Generation for Modelica using Dymola	621
U. Nordström ^[1,2] , J. D. Lopez ^[1] , H. Elmqvist ^[1]	
^[1] Dynasim AB, Sweden, ^[2] Lund Institute of Technology, Sweden	

Session 6b: Thermodynamic Systems and Applications	629
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks	631
F. Casella ^[1] , M. Otter ^[2] , K. Proelss ^[3] , C. Richter ^[4] , H. Tummescheit ^[5]	
^[1] Politecnico di Milano, Italy, ^[2] German Aerospace Center (DLR), Germany,	
^[3] Technical University Hamburg-Harburg, Germany,	
^[4] Technical University Braunschweig, Germany, ^[5] Modelon AB, Sweden	
Shock Wave Modeling for Modelica.Fluid Library using Oscillation-free Logarithmic Reconstruction.....	641
J. D. Lopez	
Dynasim AB, Sweden	
Modelling of an Experimental Batch Plant with Modelica	651
K. Poschlad ^[1] , M.A.P. Remelhe ^[1] , M. Otter ^[2]	
^[1] University of Dortmund, Germany, ^[2] German Aerospace Center (DLR), Germany	
Integral Analysis for Thermo-Fluid Applications with Modelica	661
J.J. Batteh	
Ford Motor Company, Research and Advanced Engineering, USA	
Session 6c: Free and Commercial Libraries 2	669
Integration of CATIA with Modelica	671
P. Bhattacharya ^[1] , N. Suyam Welakwe ^[2] , R. Makanaboyina ^[1] , A. Chimalakonda ^[1]	
^[1] DaimlerChrysler Research and Technology, India,	
^[2] DaimlerChrysler Research and Technology, Germany	
A Modelica Library for Simulation of Household Refrigeration Appliances - Features and Experiences.....	677
C. Heinrich, K. Berthold	
Institute for Air Conditioning and Refrigeration, Germany	
A New Energy Building Simulation Library	685
J.I. Videla, B. Lie	
Telemark University College, Norway	
UnitTesting: A Library for Modelica Unit Testing	695
M.M. Tiller, B. Kittirungsi	
Emmeskay, Inc., USA	
Session 6d: Multidomain Systems	705
If We Only had Used XML	707
U. Reisenbichler, H. Kapeller, A. Haumer, C. Kral, F. Pirker, G. Pascoli	
arsenal research, Austria	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
P. Matthes, T. Haase, A. Hoh, T. Tschirner, D. Müller	
TU Berlin, Germany	
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica.....	725
F.-L. Zhou, L.-P. Chen, Y.-Z. Wu, J.-W. Ding, J.-J. Zhao, Y.-Q. Zhang	
Huazhong University of Science and Technology, China	

Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of
Multi-domain Physical Systems Based on Modelica.....733
Y.-Z. Wu, F.-L. Zhou, L.-P. Chen, J.-W. Ding, J.-J. Zhao
Huazhong University of Science and Technology, China

Index of Authors

Aiordachioaie, D.	
On the Noise Modelling and Simulation	369
Åkesson, J.	
Modeling, Calibration and Control of a Paper Machine Dryer Section.....	411
Akhvlediani, D.	
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging.....	459
Andreasson, J.	
NowaitTransit Concept Assessment. Modeling of Trains on Complex Track Geometry	225
The VehicleDynamics Library - Overview and Applications	43
Aronsson, P.	
Robust Initialization of Differential Algebraic Equations	607
Bachmann, B.	
Robust Initialization of Differential Algebraic Equations	607
Banerjee, A.	
Parameterisation of Modelica Models on PC and Real Time Platforms	267
Basurko, J.	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method.....	97
Batteh, J.J.	
Integral Analysis for Thermo-Fluid Applications with Modelica	661
Modeling the Dynamics of Vehicle Fuel Systems.....	147
Bäumel, T.	
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
Beltrame, T.	
Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism.....	73
Berenguel, M	
Object Oriented Modelling of DISS Solar Thermal Power Plant.....	449
Berthold, K	
A Modelica Library for Simulation of Household Refrigeration Appliances - Features and Experiences	677
Bhattacharya, P.	
Integration of CATIA with Modelica	671
Bodmann, M	
Simulation of Complex Systems using Modelica and Tool Coupling.....	485
Bödrich, T.	
System and Component Design of Directly Driven Reciprocating Compressors with Modelica	421
Bouskela, D.	
Modelling of a Water/Steam Cycle of the Combined Cycle Power Plant “Rio Bravo 2” with Modelica	11
Pressurized Water Reactor Modelling with Modelica	127
Broman, D.	
Types in the Modelica Language.....	303

Bunus, P.	
A Numeric Library for Use in Modelica Simulations with Lapack, SuperLU, Interpolation, and MatrixIO	285
Buschle, J.	
Analysis of Steam Storage Systems using Modelica.....	235
Casella, F.	
A Modelica Library for Space Flight Dynamics.....	107
Fast Start-up of a Combined-Cycle Power Plant: A Simulation Study with Modelica	3
Neural Network Library in Modelica	549
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks	631
Cavalcante, P.	
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Cellier, F.E.	
Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism.....	73
The Modelica Multi-bond Graph Library.....	559
Chen, L.-P.	
An Analyzer for Declarative Equation Based Models.....	349
Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica.....	733
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica	725
Chimalakonda, A.	
Integration of CATIA with Modelica	671
Claeys, F.H.A.	
Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel	193
Clauß, C.	
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Codecà, F.	
Neural Network Library in Modelica	549
Dempsey, M.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
Dersch, J.	
Modelling of a Solar Thermal Reactor for Hydrogen Generation	441
Dietz, S.	
The DLR FlexibleBodies Library to Model Large Motions of Beams and of Flexible Bodies Exported from Finite Element Programs.....	85
Ding, J.-W.	
An Analyzer for Declarative Equation Based Models.....	349
Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica.....	733
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica	725
Dittus, H.	
Modelling of Alternative Propulsion Concepts of Railway Vehicles.....	391
Donida , F	
Motorcycle Dynamics Library in Modelica.....	157

Doppelhamer, J.	
Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA...	293
Dormido, S.	
ARENALib: A Modelica Library for Discrete-Event System Simulation	539
GAPILib - A Modelica Library for Model Parameter Identification	
Using Genetic Algorithms	335
Object Oriented Modelling of DISS Solar Thermal Power Plant.....	449
Doshi, P.	
Parameterisation of Modelica Models on PC and Real Time Platforms	267
Ebner, A.	
Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation	261
Eborn, J.	
Calibration of Static Models using Dymola	615
Modeling and Dynamic Analysis of CO ₂ -Emission Free Power Processes in Modelica	
using the CombiPlant Library.....	17
Eck, M.	
Simulation of the Start-Up Procedure of a Parabolic Trough Collector Field with Direct Solar	
Steam Generation.....	135
El Hefni, B.	
Modelling of a Water/Steam Cycle of the Combined Cycle Power Plant	
“Rio Bravo 2” with Modelica	11
Elmqvist, H.	
Automatic Fixed-point Code Generation for Modelica using Dymola.....	621
Calibration of Static Models using Dymola	615
Enge, O.	
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Ernst, T.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Eschenbach, M.	
Vehicle Model for Transient Simulation of a Waste-Heat-Utilisation-Unit Containing Extended	
PowerTrain and Fluid Library Components	405
Ferretti, G.	
Motorcycle Dynamics Library in Modelica.....	157
Finsterwalder, R.	
Ascola: A Tool for Importing Dymola Code into Ascet.....	343
Fiorenzano, R.	
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Franke, R.	
Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA...	293
Fritz, O.	
Modeling and Simulation of Generator Circuit Breaker Performance	319

Fritzson, P.	
A Modelica Based Format for Flexible Modelica Code Generation and Causal Model Transformations	467
A Numeric Library for Use in Modelica Simulations with Lapack, SuperLU, Interpolation, and MatrixIO	285
Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools	359
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging.....	459
Parallel Simulation with Transmission Lines in Modelica	325
Robust Initialization of Differential Algebraic Equations	607
Types in the Modelica Language.....	303
Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel	193
Furic, S.	
Types in the Modelica Language.....	303
Gäfvert, M.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
NowaitTransit Concept Assessment. Modeling of Trains on Complex Track Geometry	225
The VehicleDynamics Library - Overview and Applications	43
Giuliani, H.	
Simulation of Hybrid Electric Vehicles.....	25
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
González, L.	
GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms	335
Gragger, J.V.	
Simulation of Hybrid Electric Vehicles.....	25
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
Gühmann, C.	
Synchronising a Modelica Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System.....	275
Guinéa, D.	
GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms	335
Haase, T.	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
Harman, P.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
Modelling Automotive Hydraulic Systems using the Modelica ActuationHydraulics Library.....	399
Haumer, A.	
Identification and Controls of Electrically Excited Synchronous Machines	589
If We Only had Used XML.....	707
Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation	261
Simulation of Components of a Thermal Power Plant	119
Heckmann, A.	
The DLR FlexibleBodies Library to Model Large Motions of Beams and of Flexible Bodies Exported from Finite Element Programs	85
Heinrich, C.	
A Modelica Library for Simulation of Household Refrigeration Appliances - Features and Experiences	677

Hirsch, T.	
Simulation of the Start-Up Procedure of a Parabolic Trough Collector Field with Direct Solar Steam Generation.....	135
Hoffmann, C.	
Dynamic Optimization of Energy Supply Systems with Modelica Models	599
Hoh, A.	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
Holm, A.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63
Huhn, M.	
Modelica CVD - A Tool for Visualizing the Structure of Modelica Libraries.....	55
Jagudin, E.	
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging.....	459
Janin, G.	
Development and Verification of a Series Car Modelica/Dymola Multi-body Model to Investigate Vehicle Dynamics Systems	167
Johansson, O.	
Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools	359
Kapeller, H.	
Identification and Controls of Electrically Excited Synchronous Machines	589
If We Only had Used XML.....	707
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
Kellner, M.	
Parameterisation of Modelica Models on PC and Real Time Platforms	267
Kenny, P.J.	
Modeling the Dynamics of Vehicle Fuel Systems.....	147
Kerkar, N.	
Pressurized Water Reactor Modelling with Modelica	127
Kittirungsri, B.	
UnitTesting: A Library for Modelica Unit Testing	695
Knobel, C.	
Development and Verification of a Series Car Modelica/Dymola Multi-body Model to Investigate Vehicle Dynamics Systems	167
Koehler, J.	
How to Dissolve Complex Dynamic Systems for Wanted Unknowns with Dymola / Modelica.....	187
Köhler, J.	
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Kosenko, I.I.	
Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation	213
Kossel, R.	
Modelica CVD - A Tool for Visualizing the Structure of Modelica Libraries.....	55
Simulation of Complex Systems using Modelica and Tool Coupling.....	485
Krabbes, M.	
Dynamic Modeling and Control of a 6 DOF Parallel Kinematics.....	385

Kral, C.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
Identification and Controls of Electrically Excited Synchronous Machines	589
If We Only had Used XML.....	707
Optimization of a Cooling Circuit with a Parameterized Water Pump Model	493
Simulation of Components of a Thermal Power Plant	119
Simulation of Hybrid Electric Vehicles.....	25
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
Lacher, H.	
Optimization of a Cooling Circuit with a Parameterized Water Pump Model	493
Lakner, M.	
Modeling and Simulation of Generator Circuit Breaker Performance	319
Larsson, J.	
A Modelica Based Format for Flexible Modelica Code Generation and Causal Model Transformations	467
Lemke, N.	
Simulation of Complex Systems using Modelica and Tool Coupling.....	485
Leopold, J.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63
Lie, B.	
A New Energy Building Simulation Library	685
Loeffler, M.	
Modelica CVD - A Tool for Visualizing the Structure of Modelica Libraries.....	55
Loginova, M.S.	
Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation	213
Lopez, J.D.	
Automatic Fixed-point Code Generation for Modelica using Dymola.....	621
Dymola interface to Java - A Case Study: Distributed Simulations	477
Shock Wave Modeling for Modelica.Fluid Library using Oscillation-free Logarithmic Reconstruction	641
The DLR FlexibleBodies Library to Model Large Motions of Beams and of Flexible Bodies Exported from Finite Element Programs	85
Lovera, M.	
A Modelica Library for Space Flight Dynamics.....	107
Magnani, G.	
Acausal Modelling of Helicopter Dynamics for Automatic Flight Control Applications	377
Makanaboyina, R.	
Integration of CATIA with Modelica	671
Martinez, F.	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method	97
Mathijssen, A.	
Modelling of a Solar Thermal Reactor for Hydrogen Generation	441
Mattes, A.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63

Matthes, P.	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
Mattsson, S.E.	
Calibration of Static Models using Dymola	615
Meissner, Ch.	
Dynamic Modeling and Control of a 6 DOF Parallel Kinematics.....	385
Müller, D.	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
Munteanu, M.	
On the Noise Modelling and Simulation	369
Murua, X.	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method.....	97
Najafi, M.	
Modeling and Simulation of Differential Equations in Scicos	177
Neumann, M.	
Parameterisation of Modelica Models on PC and Real Time Platforms	267
Nicolau, V.	
On the Noise Modelling and Simulation	369
Nikoukhah, R.	
Modeling and Simulation of Differential Equations in Scicos	177
Nordström, U.	
Automatic Fixed-point Code Generation for Modelica using Dymola.....	621
Nordwig, A.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63
Nouidui, T.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63
Nyström, K.	
Parallel Simulation with Transmission Lines in Modelica	325
Nytsch-Geusen, C.	
Advanced Modeling and Simulation Techniques in MOSILAB: A System Development Case Study	63
Obratsov, YA.P.	
Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation	213
Olsson, H.	
Calibration of Static Models using Dymola	615
Dymola interface to Java - A Case Study: Distributed Simulations	477
Otter, M.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
Modelling of an Experimental Batch Plant with Modelica	651
The DLR FlexibleBodies Library to Model Large Motions of Beams and of Flexible Bodies Exported from Finite Element Programs	85
The LinearSystems Library for Continuous and Discrete Control Systems.....	529
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks	631

Pagalday, J.M.	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method.....	97
Pascoli, G.	
Identification and Controls of Electrically Excited Synchronous Machines	589
If We Only had Used XML.....	707
Pentori, B.	
Pressurized Water Reactor Modelling with Modelica	127
Philipson, N.	
Leaf Spring Modeling.....	205
NowaitTransit Concept Assessment. Modeling of Trains on Complex Track Geometry	225
Pirker, F.	
Identification and Controls of Electrically Excited Synchronous Machines	589
If We Only had Used XML.....	707
Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation	261
The SmartElectricDrives Library - Powerful Models for Fast Simulations of Electric Drives	571
Plainer, M.	
Simulation of Components of a Thermal Power Plant	119
Pop, A.	
Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools	359
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging.....	459
Poschlad, K.	
Modelling of an Experimental Batch Plant with Modelica	651
Prat, V.S.	
ARENALib: A Modelica Library for Discrete-Event System Simulation	539
Pretolani, F.	
Fast Start-up of a Combined-Cycle Power Plant: A Simulation Study with Modelica	3
Proelss, K.	
Modeling of Frost Growth on Heat Exchanger Surfaces.....	509
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks	631
Pujana, A.	
3D Flexible Multibody Thin Beams Simulation in Modelica with the Finite Element Method.....	97
Pulecchi, T.	
A Modelica Library for Space Flight Dynamics.....	107
Putz, H.	
Dynamic Optimization of Energy Supply Systems with Modelica Models	599
Reisenbichler, U.	
If We Only had Used XML.....	707
Remar, A.	
OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging.....	459
Remelhe, M.A.P.	
Modelling of an Experimental Batch Plant with Modelica	651

Richter, C.C.	
Modelica CVD - A Tool for Visualizing the Structure of Modelica Libraries.....	55
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible	
Thermo-Fluid Pipe Networks	631
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Roeb, M.	
Modelling of a Solar Thermal Reactor for Hydrogen Generation.....	441
Rubio, M.A.	
GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms	335
Sandholm, A.	
A Numeric Library for Use in Modelica Simulations with Lapack, SuperLU, Interpolation,	
and MatrixIO	285
Sattler, C.	
Modelling of a Solar Thermal Reactor for Hydrogen Generation.....	441
Savaresi, S.M.	
Motorcycle Dynamics Library in Modelica.....	157
Schiavo, F.	
Motorcycle Dynamics Library in Modelica.....	157
Schimon, R.	
Simulation of Components of a Thermal Power Plant	119
Schlegel, C.	
Ascola: A Tool for Importing Dymola Code into Ascet.....	343
Schmidt, G.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Schmitz, G.	
Modeling of Frost Growth on Heat Exchanger Surfaces.....	509
Schneider, P.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Schwarz, P.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Schwunk, S.	
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Selimovic, F.	
Modeling and Dynamic Analysis of CO ₂ -Emission Free Power Processes in Modelica	
using the CombiPlant Library.....	17
Simic, D.	
Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation	261
Optimization of a Cooling Circuit with a Parameterized Water Pump Model	493
Simulation of Hybrid Electric Vehicles.....	25
Simulation of Components of a Thermal Power Plant	119
Sirbu, G.	
On the Noise Modelling and Simulation	369

Slättke, O.	
Modeling, Calibration and Control of a Paper Machine Dryer Section.....	411
Slättorp, K.	
HydroPlant – a Modelica Library for Dynamic Simulation of Hydro Power Plants.....	251
Souyri, A.	
Pressurized Water Reactor Modelling with Modelica.....	127
Stavrovskaya, M.S.	
Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation	213
Steinmann, W.D.	
Analysis of Steam Storage Systems using Modelica.....	235
Sundén, B.	
Modeling and Dynamic Analysis of CO ₂ -Emission Free Power Processes in Modelica using the CombiPlant Library.....	17
Suyam Welakwe, N.	
Integration of CATIA with Modelica	671
Tamme, R.	
Analysis of Steam Storage Systems using Modelica.....	235
Tanelli, M.	
Motorcycle Dynamics Library in Modelica.....	157
Tegethoff, W.	
Simulation of Complex Systems using Modelica and Tool Coupling.....	485
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Tiller, M.M.	
UnitTesting: A Library for Modelica Unit Testing	695
Tischendorf, C.	
Using Modelica as a Design Tool for an Ejector Test Bench.....	501
Treffinger, P.	
Coordinated Automotive Libraries for Vehicle System Modelling.....	33
Vehicle Model for Transient Simulation of a Waste-Heat-Utilisation-Unit Containing Extended PowerTrain and Fluid Library Components	405
Tschirner, T.	
Coupled Simulation of Building Structure and Building Services Installations with Modelica.....	717
Tummescheit, H.	
The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks	631
Tuszynski, J.	
HydroPlant – a Modelica Library for Dynamic Simulation of Hydro Power Plants.....	251
NowaitTransit Concept Assessment. Modeling of Trains on Complex Track Geometry	225
Tuszynski, K.	
HydroPlant – a Modelica Library for Dynamic Simulation of Hydro Power Plants.....	251
Ungethüm, J.	
Modelling of Alternative Propulsion Concepts of Railway Vehicles.....	391
Vehicle Model for Transient Simulation of a Waste-Heat-Utilisation-Unit Containing Extended PowerTrain and Fluid Library Components	405

Urquia, A.	
ARENALib: A Modelica Library for Discrete-Event System Simulation	539
GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms	335
Vanrolleghem, P.A.	
Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel	193
Vetter, M.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Quasi-stationary AC Analysis Using Phasor Description With Modelica	579
Videla, J.I.	
A New Energy Building Simulation Library	685
Viganò, L.	
Acausal Modelling of Helicopter Dynamics for Automatic Flight Control Applications	377
Wang, G.B.	
An Analyzer for Declarative Equation Based Models.....	349
Wetter, M.	
Multizone Airflow Model in Modelica.....	431
Multizone Building Model for Thermal Building Simulation in Modelica.....	517
Winkler, D.	
Synchronising a Modelica Real-Time Simulation Model with a Highly Dynamic	
Engine Test-Bench System.....	275
Wischhusen, S.	
An Enhanced Discretisation Method for Storage Tank Models within Energy Systems	243
Wittwer, C.	
Advanced Modeling and Simulation Techniques in MOSILAB:	
A System Development Case Study	63
Woodruff, A.	
Development and Verification of a Series Car Modelica/Dymola Multi-body Model to Investigate	
Vehicle Dynamics Systems	167
Wu, Y.-Z.	
An Analyzer for Declarative Equation Based Models.....	349
Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of	
Multi-domain Physical Systems Based on Modelica.....	733
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems	
Based on Modelica	725
Yebra, L.J.	
Object Oriented Modelling of DISS Solar Thermal Power Plant.....	449
Zarza, E.	
Object Oriented Modelling of DISS Solar Thermal Power Plant.....	449
Zhang, Y.-Q.	
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain	
Physical Systems Based on Modelica.....	725
Zhao, J.-J.	
Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of	
Multi-domain Physical Systems Based on Modelica.....	733
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems	
Based on Modelica	725

Zhou, F.-L.	
An Analyzer for Declarative Equation Based Models.....	349
Domain Library Preprocessing in MWorks - A Platform for Modeling and Simulation of	
Multi-domain Physical Systems Based on Modelica.....	733
MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems	
Based on Modelica	725
Zimmer, D.	
The Modelica Multi-bond Graph Library	559

Session 4

Poster Session

GAPILib - A Modelica Library for Model Parameter Identification Using Genetic Algorithms

Miguel A. Rubio⁺, Alfonso Urquia*, Leandro Gonzalez⁺, 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, ETS 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 *GAPILib* is discussed in this manuscript. *GAPILib* is a new Modelica library for parameter identification in Modelica models, using genetic algorithms (GA). *GAPILib* can be used for parameter estimation in any Modelica model and the estimation process does not require to perform model modifications. This new library supports simple- and multi-objective optimization. *GAPILib* library is composed of a set of functions that can be easily used, modified and extended. The use of *GAPILib* is illustrated by means of a case study: the estimation of electrochemical parameters in fuel cell models, which have been composed by using *FuelCellLib* library. *GAPILib* is completely written in Modelica language and it will be freely available soon.

Keywords: Genetic Algorithm, Fuel cell, parameter identification

1 Introduction

Frequently, the modelling process includes the estimation of model parameters from experimental data. The use of genetic algorithms (GA) to perform this task is broadly accepted.

GA are numerical optimisation algorithms inspired by the natural selection processes that take place among the live beings. Individuals evolve through adaptation to their external environment. Most capable individuals pass on their genetic information to later generations. As a consequence, the population evolution after a number of generations allows to obtain optimum results.

The use of GA for parameter estimation in Modelica models has been previously proposed by Hongesombut et al. [1]. However, these authors programmed and ran the GA using Matlab/Simulink. As a consequence, these authors' approach requires the combined use of Modelica/Dymola and Matlab/Simulink.

The lack of a freely-available Modelica library implementing GA, suited for parameter estimation in Modelica models, has motivated the implementation of *GAPILib* library.

The fundamentals of the GA supported by *GAPILib* library are briefly explained in Section 2 and the library structure is discussed in Section 3. Finally, the use of *GAPILib* is illustrated by means of a case study: the estimation of electrochemical parameters in fuel cell models. This case study is described in Section 4.

2 GA supported by GAPILib

The GA supported by *GAPILib* library is schematically represented in Figure 1. The algorithm consists of the steps described next.

The GA starts with an initial population, which is randomly selected from the search space. Each individual of the population is formed by a group of chromosomes, which represents a solution to the problem.

This initial population is evaluated by using a cost function. This function is used to calculate the validity of the population members, which are ordered according to this criterion. The most valid member is selected for the crossover process, which generates a new population. This new population is evaluated and recombined to obtain

a new generation, and so on. These steps of the algorithm are repeated until the stop condition is satisfied (see Figure 1).

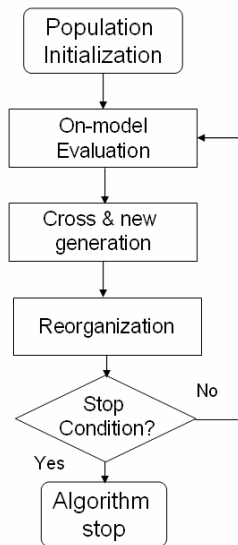


Figure 1: GA supported by *GAPILib*

The GA supported by *GAPILib* includes several processes intended to improve the algorithm performance, such as:

1. *Elitism*. Most valid individuals are passed on to the next generation without being altered by genetic operators. Using elitism ensures that the best solution is never lost from one generation to the next.
2. *Mutation*. It is a genetic operator that introduces random changes on the individuals, maintaining genetic diversity from one generation of the population of chromosomes to the next. The purpose of mutation is to allow the algorithm to avoid local minima.

A candidate problem solution is a value selection of the parameters under estimation. These parameter values are included in the model and this is simulated. A “fitness function” is applied to the obtained model response in order to evaluate the candidate solution. The next candidate solution is obtained by applying different genetic operators. The new values of the parameters are included in the model, which is simulated, and so on.

3 *GAPILib* structure

GAPILib has been programmed by combining the use of the scripting Modelica language (see Figure 2a) and of functions written in Modelica language

(see Figure 2b). The overall library structure is schematically represented in Figure 2.

The GA execution starts by running the script file *GAPILib.mos*. This file only contains the sentences required to execute the following two script files: *GAPILib_INI.mos* and *GAPILib_CYCLE* (see Figure 2a). The purpose of these files can be summarized as follows:

- The script file *GAPILib_INI.mos* carries out the initialization of the *GAPILib* parameters and generates the initial population.
- The script file *GAPILib_CYCLE* performs the operations required to obtain the next generations. Its execution finishes when the stop condition is satisfied. The stop condition shown in Figure 2a is of the type: “N_Cycle generations have been obtained”. Other stop conditions are possible, e.g., “the calculated fitness value is smaller than a given value”.

Further details about these two script files are provided next.

3.1 Script file *GAPILib_INI.mos*

The script file *GAPILib_INI.mos* contains the required function calls to perform the following tasks (see Figure 2a):

1. To define the directory path and the name of the Modelica models. This is the only place where the user has to provide this information.
2. To set the value of the GA parameters, which are listed in Table 1.

Table 1: *GAPILib*’s GA parameters

N_Population	Number of individuals in the population.
N_Parameters	Number of parameters to identify in the model (i.e., in <i>Model.mo</i>).
N_Parents	Number of parents of the population selected to cross.
N_Elitism	Number of elite individuals. If N_Elitism=0, then the Elitism function is no applied.
N_Cross_Point	Number of crossing points.
N_Cycle	Number of generations calculated. This value sets the stop condition.
F_Mut	Probability of random modification of an individual due to mutation.

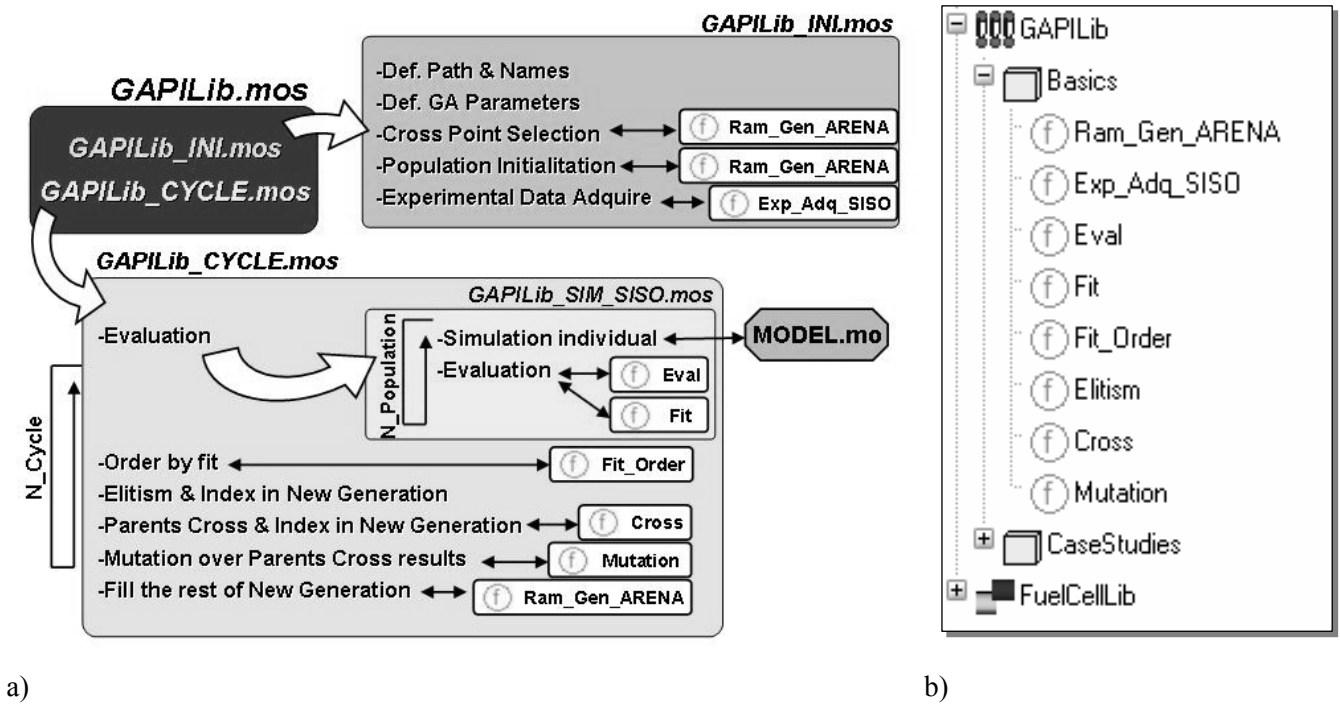


Figure 2: Schematic representation of GAPILib library architecture

- To carry out the random selection of the crossing points used for the crossover process. The function **Ram_Gen_arena** is used to generate the pseudo-random numbers. This function implements the pseudo-random number generator used by Arena 7.0 simulation environment.
- To select the initial population, which is composed of random elements. Again, the **Ram_Gen_arena** function is used for pseudo-random number generation. The user is allowed to select the range of each parameter under estimation. This capability allows to reduce the search space.
- To read the experimental values used as a reference to fit the model. The **Exp_Adq_SISO** function is used to perform this task.

3.2 Script file GAPILib_CYCLE.mos

The script file **GAPILib_CYCLE.mos** contains the required function calls to perform the following tasks (see Figure 2a):

- To execute the script file **GAPILib_SIM_SISO**, that performs the simulation of the model **Model.mo** with the parameter values corresponding to each of the individuals of the population. The model is simulated as many times as individuals are in the population. The simulation results are stored and compared with the experimental

data. The **Eval** and **Fit** functions are used. All the population individuals are evaluated.

- To sort the population individuals according to the fitness values previously calculated. The **Fit_Order** function is used.
- To pass on the elite individuals to the next generation. These individuals are not altered by crossover and mutation.

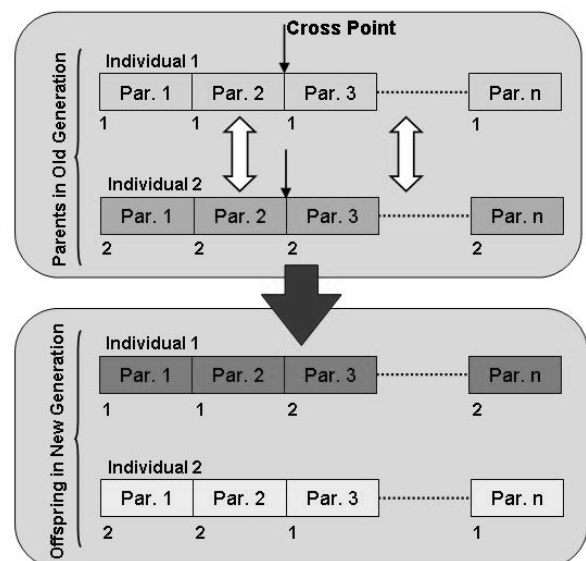


Figure 3: New generation obtained by crossover

Table 2: Input and output variables of *GAPILib* functions

Function name	Input variables	Output variables
Ram_Gen_ARENA	<ul style="list-style-type: none"> Seed [1, 6]: seed of ARENA algorithm N_Ram: number of pseudo-random numbers to generate 	<ul style="list-style-type: none"> Ram_list:[1,N_Ram]: Array of N_Ram pseudo-random numbers
Exp_Adq_SISO	<ul style="list-style-type: none"> namefile: name of .mat file where the experimental data is stored Xmatrixname: name of X variable matrix Ymatrixname: name of Y variable matrix 	<ul style="list-style-type: none"> State_Exp_Adq: state of experimental data SizeMatrix [2]: size of the Xmatrixname and Ymatrixname matrices Xexp: array of experimental X data Yexp: array of experimental Y data
Eval	<ul style="list-style-type: none"> SimuPath: path of the model Model.mo SimuCaseStudyName: name of the model to simulate Xexp: Array of experimental X data New_Generation: complete set of population values 	<ul style="list-style-type: none"> DATASim_Int: interpolated values with Xexp of model simulate result
Fit	<ul style="list-style-type: none"> DATASim_Int: interpolated values with Xexp of model simulate result Yexp: array of experimental Y data 	<ul style="list-style-type: none"> Eval_Mod: fitness evaluation result
Fit_Order	<ul style="list-style-type: none"> Pop: population to be ordered Fit: fitness of all population Population: number of population individuals Parents: number of population parents Elitism: number of elitism individual 	<ul style="list-style-type: none"> Pop_Ordered: population sorted by fitness value
Cross	<ul style="list-style-type: none"> Parents: number of population parents Elitism: number of elitism individual. OldGeneration: old generation of population ordered by fitness 	<ul style="list-style-type: none"> NewGeneration_Cross: population obtained from elitism and crossover
Mutation	<ul style="list-style-type: none"> Nparameters: complete population parameters of NewGeneration_Cross F_Mut: mutation factor RamMut: pseudo-random number generated using Ram_Gen_ARENA. This number is used to decide whether a parameter is mutated and its value 	<ul style="list-style-type: none"> NewGeneration_Mut: population obtained after elitism, crossover and mutation

- To cross the selected parents (the most capable individuals), using the crossing point calculated from `GAPILib_INI`. The `Cross` function is used. The algorithm implemented is shown in Figure 3.
- To apply the `Mutation` function. The mutation factor is the probability used to mutate any parameter of an individual.
- The new population is completed with random elements. The `Ram_Gen_ARENA` function is used.

The input and output variables of *GAPILib* functions are shown in Table 2. These functions are stored within the `GAPILib.Basics` package (see Figure 2b).

4 Case study

GAPILib has been successfully applied to the estimation of electrochemical parameters in fuel cell models composed by using *FuelCellLib*. The packages and models of *FuelCellLib* are shown in Figure 4. Further information about this free Modelica library can be found in [2].

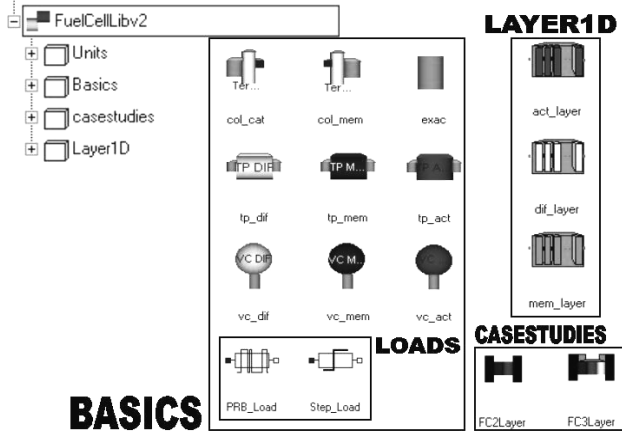


Figure 4: Packages and models of *FuelCellLib*

The obtained models can be used to simulate the steady-state and the dynamic behavior [3,4,5] of the fuel cells along their complete range of operation. For instance:

- The experimental and simulated polarization curve (I-V) of a fuel cell is shown in Figure 5.
- The experimental and simulated fuel cell voltage, obtained in response to step changes in the load, is shown in Figure 6a.
- The simulated vs. experimental data of the water long-term effect is shown in Figure 6b. The simulation reproduces: (1) the slow voltage rise due to the membrane hydrate; and (2) the voltage fall due to the water flooding of the cathode.

Next, the these three fitness processes are discussed.

The experimental data used in this work have been obtained in the laboratory of renewable energy of the IAI, CSIC.

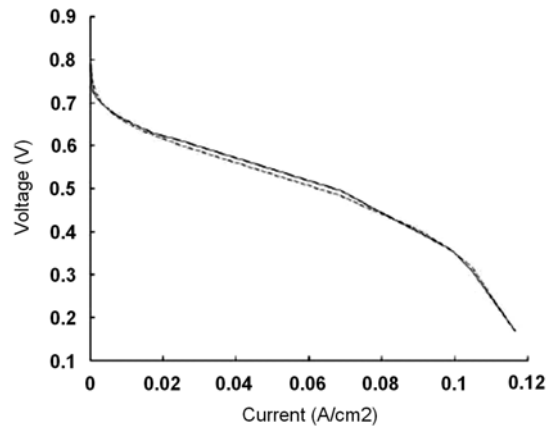


Figure 5: Fuel-cell polarization curve: (o) experimental; (--) simulated using *FuelCellLib*

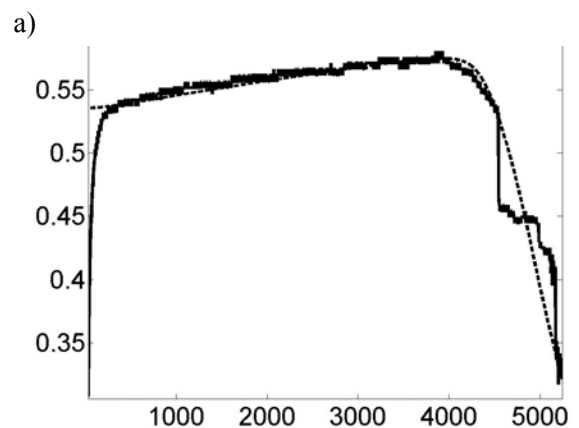
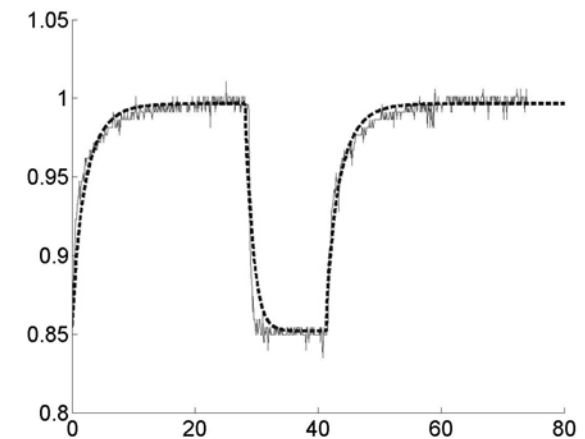


Figure 6: Experimental (o) and simulated using *FuelCellLib* (--). Time [s], X axis. Voltage [V], Y axis. a) Fuel cell voltage in response to step changes in the load. b) Long-term effect of the water, with constant load.

4.1 Fitness of the polarization curve

In order to obtain the polarization curve, the model has to be simulated, for each of the operation points composing the curve, until the steady-state is reached.

The parameters identified to fit the polarization curve are shown in Table 3. In all experiments, one crossing point was used. The fitness function is defined as the sum of the quadratic differences between the experimental and the simulated values of the variable. I.e.:

$$\text{Fit} = \sum (Y_i - \hat{Y}_i)^2 \quad (1)$$

The GA parameters are set to the following values:

- The stop condition is satisfied after 5000 generations.
- Each population was composed of 100 individuals.
- The mutation factor was 0.25.
- 70 parents and one elite element were used in each generation.

The values of a parameter obtained with best fitness value are shown in the Table 4.

Table 3: Parameters to estimate

Parameter	
A	Tafel slope
I_n	Inner current density
I_o	Exchange current density
B	Mass transfer slope
R	Inner area specific resistance
I_{lim}	Limiting transport current density

Table 4: Result of the fitness

Parameter	Value
A (V)	0.0390
I_n (A cm ⁻²)	1.4×10^{-3}
I_o (A cm ⁻²)	1.5856×10^{-6}
B (V)	0.0918
R (Ω cm ²)	7.2860×10^{-4}
I_{lim} (A cm ⁻²)	0.2265

4.2 Fitness of the fuel cell voltage in response to step changes in the load

It is this case, *GAPLib* is used to estimate the values of the four parameters shown in Table 5. The GA parameters are set to the following values:

- The stop condition is satisfied after 200 generations.
- 150 individuals were used in each population.
- A factor of mutation of 0.25 was applied.
- 100 parents were used in each generation.
- One elite element was used.

The obtained values of the parameters are shown in the Table 6.

Table 5: Parameters to estimate

Parameter	
R_{inf}	Down resistance value
R_{sup}	High resistance value
C_{dl}	Doble layer capacitance
k_s	Electrical conductivity of the solid

Table 6: Results of the fitness

Parameter	Value
R_{inf} (Ω m ⁻²)	0.03315
R_{sup} (Ω m ⁻²)	5.1
C_{dl} (F m ⁻²)	10.12
k_s (S m ⁻¹)	0.01

4.3 Fitness of the long term effect of water on the fuel cell voltage with constant resistance load

To carry out this experiment, the fuel cell model was modified in order to reproduce the variation of the membrane conductivity. The parameters used to fit the model are shown in Table 7.

The GA parameters are set to the following values:

- The stop condition is satisfied after 700 generations.
- 70 individuals were used in each population.
- A factor of mutation of 0.15 was applied.

- 50 parents were used in each generation.
- One elite element was used.

The obtained values of the parameters are shown in Table 8.

Table 7: Parameters to estimate

Parameter	
da(Act_Layer)	Width of active layer
X_s	Percentage of pore volume of solid
D_{12}	Binary diffusion coefficient
d _a (Mem_Layer)	Width of membrane layer
R_{mem}	Resistance of membrane layer

Table 8: Result of the fitness

Parameter	Value
da(Act_Layer) (m)	6×10^{-8}
X_s	0.05
D_{12} (m ² /s)	5.01×10^{-9}
d _a (Mem_Layer) (m)	1.6×10^{-5}
R_{mem} (Ω m ²)	1.419×10^{-3}

which have been composed by using *FuelCellLib* library.

References

- [1] Hongesombut K, Mitani Y, Tsuji K. An Incorporated Use of Genetic Algorithm and a Modelica Library for Simultaneous Tuning of Power System Stabilizers. In: Proceedings of the 2nd International Modelica Conference, 2002, pp. 89-98.
- [2] Rubio M.A, Urquia A, Gonzalez L, Guinea D, Dormido S. *FuelCellLib*- A Modelica Library for Modeling of Fuel Cells. In: Proceedings of the 4th International Modelica Conference, 2005, pp. 75-82.
- [3] Larminie J, Dicks A. Fuel Cell Systems Explained, Wiley 2000.
- [4] Bevers D, Wöhr M, Yasuda K, Oguro K. Simulation of polymer electrolyte fuel cell electrode. J. Appl. Electrochem. 27 (1997).
- [5] Broka K, Ekdunge P. Modelling the PEM fuel cell cathode, J. Appl. Electrochem. 27 (1997).

5 Future work

Next version of *GAPILib* will support the solution of multi-objective problems. Also, it will include additional fitness functions and the user will be allowed to select among them the fitness function best suited to each particular problem.

In addition, a complete guide of *GAPILib* use will be developed.

6 Conclusions

The design, implementation and use of *GAPILib* has been discussed. *GAPILib* library is an effective tool for parameter identification in Modelica models using GA. It is completely written in Modelica language, which facilitates its use, modification and extension. *GAPILib* can be used for parameter identification in any Modelica model and the estimation process does not require to perform model modifications. *GAPILib* has been successfully applied to the estimation of electrochemical parameters in fuel cell models,

Ascola: A Tool for Importing Dymola Code into Ascet

Clemens Schlegel
Schlegel Simulation GmbH
Meichelbeckstr. 8b
D-85356 Freising
cs@schlegel-simulation.de

Reinhard Finsterwalder
University of the Federal
Armed Forces Munich
D-85577 Neubiberg
reinhard.fensterwalder@unibw.de

Abstract

A new tool, Ascola™, is presented for import of Dymola generated C-code of a Modelica simulation model into Ascet-MD. All variable names, types, default values, units and comments of the original Modelica model along with the naming structure are retained during import.

The main focus of Ascola is on delivering a user-friendly tool to aid embedded control system design using Ascet and Modelica. An example will be discussed to illustrate the import process and the use of imported models in Ascet.

1. Motivation

The tools of the Ascet product family [1] support the development of embedded software by model-based design of control and diagnostic functions, simulation, rapid prototyping, and automatic code generation. Ascet-MD provides functions to support the modeling and (offline-) simulation tasks in a structured, object oriented way. Functions like the visualization of the interdependencies in a model or the possibility to edit all implementation details of a variable help the user to focus on the main algorithms. Ascet tools are widely used in the automotive sector.

To check whether the software and the control algorithms under development fulfill all functional requirements of a design project is a core task in all stages of the development process. In the first stages this is mostly done by testing the control algorithms and the control software against a simulation model of the device to be controlled (model-in-the-loop and software-in-the-loop). In later stages hardware-in-

the-loop simulations come in. For the Ascet environment Ascet-MIP (Matlab Integration Package) [1] facilitates the import of Simulink [2] models to perform such functional tests. It requires the component Realtime Workshop [2] to generate C-code of the Simulink model, which is then imported in Ascet.

Due to its unique features (most outstanding the object oriented, acausal approach) Modelica [3] is used more and more as a modeling language for mechatronic devices which are typically controlled by an embedded controller. Naturally, developers of embedded control software wish to use also Modelica models in Ascet for functional testing. Using Dymola [4] for processing a Modelica model this can be done with some tweaking via export of the Modelica / Dymola model to Simulink and import of the resulting S-function block in Ascet via MIP [5]. However, this requires a long chain of tools, what makes the development process less robust and more expensive. In this paper we introduce Ascola as a tool for direct import of Dymola generated C-code of a Modelica model into Ascet.

2. Basic Structure of Ascet

A controller is specified (modeled) in Ascet by a set of interdependent elements, modules, classes, processes, tasks, and messages. The algorithms and the configuration of the operating system used are maintained in a project. Projects consist of modules and tasks. Modules contain several processes which are specified within the module and scheduled for execution in the tasks. Modules encapsulate several class instances as objects. Each instance belongs to a single class which contains the related methods. Each module occurs only once in a project, whereas classes may have several instances. Methods differ

from processes by containing arguments and return values. Inter-process communication is done using messages. Messages from different modules are globally available, they are automatically routed according to matching names. All elements are kept in a database to facilitate consistent handling and reuse.

Both classes and modules may be specified graphically using a block diagram editor or textually using the built-in language ESDL (Embedded Software Description Language), or using ANSI-C. All elements of Ascet contain implementation information in order to handle target- or project-specific variants. The implementation consists of the respective data types, default values, ranges and quantization of values and memory-location information (RAM, ROM, flash, etc.). Therefore C-classes and C-modules are always implementation specific and treated accordingly in Ascet. The implementation information is used for the automatic code generation. It's indispensable especially for generating code for specialized embedded targets with fixed-point arithmetic.

An important part of the controller specification is the interface definition of the controller's classes and modules. The interface of a class consists of its public methods, their arguments and return values. The interface of a module consists of its processes. Processes determine the activation of the module's functionalities, but unlike C-classes they do not define inputs, outputs or parameters. Modules communicate and interact via messages and global elements which have to be specified explicitly.

In Ascet's project editor the different processes, tasks and the runtime environment (the built in real-time operating system Ercosek) can be configured. There is also an experiment environment to run, stimulate, calibrate and monitor the generated controller code. The control algorithm may run offline on a PC, in real-time on a dedicated experiment hardware and on several target processors used in today's automotive control units.

3. Ascola

3.1 Overview

According to the described structure of Ascet an imported C-code simulation model matches the properties of a C-module inserted in a project (not in a library), because it is unique, has an encapsulated

functionality, is based on full scale floating point arithmetic (what's a target specific implementation detail from Ascet's point of view), and needs a specific task structure and a specific sampling rate, which are defined in a project, not in a module.

Started in an Ascet project editor Ascola first reads in the model specific C code generated by Dymola (dsmodel.c), parses the variable definitions (input, output, parameters, and initial states), and prepares corresponding elements according to the Ascet conventions (data type, unit, comment, etc.). It then displays the basic model properties (number and type of parameters, etc.), and asks the user for model and parameter-update sampling rates and the integration algorithm to choose (if not included in the model [6]).

From this information an Ascet C-module with body, header, and interface definition is automatically generated and inserted in the actual project. In addition the necessary make information (compiler settings, symbol definition, linking of model independent utilities, etc.) is generated. The C-module is organized in four processes (initialization, simulation, parameter update and termination) which are assigned to four dedicated tasks. Now the imported Dymola model code can be used like any other module in an Ascet experiment environment.

After having generated C-code from a Modelica model Dymola is not needed any more neither for the import procedure nor at runtime of the corresponding Ascet module, what's a major advantage compared with cosimulation approaches. More over, Dymola generated code can easily be passed to an Ascet user without disclosure of model details because that code is very hard to read. If the Dymola code is used in an offline experiment in Ascet even a variable step integrator including event detection may be used in that module, avoiding the necessity to tune the simulation model for fixed step integration, what may be a major task if the model is stiff, contains nonlinearities, state events, and / or causality changes (e.g. stick slip, hard stop, etc). This technique of encapsulating the integration algorithm along with the simulation model (as a separate or an inlined algorithm) is beneficial also in other simulation environments like Simulink (see e.g. [7]).

3.2 Mapping of Modelica Models to Ascet

Mapping of Modelica model properties to Ascet comprises mainly two aspects: handling of data types and plugging into the operating system environment.

Modelica's predefined data types (real, integer, boolean) containing values, units and comments fit quite well to according Ascet's data types (Ascet is limited to scalars and arrays in one and two dimensions). Therefore Ascola preserves the Modelica names, units and comments. Ascet's tabulated data types (1D and 2D, several interpolation methods) containing axis and dependent values have no corresponding data type built in Modelica (version 2.2.1, [3]). A user defined Modelica data type would not help, because it's structure can't be traced down in the C-code generated by Dymola.

Input and output are implemented as messages in Ascet. Modelica variables with fixed causality are mapped correspondingly. Because Ascet messages are limited to scalars, fixed causality arrays in Modelica have to be serialized.

Since Ascet does not provide integrators which can be used for C-modules they have to be added by Ascola, if not already included in the model (inline integration [4]). Despite the fixed I/O schedule (performed at a user defined rate) a variable step solver may be used as mentioned in paragraph 3.1. In the current version Ascola provides Euler and DASSL (stiff solver, variable step [8]) as integrators.

Apart from initial and terminal sections Modelica doesn't provide any keyword influencing the execution sequence of the generated code. Therefore there are code parts executed once and other parts executed in a fixed schedule imposed by the controller under test. This scheduling requirements end up with the following task scheme:

- initial task: model initialization (single shot)
- step task: input, output, model evaluation (high sampling rate)
- parameter update (low sampling rate)
- terminal task: model terminal section (single shot)

3.3 Implementation Aspects

Ascola is implemented in C++. In the current version only the PC target is supported which is based on the Borland C++-compiler.

The complete model specific Dymola code is stored in the Ascet database in order to avoid consistency problems. Ascola supports several Dymola versions (6.0 and higher) by providing version-specific header files and corresponding function libraries.

4 Example

Figure 1 shows a simple vehicle model which is suitable for checking e.g. cruise control algorithms. Ascola is invoked from an Ascet project editor (menu Dymola / Import C-code, see figure 2). After having chosen the file to import Ascola shows the main model properties (number and type of parameters, inputs, and outputs) and import defaults (task names and task timings for model code and parameter update) which may be overwritten by the user (figure 3). The different steps of the import procedure are logged in Ascet's monitor window (figure 4). After the code import Ascet's main window shows the interface elements of the generated C-code module: names of the Modelica variables (with prefixes in_, out_, and p_ for inputs, outputs and parameters) and the corresponding Ascet properties (type, scope, kind, dependency, etc.), see figure 5. In the OS tab of the project editor the generated task configuration settings are shown (figure 6). From the project editor a simulation experiment may be invoked, figure 7 shows simulation results.

5 Future

For a future version it's intended to support also Ascet's rapid prototyping targets (based on PowerPC [1]). This will facilitate model based control and realtime simulations using Modelica models. Based on user demand it's further planned to make Dymola generated C-code import also available for Intecrio, Etas' universal simulation tool [1].

6 References

- [1] www.etasgroup.com
- [2] www.mathworks.com
- [3] www.modelica.org
- [4] www.dynasim.se
- [5] C. Schlegel, Kopplung Dymola – Ascet, Technical report for BMW AG. Munich, 2003
- [6] Dymola 5.3 User's Manual, Dynasim AB, Lund, Sweden. 2004
- [7] J. Köhler, A. Banerjee, Usage of Modelica for Transmission simulation in ZF. Proceedings Modelica 2005, pp. 587-592
- [8] Brown, Hindmarsh, Petzold, BDF methods with direct and preconditioned Krylov linear solvers. SIAM J. Sci. Comp.: 15, 6, 1467 (1994) and 19, 5, 1495 (1998)

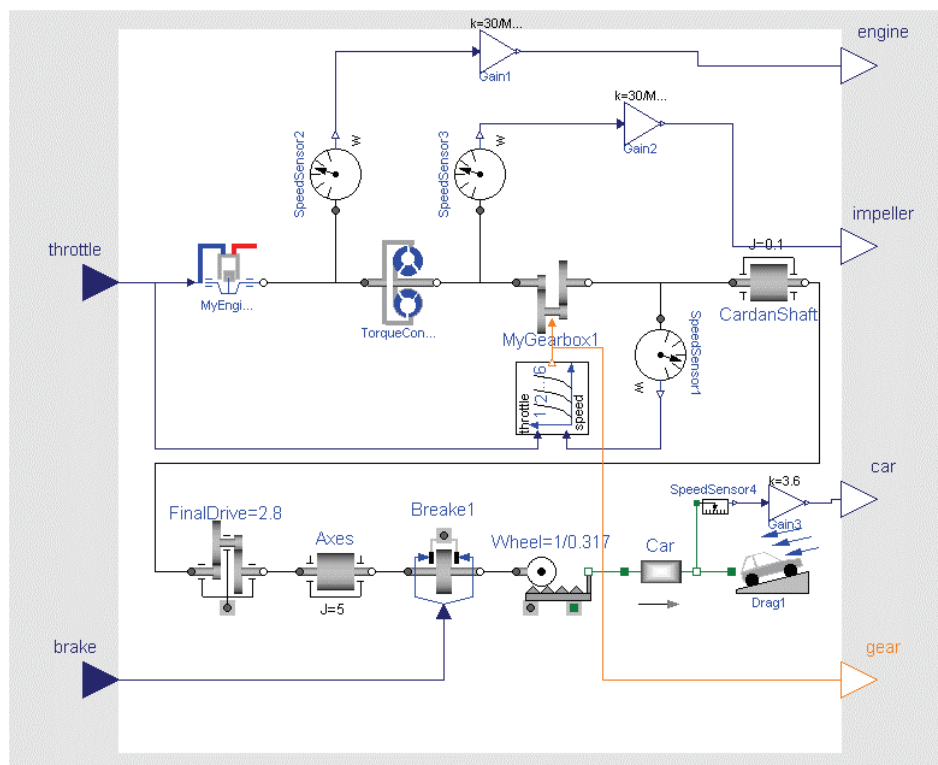


Figure 1: Simple vehicle model for checking cruise control algorithms

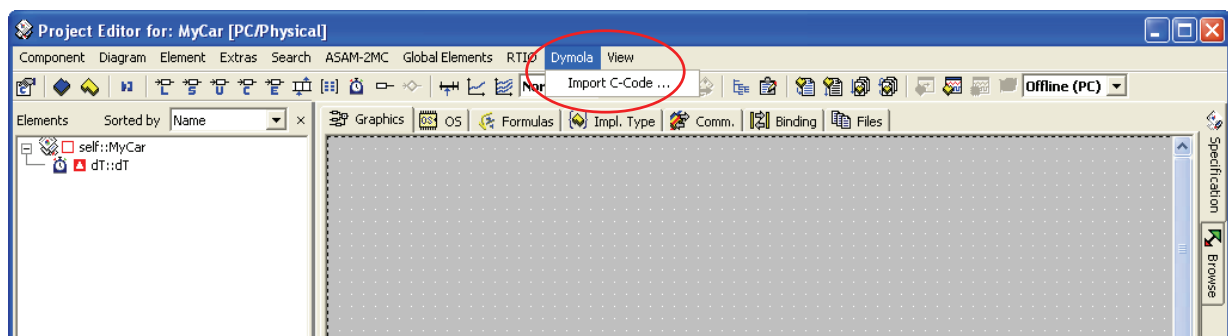


Figure 2: Invoking Ascola from an Ascet project editor

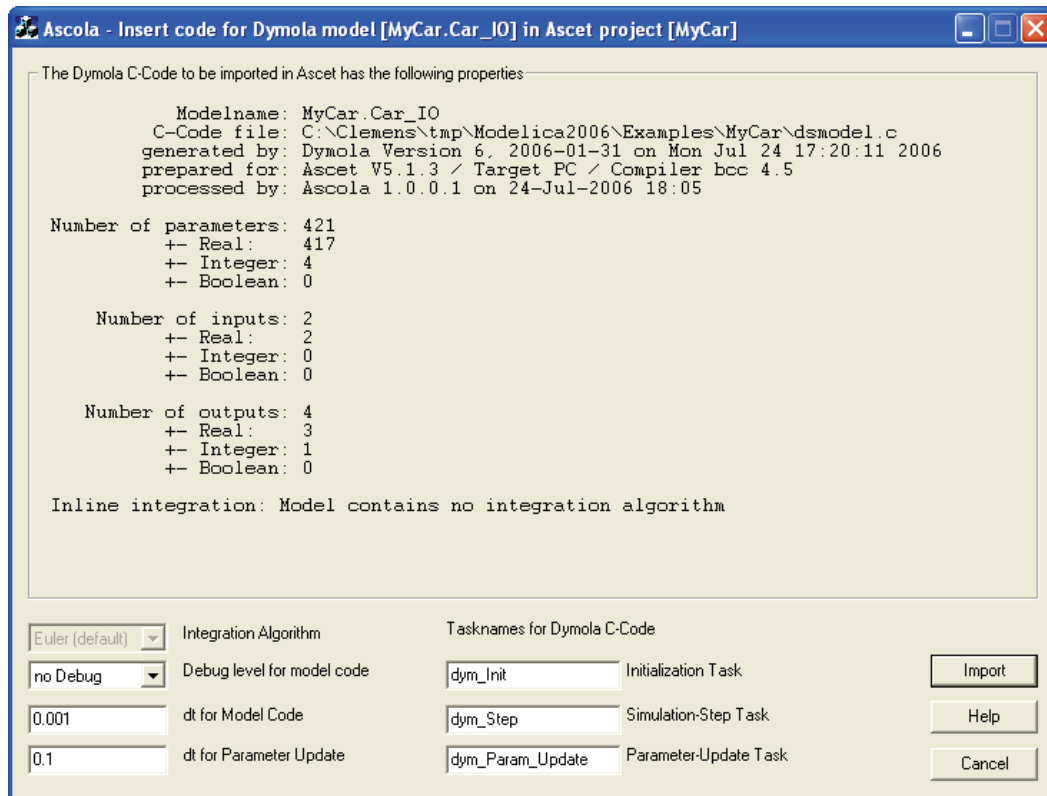


Figure 3: Graphical user interface of Ascola

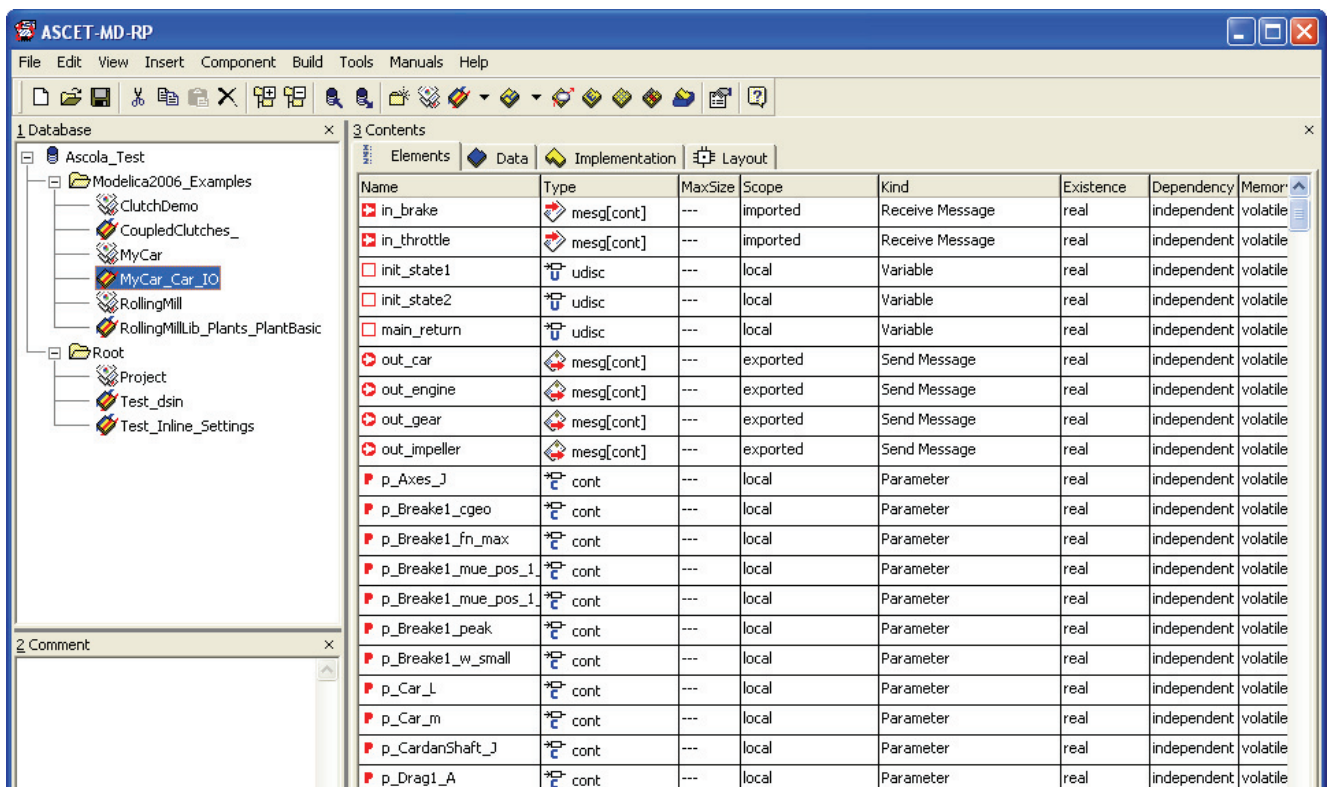


Figure 5: Ascet main window after model import

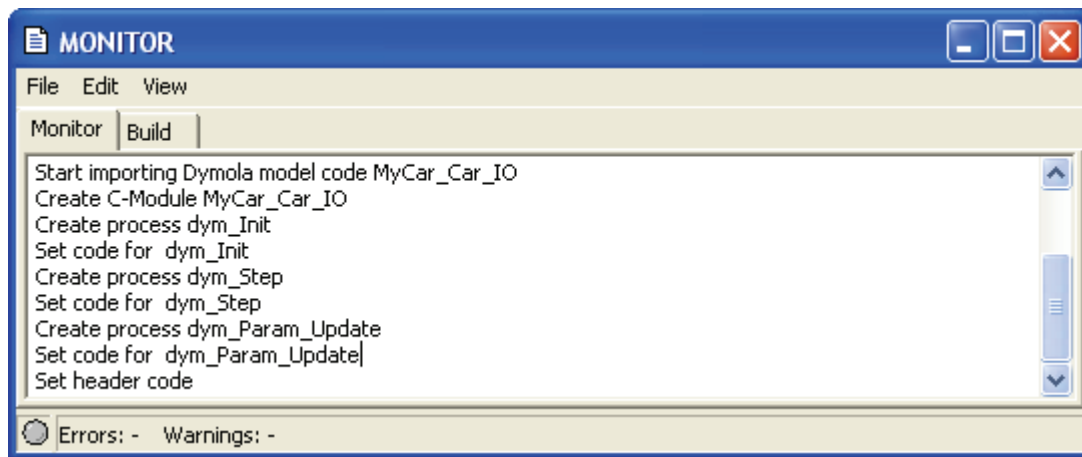


Figure 4: Ascola import messages in Ascet monitor

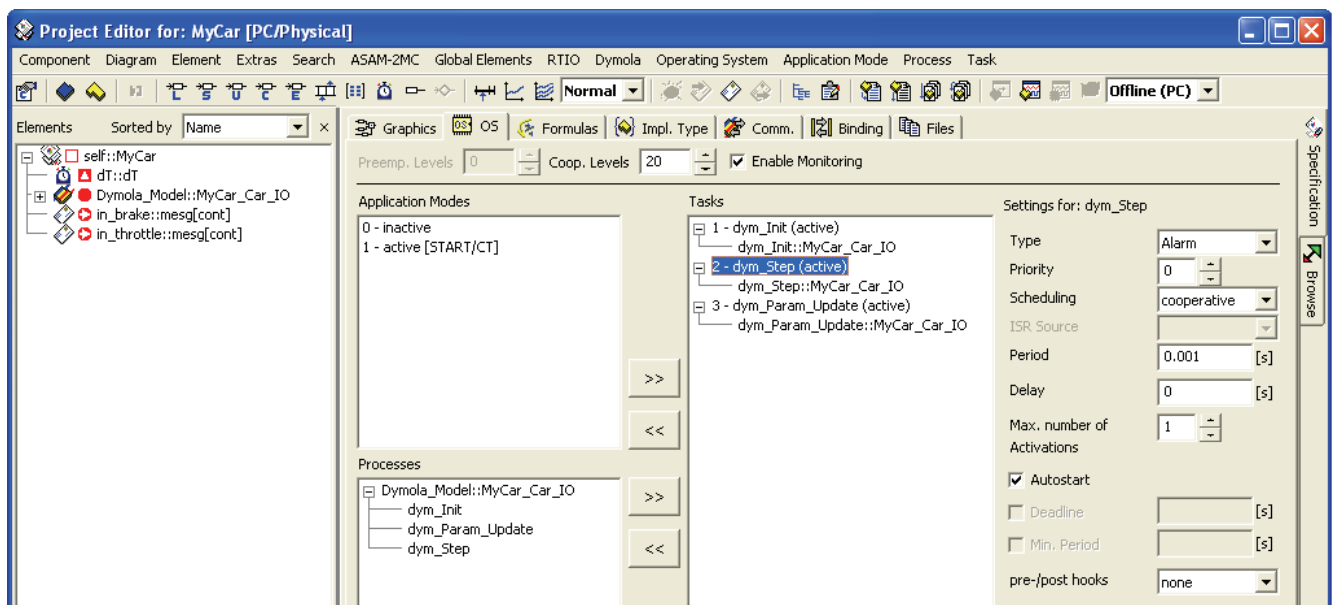


Figure 6: Operating system configuration for imported model

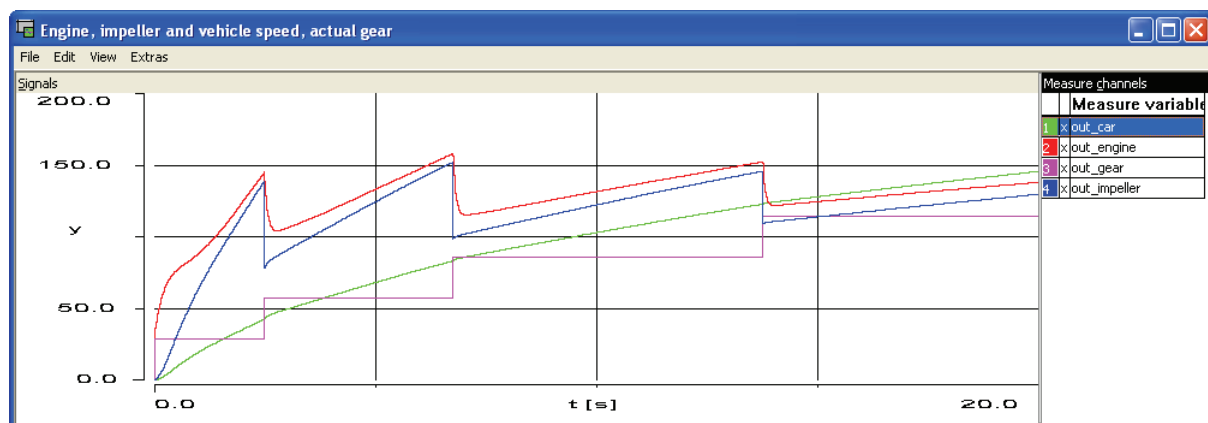


Figure 7: Vehicle acceleration simulation in Ascet (upper: engine and impeller speed, lower: vehicle speed and actual gear)

An Analyzer for Declarative Equation Based Models

JIAN-WAN Ding¹ LI-PING Chen¹ FAN-LI Zhou¹ YI-ZHONG Wu¹ GUO-BIAO Wang²

¹National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, China

²National Natural Science Foundation of China, Beijing 100085, China

Abstract

Along with its benefits, object-oriented modeling with Modelica language also brings the risk of missing or redundant equations, thus leads to an under-constrained problem or an over-constrained problem. This paper aims at facilitating debugging of structurally singular model through an analyzer in terms of reducing the number of tests to correct the model. When checking for singularities of a component, the analyzer first uses some fictitious equations to replace the constraint equations generated by the outside connections of the component, and then identifies whether the singularities derive from the component or not by structural analysis. The checking procedure is done recursively. The proposed analyzer can automatically identify faulty components that are responsible for the singularities.

Keywords: Modelica; debugging; structural analysis; structural singularity

1 Introduction

The need for mathematical modeling and simulation in engineering is continuously rising since technical systems become increasingly complex and physical prototyping becomes too expensive. Modelica is an object-oriented equation based language for efficient modeling and simulation of complex, heterogeneous, multi-domain physical systems described by ordinary differential and algebraic equations. Modelica allows describing simulation models in a declarative object-oriented manner, so that a model can be used to solve various problems and model components easily can be modified to describe similar systems. Following object-oriented modeling methodology, a Modelica model is structured as closely as possible to the corresponding physical system. In particular, models are defined in acausal form, independently of the context in which it is used. Complex models can be realized by aggregating more submodels and their connections within a composite larger model, which

may in turn be connected to other models. For continuous time modeling, by assembling the declarative equations of component models, this form of model representation gives rise to a larger scale system of differential algebraic equations (DAEs).

Along with its benefits, object-oriented modeling with Modelica language often unconsciously leads to an under-constrained problem or an over-constrained problem. In such situations, numerical solvers fail to find correct solutions to the underlying system of equations. Due to the high-level abstraction of equation based models, it is extremely hard to find and localize model singularities. Moreover, especially for complex physical systems, the underlying system of equations is of very large dimension. Even if a modeler suspects a redundant equation, it is very time-consuming to identify the equation in a large scale system of equations. Therefore, it is a key problem to check for the singularity of the model before using it for analysis or design purposes, or before generating the simulation code.

To address the aforementioned problem, Bunas and Fritzson^[1,2] have developed a debugging framework for Modelica models and have adapted traditional debugging techniques and algorithms to it. Equations and variables that probably cause the irregularities are isolated by applying graph decomposition techniques and reduced by using structural information and semantic information. The developed algorithms and methods help to statically detect and repair a broad range of errors without having to execute the simulation model.

This paper focuses on checking for structural singularities of equation based models. Our goal is to develop an analyzer, called *Model Singularity Analyzer* (MSA), to help the modeler to localize model singularities more quickly and more efficiently.

The paper is organized as follows: Section 2 discusses the three main steps of the MSA. The examples are demonstrated in Section 3. Section 4 gives a comparison of the MSA to existing methods. Section 5 presents our conclusions.

2 Structural analysis

2.1 Detecting structural singularities

Before solving a Modelica model, the Modelica source code is first translated into a so called “flat model”, which is a flat set of equations, constants, parameters and variables. For continuous time modeling, the flat model can be described by a system of differential algebraic equations of the form

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

where \mathbf{x} is the variable vector, \mathbf{u} is the input vector, \mathbf{p} is the parameter vector and t is time.

The system of equations resulting from a Modelica model can naturally be represented as a bipartite graph $G = \{V_1, V_2, E\}$ where V_1 denotes the set of equations, V_2 denotes the set of variables, and there is an edge $e \in E$ between a variable $v \in V_2$ and an equation $u \in V_1$ if variable v appears in equation u .

When using a model for simulation, it is a basic requirement that there are as many equations as variables, and that we can determine a mapping between equations and variables in such a way that each equation is related one and only one variable. If the requirement is satisfied, we call the model structural nonsingularity. Otherwise, the mode is structurally singular.

Therefore, a crucial step of checking for structural singularities is to assign each variable v_i to a unique equation e_j such that v_i appears in e_j . If it is impossible to pair variables and equations in this way then the model is structurally singular. This assignment procedure can be realized by calculating a perfect matching in the bipartite graph associated with the system of equations. If the perfect matching does not exist then the corresponding system of equations is structurally singular.

For a structurally singular model, we can isolate the over-determined and under-determined subsets of equations by the DM decomposition^[4]. However, when applied to a large system of equations, the DM decomposition finds a large under-determined or over-determined block, which is less helpful for localizing the structural singularities, so methods to help are necessary.

To check for structural singularities of DAE systems, we should not distinguish between the appearance of x and the appearances of its derivative, the appearances of \dot{x} are considered as appearances of x . Thus to check if the DAE system $F(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$ is struc-

turally singular, we check if the algebraic system $F(\mathbf{x}, \mathbf{x}, t) = 0$ is structurally singular with respect to \mathbf{x} .

2.2 Generating fictitious equations

A complete Modelica model is always made up of components, which maybe consist of other sub-components. For a composite model, the structural singularities may be caused by improper use of components, or come from structurally singular components. So if a composite model is structurally singular, we should check for structural singularities of its components to determine whether the singularities comes from its components or not.

However, a component of a model always has outside connections to communicate with the rest of the model. If we isolate a component from the environment where it is used, and check for structural singularities of the component, we can not determine which connection equations generated by outside connections should be included into the system of equations resulting from the component. The reason is connections between components are often acausal, i.e., the data flow in connection is not explicitly specified.

The object-oriented modeling methodology uses energy flow for modeling^[5,6]. It is necessary for the modeler, when designing model interfaces, to guarantee that connecting such models in an arbitrary fashion will always ensure that power, momentum, and mass are balanced at the interfaces. This means that the sum of incoming energy flows must equal the sum of outgoing energy flows at connection points. Hence, acausal connections in Modelica models in fact are a kind of connection based on energy flow.

Therefore, when checking for structural singularities of components, we disregard the connection equations generated by outside connections, and use some fictitious equations to compensate the lost constraints in the following way.

1. We generate a fictitious equation for each flow variable and make the equation contain all variables appearing in the same connector with the flow variable.
2. We generate for each input variable a fictitious equation which assigns a value to the corresponding input variable.
3. Potential variables may be more than flow variables in a connector. In such case, we assume some outside constraints on these redundant potential variables. To construct the most general constraint, we generate a fictitious equation for each redundant po-

tential variable, and make these equations contain all potential variables of all the connectors. Let C denote a model component, k be the number of the fictitious equations added for the redundant potential variables, r be the number of connectors of C , m_i be the number of potential variables in the i th connector, n_i be the number of flow variables in the i th connector, p be the number of variables of C , and q be the number of equations of C . There is the following dependence:

$$k = 0, \text{ if } p - (q + \sum_{i=1}^r n_i) < 0;$$

$$k = p - (q + \sum_{i=1}^r n_i), \text{ if}$$

$$0 \leq p - (q + \sum_{i=1}^r n_i) \leq \sum_{i=1}^r (m_i - n_i);$$

$$k = \sum_{i=1}^r (m_i - n_i), \text{ if } p - (q + \sum_{i=1}^r n_i) \geq \sum_{i=1}^r (m_i - n_i).$$

Where $\sum_{i=1}^r n_i$ is the number of fictitious equations added for flow variables.

The aim of case 1 is to construct a general energy flow constraint equation. To explain the idea, we first define a generic physical connector class as follows:

connector generic

Real e ; //potential variable

flow Real f ; //flow variable

end generic;

We assume $c1$ and $c2$ are two instances of the connector class generic. According to Modelica semantics, the connection `connect(c1, c2)` produces two equations, namely: $c1.e = c2.e$ and $c1.f + c2.f = 0$. From this two connection equations, we can get $c1.e * c1.f + c2.e * c2.f = 0$. We further assume $c1.e * c1.f = C$, where C is a constant, then $c2.e * c2.f = -C$. Since we only focus on which variables appear in each equation rather than how they appear when checking for structural singularities, we can use the general form equations $f1(c1.e, c1.f) = 0$ to express $c1.e * c1.f = C$, and $f2(c2.e, c2.f) = 0$ to express $c2.e * c2.f = -C$. However, there are sometimes more than one matched flow variable and potential variable in a connector. Hence, without loss of generality, we generate for each flow variable a general form equation which contains all flow and potential variables of a connector, i.e., $g(e1, e2, \dots, f1, f2, \dots) = 0$, where $f_i (i=1, 2, \dots)$ is flow variable, $e_i (i=1, 2, \dots)$ is potential variable.

The fictitious equations generated for flow variables can be regarded as constraint equations about power, momentum, and mass which passes connectors. This is because that, in any physical system, the power can be written as the product of two adjugate variables, called the potential and the flow in Modelica language. For example, in electrical systems the power $P = v * i$ where v denotes the voltage and i denotes the current, and in translational mechanical systems the momentum $P = v * f$ where v denotes the velocity and f denotes the force.

By generating fictitious equations, we can obtain the system of equations resulting from a component. By checking for structural singularities of the system of equations, we can determine that the component is structurally singular or not.

2.3 Locating faulty components

A connector can be connected, and also be not connected. When checking for singularities of a component, two possibilities are considered. It is noticed that, once a connector is assumed to be not connected, all other connectors of the same component are considered to be connected. If a connector is assumed to be not connected, the fictitious equations for flow variables of the connector are not generated, instead flow variables are set to zero.

If a structurally singular component is made up of subcomponents, the checking procedure can be done recursively, until the fault component is a primitive component, or each subcomponent of the fault component is structurally nonsingular.

For that, we introduce the following definition.

Definition 1. Let C be a structurally singular component. C is a minimal structurally singular (MSS) component if either of the following two conditions is satisfied:

1. C is a primitive component described in terms of equations;
2. C is a composite component consisting of other connected subcomponents, and none of its subcomponents is structurally singular.

We therefore propose the following strategy for locating structural singularities in Modelica models. Our aim is to locate all the MSS components of a structurally singular model. First, we check whether a perfect matching exists or not in the bipartite graph associated with a whole model. This can be done by solving the maximum matching problem^[7,8]. If a perfect matching does not exist, we apply the DM decomposition to isolate the over-determined and under-determined subsets of equations. Then we check

for structural singularities of each component in turn to determine whether the singularities derive from the component or not. So it is very natural to consider the following subproblem procedure.

Procedure $P(C)$

Input: a structurally singular component C

Output: the structurally singular subcomponent set T

begin

set $T := \emptyset$;

let Q be a subcomponent set; set $Q := \emptyset$;

add the subcomponents of C to Q ;

for each $C' \in Q$ **do**

begin

generate fictitious equations for C' ;

obtain the system of equations E of C' ;

construct the bipartite graph G for E ;

determine a maximum matching W in G ;

if W is not perfect **then**

add C' to T ;

end

end

By performing the above subproblem procedure, we can obtain all the singular subcomponents of a singular component, if they exist. For each singular subcomponent, we further perform the above procedure to obtain its singular subcomponents. This procedure is performed iteratively until the singular subcomponent is a MSS component. Clearly, the recursive application of the above procedure can construct a tree of subproblems. By using the depth first rule, we obtain our algorithm which outputs all the MSS components of a structurally singular model. The algorithm is described as follows:

Algorithm 1: Obtaining the MSS components

Input: a structurally singular component M

Output: the MSS components set S

begin

set $S := \emptyset$;

if M is composite **then**

begin

let L be the list of components;

set $L := P(M)$;

while L is not empty **do**

begin

let C be the last component in L ;

remove C from L ;

if C is composite **then**

begin

let K be a component set;

set $K := P(C)$;

if $K = \emptyset$ **then**

add C to S ;

else

add each element of K to the end of L ;

end

else

add C to S ;

end

end

else

add M to S ;

end

Finally, for each MSS component it is desirable to give the user some hints what is wrong. If a MSS component is a composite component, we inform the user that some subcomponents of the MSS component may be improperly used. If a MSS component is a primitive component, we produce some hints by locating critical parts of the component that are responsible for singularities. For an over-constrained problem, the redundant equations must appear in both the over-determined subset and the MSS component. Similarly, for an under-constrained problem, the free variables must appear in both the under-determined subset and the MSS component.

3 Examples

The first example is an oscillator model depicted in figure 1. A mass Ma is hanging in a spring Sa which is connected to a fixed housing Fa . The mass Ma is subject to the gravitational force and the force from the spring. It is given an initial position $s = -0.5$, which is offset from the equilibrium position and therefore starts an oscillating movement up-and-down. The positive coordinate direction is upward in the figure, which applies to both positions and forces. The Modelica description of the oscillator is presented as follows:

model Oscillator

Mass $Ma(L=1, s(\text{start}=-0.5))$;

Spring $Sa(s_{\text{rel0}}=2, c=10000)$;

Fixed $Fa(s_0=1.0)$;

equation


```

connect(Sa.flange_b, Fa.flange_b);
connect(Ma.flange_b, Sa.flange_a);
end Oscillator;

```

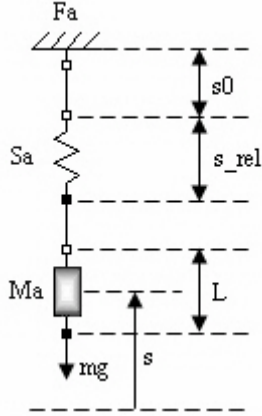


Figure 1. The oscillator Model

The component model Spring and Fixed are available in Modelica class libraries. The definition of the component model Mass is presented as follows:

model Mass

```

extends Rigid;
parameter Real m = 1;
constant Real g = 9.81;
Real v;
Real a;

```

equation

```

v = der(s);
a = der(v);
flange_b.f = m*a - m*g;
v = 6; //an additional equation

```

end Mass;

In order to obtain an over-constrained problem, we introduce an additional equation ($v=6$) in the model Mass. The set of equations generated from the Oscillator model is presented in table 1.

Table 1. The set of equations and variables corresponding to the Oscillator model

e1: $Ma.v = \text{der}(Ma.s)$	v1: $Ma.s$
e2: $Ma.a = \text{der}(Ma.v)$	v2: $Ma.v$
e3: $Ma.flange_b.f = Ma.m * Ma.a$ $- Ma.m * Ma.g$	v3: $Ma.a$
e4: $Ma.v = 6$	v4: $Ma.flange_a.s$
e5: $Ma.flange_a.s = Ma.s - Ma.L/2$	v5: $Ma.flange_a.f$
e6: $Ma.flange_b.s = Ma.s + Ma.L/2$	v6: $Ma.flange_b.s$
e7: $Sa.f = Sa.c * (Sa.s_rel - Sa.s_rel0)$	v7: $Ma.flange_b.f$
e8: $Sa.s_rel = Sa.flange_b.s$ $- Sa.flange_a.s$	v8: $Sa.s_rel$
e9: $Sa.flange_a.f = -Sa.f$	v9: $Sa.f$
e10: $Sa.flange_b.f = Sa.f$	v10: $Sa.flange_a.s$
	v11: $Sa.flange_a.f$
	v12: $Sa.flange_b.s$

e11: $Fa.flange_b.s = Fa.s0$	v13: $Sa.flange_b.f$
e12: $Ma.flange_b.s = Sa.flange_a.s$	v14: $Fa.flange_b.s$
e13: $Ma.flange_b.f + Sa.flange_a.f = 0$	v15: $Fa.flange_b.f$
e14: $Fa.flange_b.s = Sa.flange_b.s$	
e15: $Fa.flange_b.f + Sa.flange_b.f = 0$	
e16: $Ma.flange_a.f = 0$	

Performing the DM decomposition, the over-constrained subgraph is found and represented graphically in figure 2, where the covered edges by the maximum matching are marked by thick lines.

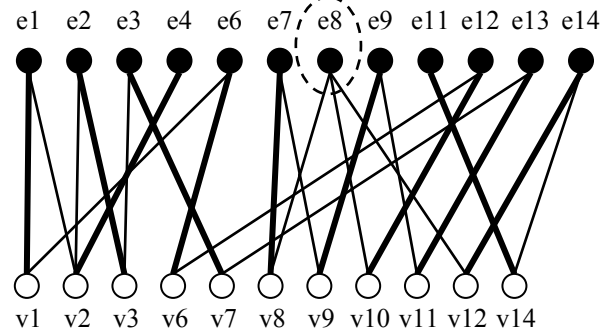


Figure 2. The over-constrained subgraph

When check for structural singularities of the component Ma, we first assume that both the connectors flange_a and flange_b are connected, and generate e1': $f(Ma.flange_a.f, Ma.flange_a.s) = 0$ for the flow variable $Ma.flange_a.f$ and e2': $f(Ma.flange_b.f, Ma.flange_b.s) = 0$ for the flow variable $Ma.flange_b.f$. The corresponding bipartite graph to the component Ma is shown in figure 3, where a maximum matching is marked by thick lines.

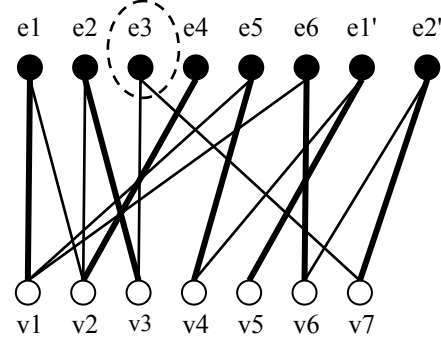


Figure 3. The bipartite graph corresponding to the component Ma with a maximum matching

In figure 3, e3 is a free vertex, so the component Ma is structurally singular, and there exists one redundant equation. It means the primitive component Ma is a MSS component. The equations that appear in both the over-constrained subgraph and the component Ma are e1, e2, e3, e4 and e6, one of which is redundant. Similarly, by generating fictitious, we can determine the components Sa and Fa are structurally

nonsingular. In this case, the following message is presented to the modeler.

Error: The model Oscillator is structurally singular.
The singularity comes from the component Ma.
There is 1 redundant equation in the equations:

```

v = der(s);
a = der(v);
flange_b.f = m*a- m*g;
v = 6;
flange_b.s = s+L/2;

```

Figure 4. The error message for the model Oscillator

The second example is an AC motor model depicted in figure 5. This model contains components from the two domains: mechanical domain and electrical domain.

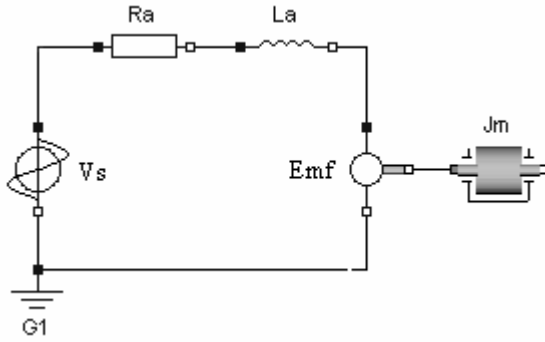


Figure 5. The AC motor model

The Modelica description of the ACMotor model appears as follows:

model ACMotor

```

SineVoltage Vs(V=220,freqHz=50);
Resistor Ra(R=0.5);
Inductor La(L=0.1);
EMF Emf;
Inertia Jm(J=0.001);

```

Ground G1;

equation

```

connect(Vs.p, Ra.p);
connect(Ra.n, La.p);
connect(La.n, Emf.p);
connect(Emf.flange_b, Jm.flange_a);
connect(Emf.n, G1.p);
connect(Vs.n, G1.p);

```

end ACMotor;

The component models Inductor, EMF, Inertia and Ground are available in Modelica class libraries. In order to make the ACMotor singular, the following Resistor model is defined.

model Resistor

```

extends OnePort;
parameter Real R=1;
Real s;

```

equation

```

R*i = v+s;
p.v=12;

```

end Resistor;

The complete set of equations (shown in Table 2) generated from the ACMotor class consists of 37 differential algebraic equations and 37 variables. This is a structurally singular problem where under-constrained and over-constrained situations appear simultaneously. The DM decomposition will find the over-constrained, well-constrained and under constrained subgraphs. The over-constrained subgraph contains equations e1, e4, e9, e20, e29, e30 and e37, and variables v1, v3, v5, v7, v25 and v29. The well-constrained subgraph contains equation e33 and variable v34. All other equations and variables are contained in the under-constrained subgraph. Because of space limitation, the over-constrained and under-constrained subgraphs are not depicted here.

Table 2. The set of equations and variables corresponding to the AC motor model

e1: $Vs.v = Vs.p.v - Vs.n.v$	v1: $Vs.p.v$
e2: $0 = Vs.p.i + Vs.n.i$	v2: $Vs.p.i$
e3: $Vs.i = Vs.p.i$	v3: $Vs.n.v$
e4: $Vs.v = Vs.V \sin(2 \cdot Vs.PI \cdot Vs.freqHz \cdot time)$	v4: $Vs.n.i$
e5: $Ra.v = Ra.p.v - Ra.n.v$	v5: $Vs.v$
e6: $0 = Ra.p.i + Ra.n.i$	v6: $Vs.i$
e7: $Ra.i = Ra.p.i$	v7: $Ra.p.v$
e8: $Ra.R \cdot Ra.i = Ra.v + Ra.s$	v8: $Ra.p.i$
e9: $Ra.p.v = 12$	v9: $Ra.n.v$
e10: $La.v = La.p.v - La.n.v$	v10: $Ra.n.i$
e11: $0 = La.p.i + La.n.i$	v11: $Ra.v$
e12: $La.i = La.p.i$	v12: $Ra.i$
e13: $La.L \cdot der(La.i) = La.v$	v13: $Ra.s$

e14: $\text{Emf.v} = \text{Emf.p.v} - \text{Emf.n.v}$	v14: La.p.v
e15: $0 = \text{Emf.p.i} + \text{Emf.n.i}$	v15: La.p.i
e16: $\text{Emf.i} = \text{Emf.p.i}$	v16: La.n.v
e17: $\text{Emf.w} = \text{der}(\text{Emf.flange_b.phi})$	v17: La.n.i
e18: $\text{Emf.k} * \text{Emf.w} = \text{Emf.v}$	v18: La.v
e19: $\text{Emf.flange_b.tau} = -\text{Emf.k} * \text{Emf.i}$	v19: La.i
e20: $\text{G1.p.v} = 0$	v20: Emf.v
e21: $\text{Jm.w} = \text{der}(\text{Jm.phi})$	v21: Emf.i
e22: $\text{Jm.a} = \text{der}(\text{Jm.w})$	v22: Emf.w
e23: $\text{Jm.J} * \text{Jm.a} = \text{Jm.flange_a.tau} + \text{Jm.flange_b.tau}$	v23: Emf.p.v
e24: $\text{Jm.flange_a.phi} = \text{Jm.phi}$	v24: Emf.p.i
e25: $\text{Jm.flange_b.phi} = \text{Jm.phi}$	v25: Emf.n.v
e26: $\text{Jm.flange_a.phi} = \text{Emf.flange_b.phi}$	v26: Emf.n.i
e27: $\text{Emf.flange_b.tau} + \text{Jm.flange_a.tau} = 0$	v27: Emf.flange_b.phi
e28: $\text{Emf.n.i} + \text{G1.p.i} + \text{Vs.n.i} = 0$	v28: Emf.flange_b.tau
e29: $\text{G1.p.v} = \text{Emf.n.v}$	v29: G1.p.v
e30: $\text{Vs.n.v} = \text{Emf.n.v}$	v30: G1.p.i
e31: $\text{Emf.p.i} + \text{La.n.i} = 0$	v31: Jm.flange_a.phi
e32: $\text{La.n.v} = \text{Emf.p.v}$	v32: Jm.flange_a.tau
e33: $\text{Jm.flange_b.tau} = 0$	v33: Jm.flange_b.phi
e34: $\text{La.p.i} + \text{Ra.n.i} = 0$	v34: Jm.flange_b.tau
e35: $\text{Ra.n.v} = \text{La.p.v}$	v35: Jm.phi
e36: $\text{Ra.p.i} + \text{Vs.p.i} = 0$	v36: Jm.w
e37: $\text{Vs.p.v} = \text{Ra.p.v}$	v37: Jm.a

When check for structural singularities of the component Ra, if we assume the connector p is connected and the connector n is not connected, we can determine that Ra is structurally singular. The bipartite graph corresponding to Ra is presented in figure 6, where $e1': f(\text{Ra.p.v}, \text{Ra.p.i}) = 0$ is the fictitious equation generated for flow variable Ra.p.i, and $e2': \text{Ra.n.i} = 0$ is used to set flow variable Ra.n.i to zero.

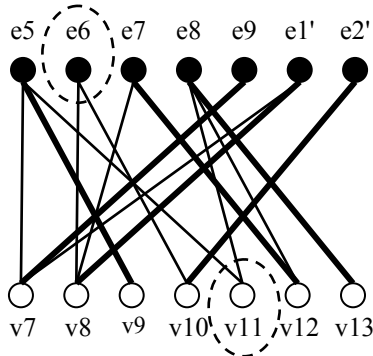


Figure 6. The bipartite graph corresponding to the component Ra with a maximum matching

In figure 6, e6 and v11 are free vertices, so the primitive component Ra is a MSS component. All other components of the model ACMotor are structurally nonsingular.

In this case, only the equation e9 appears in both the over-constrained subgraph and the component Ra. The variables that appear in both the under-constrained subgraph and the component Ra are v8, v9, v10, v11, v12 and v13. For this model, the error message is presented in figure 7.

Error: The model ACMotor is structurally singular. The singularity comes from the component Ra. There is 1 redundant equation in the equations:
 $p.v=12$;
 1 equation is missing for the variables:
 $p.i$;
 $n.v$;
 $n.i$;
 v ;
 i ;
 s ;

Figure 7. The error message for the model ACMotor

The third example is a modified AC motor depicted in figure 8, where the motor contains two ground points instead of one. The Modelica description of the modified motor model appears as follows:

model ModifiedMotor

SineVoltage Vs(V=220,freqHz=50);
 Resistor Ra(R=0.5);
 Inductor La(L=0.1);
 EMF Emf;
 Inertia Jm(J=0.001);
 Ground G1;

equation

```
connect(Vs.p, G2.p);
connect(Ra.p, G2.p);
connect(Ra.n, La.p);
connect(La.n, Emf.p);
connect(Emf.flange_b, Jm.flange_a);
connect(Emf.n, G1.p);
connect(Vs.n, G1.p);
```

end ModifiedMotor;

All the component models of the model ModifiedMotor are available in Modelica class libraries.

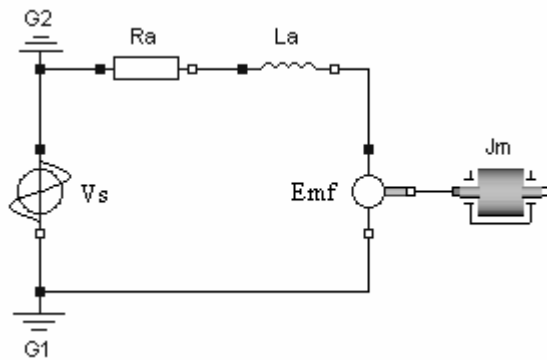


Figure 8. The AC motor with two ground points

The modified motor model also leads to a structurally singular problem where under-constrained and over-constrained situations appear simultaneously. When checking for structural singularities, all the components are determined to be structurally non-singular. So the top model ModifiedMotor is the MSS component.

In this case, the singularities are caused by improper use of components. To correct the model, one should remove the redundant ground point G2 instead of some equations and variables. Hence, for this model the following message is presented.

Error: The model ModifiedMotor is structurally singular.
 The singularity may be caused by improper use of the components.
 Please check whether the components are used properly or not.

Figure 9. The error message for the modified motor

4 Comparison

The three examples presented in Section 3 illustrate that the MSA can automatically identify fault components and localize model singularities. It is very useful for the modeler to correct singular models. For a complex singular model, it is advisable to localize model singularities in such a way.

Currently, there are only a few methods that can help the modeler to debug singular equation-based models. For the first and the second examples, the method proposed in [1,2] is helpful, and can present efficient messages. However, for the third example where the singularities are not caused by equations and variables, the method can not deal with it. Moreover, it may be less efficient to debug complex models only by using structural information and semantic information.

If a structurally singular problem is caused by an over-constrained or under-constrained component, Dymola can identify such singular components. For the first example, Dymola can find the faulty component Ma and give the modeler efficient message. For the second example, Dymola does not find the faulty component Ra and considers the singularity is at the top level, and only informs the modeler that there is 1 one equation too many in a set of 7 equations and that 1 equation is missing for a set of 33 variables. For the third example, Dymola also consider the singularity is at the top level, and inform the modeler that there is 1 one equation too many and that 1 equation is missing, so the presented message is less helpful.

5 Conclusions

In this paper we have discussed an analyzer for declarative equation-based models. The examples presented in Section 3 are all quite trivial. However, they illustrate that it is possible to identify faulty components of a structurally singular model. From the modeler's point of view, the MSA is very beneficial because it can make correcting structurally singular models more quickly by automatically identifying faulty components and providing efficient error messages to show what is wrong.

The proposed techniques and strategies are also suitable for other object-oriented equation based modeling languages and not only for Modelica.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (Grant No. 60574053), the National High-Tech Development 863 Program of China (Grant No. 2003AA001031), and the National Basic Research 973 Program of China (Grant No. 2003CB716207).

References

- [1] Bunus P, Fritzson P. Automated static analysis of equation-based components. *Simulation: Transactions of the Society for Modeling and Simulation International*, 2004, 80(8):321-345
- [2] Bunus P. An empirical study on debugging equation-based simulation models. In *Proceedings of 4th International Modelica Conference*, Hamburg, Germany, 2005
- [3] Asratian A S, Denley T, Häggkvist R. *Bipartite Graphs and their Applications*, Cambridge University Press, 1998
- [4] Dulmage A L, Mendelsohn N S. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 1963, 10:517-534
- [5] Cellier FE, Elmqvist H, Otter M. Modeling from Physical Principles. *The Control Handbook*, CRC Press, pp.99-108
- [6] Fritzson P. Principles of object-oriented modeling and simulation with Modelica 2.1, IEEE Press, 2003
- [7] Hopcroft J E, Karp R M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 1973, 2(4): 225-231
- [8] Uno T. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In *Lecture Note in Computer Science 1350*, Springer-Verlag, 1997, pp. 92-101

Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools

Olof Johansson, Adrian Pop, Peter Fritzson

PELAB – Programming Environment Lab, Dept. Computer Science

Linköping University, S-581 83 Linköping, Sweden

{olojo,adrpo,petfr}@ida.liu.se

Abstract

This paper briefly describes some international standards used for engineering design tools that precede simulation in the product development process. Very much information in such design tools can be reused when developing Modelica simulation models. Examples are product structures, component parameters, and component connection information.

The Modelica Standard Library (MSL) with the latest version 2.2.1 has grown significantly the last years. An analysis of the contents of MSL is provided, and a classification method described to ease the work of mapping structures, component parameters and connections in engineering design tools such that their information content can be reused for development of Modelica simulation models.

ModelicaXML has been upgraded to support Modelica 2.2, and is briefly described as one of the most promising intermediate formats for exchange of models between engineering design tools and Modelica simulation tools.

1 Introduction

Modelica is currently the most promising language for developing integrated simulation models of power plants that cover the process-, electrical- and control system aspects. Today such integrated simulation models of a whole power plant are in practice not possible to develop at reasonable costs, because of the lack of compatible interoperating standards.

Currently plant industry struggles with classification standards for electronic data exchange of plant models. When this struggle has settled on a reasonable set of working standardized solutions, one coming large standards struggle will be in exchange of simulation models for power plants, subsystems, and components provided by the supply chain. Integrated simulation mod-

els can provide significant added value to the current engineering systems during the whole life cycle of a power plant.

This paper outlines a path to bridge the gap between different standardized engineering languages supported by commercial engineering tools and Modelica simulation tools.

2 Contents of International Standards for Engineering Design Tools

ISO, IEC, DIN and VGB provide standards for the power plant industry, which are supported by commercial engineering tools for power plant system design. These standards apply to:

- 1) Structures and designation systems
- 2) Symbols on graphical diagrams
- 3) Document classification
- 4) Component classification
- 5) Electronic information exchange

Figure 1. Standardized Plant Model Features

The systems engineering [2] of a power plant is usually divided into:

- 1) Process Engineering
- 2) Electrical Engineering
- 3) Control Systems Engineering

Figure 2. Engineering Disciplines

Each of these engineering disciplines has their special views on a product model of a power plant, which often follow an international standard.

A designation system standard specifies how to name an object in the plant, and how to refer to it with relative names on printed documentation like diagrams and data listings. Power plant designation system stan-

standards like KKS [27] and IEC 61346 [13] provide three major aspects:

- 1) Function aspect – the function and functional organization of a modeled system
- 2) Location aspect – the physical locations of subsystems and components
- 3) Product aspect – the assembly structure of the system implementation

Figure 3. Major aspects

The same modeled object (e.g. a pump, or electrical motor) participates in all these aspects, but provides different types of component related information in each of them. Each aspect has its own structure, so it is possible to navigate to an object using a functional structure, then change to the location aspect and find out where it is located in the plant.

The functional aspect contains most relevant information for Modelica simulation models. However, the location aspect, together with geometric layout drawings, provide additional parameter information like the length of pipes, positions with regards to height, lengths of electrical cables to calculate losses due to resistance, etc.

3 Illustration Example

This section illustrates the contents of standardization listed in Figure 1 to Figure 3.

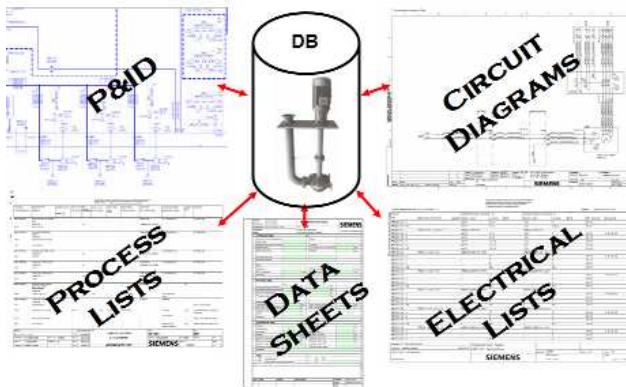


Figure 4. Aspects on a lubrication oil pump

Figure 4 shows a pump object in a plant product model database, and how this object's information is made available in various standardized document views in an engineering design system. More specifically it is a centrifugal oil pump immersed in the oil tank. This pump is driven by an electrical motor on the top of the oil tank.

3.1 Process Diagrams

A Process and Instrumentation Diagram (P&ID) shows what function the pump has in the lubrication oil system. The P&ID is developed by process engineers, and is part of the function aspect of the plant. The symbols on the P&ID are standardized (e.g. acc to DIN 2481 or ISO 14617), and the P&ID belongs to a standardized document class (acc to IEC 61355). The designations of objects on the P&ID are standardized according to KKS.

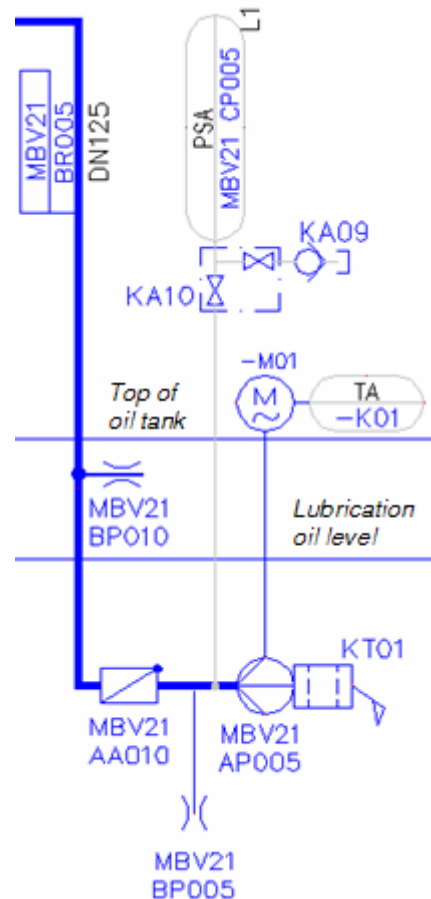


Figure 5. Detail from P&ID

Figure 5 shows a part of the P&ID in detail. The KKS standard for the structure and designation system decides the name of the different objects on the P&ID. The designation of the pump is **MBV21 AP005**. This standardized designation tells that the pump is a part of a lubrication system (KKS Function Key MBV), and that it is a pump (KKS Equipment Unit Key AP). When the designation is printed on a Process List with adjacent parameter data, a user can directly infer from its name that it is a pump within a lubrication system.

The complete KKS designation of the electrical motor is **MBV21 AP005 -M01**, which tells that it is an electrical motor (KKS Component Key -M), which is mounted on the pump equipment unit.

Similarly, the standardized graphical shape of the

symbols and their KKS designations provide information like: **MBV21 CP005** represents a pressure measurement instrument, **MBV21 BP005** a flow restrictor, **MBV21 AA010** a non-return valve, **MBV21 BR005** a pipe where the flag is oriented in the direction of the flow. The symbols are all according to DIN 2481, which is mostly compatible with the more modern and extendable process symbol standard ISO 14617.

3.2 Process Lists and Data Sheets

Back to Figure 4, different types of process lists provide selected parameter data from all components within the lubrication system. These are used for various process engineering purposes. A component class specific data sheet collects all parameter data for the pump object in one document. The data sheet may be used as information supply by several engineering disciplines.

A pump belongs to a certain class (e.g. the KKS Equipment Unit Key AP), which tells what class specific attributes should be available to fill in on data sheets or display on listings. Different component classification standards (e.g. IEC 61346-2) may organize the classification of objects in other ways. For historical reasons different tool vendors and engineering companies have had to invent their own component classification standards, to provide the necessary functionality for entering parameter data. This is what the big struggle towards common classification standards is all about.

3.3 Electrical Diagrams, Lists, and Data Sheets

To proceed with another engineering discipline, with its own aspects in Figure 4, the circuit diagram shows how the power to the electrical motor is connected through protection devices, and how a built-in thermistor for temperature measurement is connected through various terminal boxes. Other circuit diagrams show how the wires, identified by their reference designations finally end up at an IO card in the control system. Circuit diagrams are developed by electrical engineers and follows IEC 61082 which provides general guidelines of what information to present in electrical documents. A specific standard IEC 60617 specifies the graphical appearance of the symbols.

Various electrical lists, for example cable tables, connection tables for wires etc., provide connection information. Other electrical lists provide subsets of parameter data from different electrical devices. The overall course classification of electrical devices, which in turn determines what attributes should be available follows IEC 61346-2, or now obsolete standards like IEC 750.

3.4 Document Classification

A complete plant installation contains several thousands of objects, and there is a large number of P&IDs, circuit diagrams, lists and data sheets. These documents must be organized into structures to be easy to find. A document classification standard is used for identification and searching for different kinds of document types. More specifically each document class has a document kind classification code (DCC), whose lettering and format is determined by IEC 61355.

4 International Standards for Engineering Design Tools

In the following a number of standards will be described according to the structuring introduced in section 2, and brief comments given.

4.1 Structures and Designation Systems

Process and Electrical design

- KKS – "Kraftwerk Kennzeichen System" [22]-[28] is divided into two parts. 1) KKS Rules [25],[27] and 2) KKS Application Commentaries [26],[28]. 2) shows examples of how to designate (give a name to) engineering objects in different disciplines like Process, Electrical and Control systems engineering. 1) describes how the designation system is organized, and provides 3 listings of classification keys and their descriptions.
- Function Keys – 3 letter hierarchical classification of system functions within a plant. **Examples:** Level 1: **M** *Main machine sets*, Level 2: **MB** *Gas turbine plant*, Level 3: **MBV** *Lubricant supply system*. There are about 800 standardized keys with many letter code series on level 3 left open for company- or project specific standardization.
- Equipment Unit Keys – 2 letter hierarchical classification of equipment unit functions. **Examples:** Level 1: **A** *Mechanical equipment*; **C** *Direct measuring circuits*. Level 2: **AP** *Pump units*; **AC** *Heat exchangers, heat transfer surfaces*; **CP** *Pressure*. About 130 standardized keys.
- Component Keys – 2-letter hierarchical classification of component functions. **Examples:** Level 1: - *Electrical components*; **K** *Mechanical components*. Level 2: -**M** *Motors*; **KA** *Gate valves, globe valves, dampers, cocks, rupture disks, orifices*. About 90 standardized keys.
- IEC 61346 - Industrial systems, installations and

equipment and industrial products – Structuring principles and reference designations [13]-[17]. Provides a framework for a designation system with similar functionality as KKS. IEC 61346-2 [15] standardizes a 1 letter functional classification system for components that can be extended with appended letters for more specific subclasses according to industrial branch, company or project specific standards. Examples: Level 1: **M** *Drive, act*; **Q** *Open, close, vary*; **R** *Restrict flow or motion*; **W** *Guide*; **X** *Connect*. The classification letters can be used in the component designations to indicate the class of the object.

- IEC 61082 - Preparation of documents used in electro technology [31]-[35] provides very general rules and guidelines for electrical documents like circuit diagrams, electrical tables and lists. It establishes a kind of minimum criteria for documents suitable for electrical engineering purposes and gives much freedom to tool developers.

4.2 Symbols on Graphical Diagrams

Process design:

- DIN 2481 Thermal Power Generating Plants, graphical symbols [29] – provides a library of graphical symbols with description, identification code and short application examples. Contains commonly used process design symbols.
- DIN 1219 Fluid Power systems and components [30] – provides a library of graphical symbols of which a subset complements DIN 2481 for process design, with regards to hydraulically operated valve arrangements for steam inlets etc.
- ISO 14617 Graphical symbols for diagrams - Part 1-15 [52]-[67] - The best engineered graphical symbols standard available today for process design. A major benefit compared to other standards is its consistency and extensibility. By reusing existing standardized graphical symbols and following the standardized application rules, new symbols can be assembled that still conform to the standard.

Electrical design:

- IEC 60617 Graphical Symbols for Diagrams - online database [50] – The most comprehensive standard library of electrical symbols. Symbols are built from standard primitive symbols. They are annotated with a function class according to IEC 61346-2. The on-line database succeeds the original document release of IEC 617 [36].
- ISO 14617 – See above. Sometimes it is necessary to draw symbols of process equipment on circuit diagrams to adequately specify the circuit function.

Control system design

- IEC 61131-3 Programmable controllers - Part 3: Programming languages [51] – provides a textual and graphical language that consists of programs, function blocks and configuration elements. Configuration elements are configurations, resources, tasks, global variables, access paths, and instance-specific initializations, which support the installation of PLC programs into programmable controller systems. The **function block diagram** (FBD) is a graphic language for PLC programming which is consistent, as far as possible, with IEC 60617-12 Binary logic elements [48].

4.3 Document Classification

All engineering disciplines:

IEC 61355 Classification and designation of documents for plants, systems, and equipment – provides document kind classification codes (DCCs) which is based only on the content of information in the document. It also provides a document designation system that starts with the designation of the object that the document describes (e.g. a KKS or IEC 61346 designation), adds a "&" and the DCC to create the unique document designation.

DCC Examples: **FB103** *Piping and instrument diagram (P&ID)*; **FS101** *Circuit diagram*.

The general DCC format is A1 A2 A3 NNN, where the code letter A1 is an optional identifier for the technical area¹, A2 and A3 hierarchical classification codes, and NNN a document kind counting number for company or project specific standardization.

The code letter A2 may be:

- A Documentation-describing documents
- B Management documents
- C Contractual and non-technical documents
- D General technical information documents
- E Technical requirement and dimensioning documents
- F Function-describing documents
- L Location documents
- M Connection-describing documents
- P Product listings
- Q Quality management documents; safety-describing documents
- T Geometry-related documents
- W Operation records

¹ E.g. A1 = **P** *Process engineering*, **E** *Electrotechnology*

The code letter A3 provides a subclass to A2.

4.4 Component Classification

Process design:

- KKS Equipment Unit and Component keys – see Section 4.1

Electrical design:

- IEC 61346-2– see section 4.1.

4.5 Information Exchange

The following international classification standards are primarily developed for conducting e-business. However, a significant fraction of these standards are applicable to simulation models.

- IEC 61360 Standard data element types with associated classification scheme for electric components [19] – The first sustainable engineered international standard for information exchange.
- ecl@ss [21] – a non-profit organization of German origin who together with its ~30 German and European member companies provide the probably largest classification system in the world today. The dictionary version 5.1.1 contains ~27 000 classes, ~7000 attributes that are reused on the classes ~440 000 times, and ~4500 value codes (enumeration literals) for enumeration attributes. It is free for download after registration.
- RosettaNet technical Dictionary (RNTD) [23] – RosettaNet is a non-profit consortium of more than 500 organizations working to create, implement and promote open e-business standards and services. RNTD version 4.1.1 contains 966 classes, 888 sets of attributes that are reused on many different classes and 4147 attributes which are reused in the sets of attributes.

A classification system standard defines:

- 1) A basic terminology framework for describing classes, attributes and value codes.
- 2) An electronic exchange format for a dictionary of classes, attributes and value codes.
- 3) Maintenance procedures for updating the dictionary
- 4) A standard library with a large number of classes, attributes and value codes.

The above examples lack many technical attributes for parameters used during design and simulation. They are however extendable, and build on the IEC 61360 framework for an international classification system.

Such a framework stays stable and working for a large fleet of integrated continuously operating e-business systems, which must remain operational over successive upgrades of the standard library. Such issues must be taken into consideration already in the standard, since it may take several years before all integrated systems have upgraded to a new standard library release.

5 Analysis of Modelica Standard Libraries

An analysis of Modelica Standard Libraries (MSL) version 1.5, 2.1, 2.2 and 2.2.1 gives the following results:

MSL Version:	1_5	2_1	2_2	2_2_1
Source files	36	87	106	111
Imports	93	286	411	488
Definitions	910	1447	3823	4209
Components	1628	4636	10182	11444
Equations	1055	2768	3841	4269
Algorithms	99	633	3067	3351
Connect-equations	370	903	1574	1801
Component-references	30304	60838	123217	136900
Expression-lists	14736	23715	38535	41795
Real literals	4413	5833	38418	38251
Comments	1720	4755	9052	10482
String-Comments	1322	3722	8805	10282
Annotations	1326	3120	6377	7633
String-literals	3503	7218	17575	20579
Integer-literals	33187	59604	88609	98174
Other	88621	156857	310638	339147
Total elements:	183323	336422	664230	728916

Figure 6. Analysis of MSL versions

The number columns show the Modelica language element count from different releases of the Modelica standard libraries. MSL 1_5 was downloaded from the public library page [5]. MSL 2_1 were obtained from the Modelica CVS repository 2004-11-15. MSL 2.2 and 2.2.1 were accessed from the Modelica SVN repository.

The source code directory contents of these libraries was converted to a single xml file for each library release by ModelicaXML [9], which then were preprocessed for import into ModelicaDB [11].

The *Imports* row is an indicator of reuse. The *Component-references* row gives the count of the uses of component variables in expressions. *Connect-equations* give the number of connections amongst components within the libraries.

The *Comment* row is a higher level parse node for *String_comments* and *Annotations*.

String literals and *Integer literals* are heavily used within annotations, especially for graphical object annotations in Modelica diagrams.

The above analysis shows that the sizes of the standard library versions are substantial. Commercial Modelica development tools like [1],[3] provide user interfaces with tree views of the package hierarchy, connection diagrams, and string based text searches, for quick navigation in the libraries.

6 Method for Classification of MSL Components

Commercial design tools for process and electrical engineering usually contain a classification system for organizing and implementing their component library catalogues. Classification trees allow a user to with a few clicks down the tree to find a set of relevant component models. The classification tree itself is an aid for remembering where to find certain components. Many tools use an existing standard like IEC 61346-2 or KKS to organize the upper hierarchical levels and have tool specific customizable classification structures at lower levels. Users familiar with the standard can thus easily find the tree branch that may contain the component searched for.

This section briefly describes a successful approach to add classification information to a large class library, how to implement a mapping between different classification system standards, and some suggestions how to apply this method for classifying MSL components according to different standards.

6.1 The ecl@ss and its Mapping to KKS and IEC 61346-2

Within a larger project at Siemens Industrial Turbo machinery AB in Finspång, Sweden, a method and tool support was developed for mapping different classification systems.

Figure 7 shows an example of a navigator for a classification tree that enables users familiar with the 4-

level ecl@ss (see section 4.5) classification structure to quickly find a class of interest, its attributes, their units, or value codes (if it was an enumeration attribute).

L1	preferred_name_EN			coded_name
27	Electric engineering, automation, process control engineering			27000000
L2	preferred_name_EN			coded_name
+ 01	Generator			27010000
+ 02	Electrical drive			27020000
L3	preferred_name_EN			coded_name
+ 21	Low-voltage three-phase current asynchronous motor			27022100
+ 22	High voltage three-phase current asynchronous motor			27022200
+ 23	Single-phase alternating current motor			27022300
+ 24	Synchronous motor			27022400
+ 25	DC engine			27022500
+ 26	Servo motor			27022600
L4	preferred_name			coded_name
+ 03	Servo DC motor			27022603
	idatt	preferred_name	definition	unit_of
	+ BAC077001	Rated voltage	Manufacturer's value for the voltage, which is derived from measured values that have been obtained	V
	+ BAE088001	Rotation speed at continuous torque	Rotation speed which is reached with continuous torque.	1/min
	+ BAE129001	Thermal motor protection	Protection of the motor against overload and/or short circuit, particularly protection of the coil insulation	
	idvl	preferred_name	definition	preferred
	+ BAB643001	Cold conductor		Kaltleit
	+ BAB648001	continual temp. recording		kontinu
	+ BAB693001	Thermostats		Thermo
	+ WAA090001	Other		Sonstig
	+ WPA050001	without		ohne

Figure 7. ecl@ss navigator

Siemens uses KKS as a designation system for their power plants. There was a need to map the KKS classification to ecl@ss. A mapping tool was implemented that enables product experts familiar with KKS to map KKS classes to ecl@ss classes. This tool provides interactive associative navigation and search facilities through several different categorization/classification standards like KKS, ISO 31 [20] and IEC 61360 [19].

Once the experts had become familiar with ecl@ss and categorized a business relevant set of ecl@ss classes according to KKS by assigning KKS keys, the result was browsable in a navigator that enables users familiar with KKS to quickly find the relevant ecl@ss classes, their attributes and value codes.

Figure 8 shows parts of the resulting navigation tree for KKS Equipment Units to ecl@ss classes.

The same approach may be used for mapping international classification standards like KKS and IEC 61346-2 to Modelica components in MSL.

The versions of the Modelica Standard Library (MSL) already have a standard classification structure in their package hierarchies. However, secondary categorization trees organized according to different standards can be implemented as an “add-on” feature to the Modelica Standard Library, given some tool support for reading the classification tree from a file, indexing the Modelica components according to the classification

key, and display the tree in a navigator.

	A1	description_EN			
▶	A	Mechanical equipment			
	A1A2	description_EN			
	+	AA	Valves, dampers, etc., incl. actuators, also manual, r		
	+	AB	Isolating elements, air locks		
	+	AC	Heat exchangers, heat transfer surfaces		
	+	AE	Turning, driving, lifting and slewing gear (also manipu		
	+	AF	Continuous conveyors, feeders (escalators)		
	+	AG	Generator units		
	+	AH	Heating, cooling and air conditioning units		
	+	AJ	Size reduction equipment, only as part of process		
	+	AK	Compacting and packaging equipment, only as part c		
	+	AM	Mixers, agitators		
	+	AN	Compressor units, fans		
	▶	AP	Pump units		
			L1	L2	L3 L4 preferred_name
	+	36	41	01	01 Radial centrifugal pump
	+	36	41	01	02 Submersible motor pump
	+	36	41	01	03 Pump with non-clogging impeller
	+	36	41	01	04 Regenerative pump
	+	36	41	01	05 Circulation accelerating pump
	+	36	41	01	06 Axial-, semiaxial-flow pump
	+	36	41	01	07 Fluid draw works/booster pump
	+	36	41	01	90 Centrifugal pump (unclassified)
	*				
	+	AS	Adjusting and tensioning equipment for non-electrical		
	+	AT	Cleaning, drying, filtering and separating equipment, r		
	+	AU	Braking, gearbox, coupling equipment, non-electrical		
	+	AV	Combustion equipment		
	+	AW	Stationary tooling, treatment equipment		
	+	AX	Test and monitoring equipment for plant maintenance		
	+	AZ	(not provided in KKS AggregateKeyImport)		
	*				
+	B	Mechanical equipment			
+	C	Direct measuring circuits			
+	D	Closed loop control circuits			
+	E	Analog and binary signal conditioning			
+	F	Indirect measuring circuits (gated, corrected, suppressed, c			
+	G	Electrical equipment			
+	H	Subassemblies of main and heavy machinery			

Figure 8. Classification navigator for a mapping between KKS Equipment Unit keys and ecl@ss classes

The one-time library effort required is for an expert familiar with the classification standard and the to-be-classified part of MSL to assign a classification key to each MSL component. The assignment can be implemented in several ways:

1) As an external mapping table, where the mapping key is the classification key from the standard and the mapping value is a fully qualified MSL component name, including the path in the Modelica package hierarchy.

+ Nothing has to change in MSL.

- The table has to be maintained when new versions of MSL appear, where the classes have moved or have been renamed.

2) As an annotation that specifies the kind of category, (i.e. IEC 61346-2, KKS Equipment Unit, KKS Component) and the classification key that applies to the particular Modelica component, for building the

index. An annotation example for a Motor model:

```
annotation (category1="IEC61346-2=M");
annotation (category2="KKS_Component=-M");
```

There are many alternative ways to implement this using Modelica annotations. Investigations and practical evaluations are needed to find the best long-term sustainable approach.

+ The library developer has control over the classification.

- The library developer has to learn the classification standard and do the classification work.

At the moment of this writing, the interactive performance of the ModelicaDB implementation is still not adequate for launching a classification effort of MSL. The current object-oriented information model of ModelicaDB [11] contains 73 classes, 69 relationships and 170 attributes, which is more complicated than the one of the ecl@ss navigator which contains 13 tables, 5 relationships and 135 attributes. An efficient MSL classification effort for relevant international standards will probably also require fast interactive inspection of Modelica diagrams, which is currently not supported by the ModelicaDB front-end.

7 Using ModelicaXML as Exchange Format between Engineering Design Systems and Modelica Tools

ModelicaXML is a program that converts Modelica source code into XML-files [9]. A whole Modelica Library stored in a directory structure can be converted into one XML-file. Recent additions allows ModelicaXML to parse Modelica 2.2 source code [4]. The sizes of the created files for MSL version 1.5, 2.1, 2.2 and 2.2.1 are 16MB, 39 MB, 58 MB and 62 MB respectively. Their collected Modelica .mo source code files contained 1.0 MB, 2.9 MB, 4.8 MB and 5.3 MB.

ModelicaXML files were used as input for the analysis presented in Figure 6, and for loading the libraries into ModelicaDB for further analyses with SQL queries. The experiences with ModelicaXML are promising, and it could serve as a standardized vehicle to speed up development of working integrations between engineering design tools and Modelica simulation tools. Some benefits with the XML approach are:

- Easier to generate XML structures than syntactically correct Modelica source code.
- Easier to parse XML structures than Modelica source code.
- Commercial and open source software available for processing XML structures. See also [12].

- The Modelica community could share a lot of troublesome development effort if using ModelicaXML as a standard intermediate exchange format.
- ModelicaXML could serve as clipboard format for copy/paste, drag/drop of models between different tools. One drawback may be the size of the clipboard format when copying large models.

8 Summary and Conclusion

An investigation of international standards relevant for interfacing Modelica simulation modeling tools with engineering design tools that precede simulation modeling has been done and has briefly been described.

An analysis of the Modelica Standard Library contents has been conducted with aid of an upgraded version of ModelicaXML that supports Modelica 2.2.

A tool supported method for classification of MSL components has been developed and shown promising results for a simpler classification domain. Classification of components speed up mapping efforts significantly, since they efficiently narrow the search space.

The overall experience from this effort indicates that providing navigational access to Modelica components through classification trees organized according to well established international standards may significantly improve the take-up rate and learning curve for an expanding Modelica user community.

Acknowledgements

This work was supported by the Swedish Foundation for Strategic Research, ProViking project Systems Engineering and Computational Design (SECD), Swedish Governmental Agency for Innovation Systems (VINNOVA) in the project Semantic Web for Products (SWEBPROD), and Siemens Industrial Turbomachinery.

References

- [1] Dynasim, "Dymola", <http://www.dynasim.se/>.
- [2] INCOSE, "International Council on System Engineering", <http://www.incose.org>.
- [3] MathCore, "MathModelica", <http://www.mathcore.se/>.
- [4] Modelica Association, "Modelica: A Unified Object-Oriented Language for Physical Systems Modeling, - Language Specification version 2.2", February 2, 2005, <http://www.modelica.org>
- [5] Modelica Association, "Modelica Libraries", <http://www.modelica.org/library/>
- [6] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, 2003, <http://www.mathcore.com/drmodelica>.
- [7] Peter Fritzson, Peter Aronsson, Peter Bunus, Vadim Engelson, Levon Saldamli, Henrik Johansson and Andreas Karstöm, "The Open Source Modelica Project", in *Proceedings of The 2th International Modelica Conference*, March 18-19, 2002, Munich, Germany.
- [8] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In *Simulation News Europe*, 44/45, December 2005. See also: <http://www.ida.liu.se/projects/OpenModelica>
- [9] Adrian Pop, Peter Fritzson. "ModelicaXML: A Modelica XML representation with Applications", in *International Modelica Conference*, 3-4 November, 2003, Linköping, Sweden, <http://www.ida.liu.se/~adpo/modelica/>
- [10] Olof Johansson, Adrian Pop, Peter Fritzson, "A functionality Coverage Analysis of Industrially used Ontology Languages", in *Model Driven Architecture: Foundations and Applications (MDAFA)*, 2004, 10-11 June, 2004, Linköping, Sweden.
- [11] Olof Johansson, Adrian Pop, Peter Fritzson, "ModelicaDB - A Tool for Searching, Analysing, Cross-referencing and Checking of Modelica Libraries", in *Proc of 4th International Modelica Conference*, 8-10 March, 2005, Hamburg, Germany, <http://www.modelica.org/events/Conference2005>
- [12] Michael Tiller, "Implementation of a Generic Data Retrieval API for Modelica", in *Proc of 4th International Modelica Conference*, 8-10 March, 2005, Hamburg, Germany, <http://www.modelica.org/events/Conference2005>.
- [13] IEC, "IEC 61346: Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations", 1996, <http://webstore.iec.ch>
- [14] IEC, "IEC 61346-1: Part 1: Basic rules", First Edition 1996-03
- [15] IEC, "IEC 61346-2: Part 2: Classification of objects and codes for classes". First Edition 2000-04
- [16] IEC, "IEC 61346-3: Part 3: Application guidelines", First Edition 2001-01
- [17] IEC, "IEC 61346-4: Part 4: Discussion of concepts", First Edition 1998-01
- [18] IEC, "IEC 61355 Classification and designation of documents for plants, systems and equipment", First edition 1997-04
- [19] IEC, "IEC 61360 Standard data element types with associated classification scheme for electric compo-

- nents", <http://webstore.iec.ch>
- [20] ISO, "ISO 31 Quantities and Units, Part 0-13", <http://www.iso.org>
- [21] ecl@ss, "ecl@ss Organization", www.eclass.de
- [22] RosettaNet, <http://www.rosettanel.org>
- [23] RosettaNet, *RosettaNet Technical Dictionary*, <http://www.rosettanel.org/technicaldictionary>
- [24] VGB, "Verband der Großkessel-Besitzer", <http://www.vgb.org>
- [25] VGB, "B105 KKS Kraftwerk-Kennzeichensystem; Richtlinie zur Anwendung und Schlüsselteil"
- [26] VGB, "B106 KKS Anwendungs-Erläuterungen"
- [27] VGB, "B105e KKS Power Plant Classification System – Guidelines for Application and Key Part"
- [28] VGB, "B106e KKS Application Explanations"
- [29] DIN, "DIN 2481 Thermal Power Generating Plants, graphical symbols", Jun 1979
- [30] DIN, "DIN 1219 Fluid Power systems and components - Part 1: Graphic symbols for conventional use and data processing applications"
- [31] IEC, "IEC 61082 Preparation of documents used in electrotechnology"
- [32] IEC, "IEC 61082 - Part 1: General requirements", First edition 1991-12
- [33] IEC, "IEC 61082 - Part 2: Function-oriented diagrams", First edition 1993-12
- [34] IEC, "IEC 61082 - Part 3: Connection diagrams, tables and lists", First edition 1993-12
- [35] IEC, "IEC 61082 - Part 4: Location and installation documents", First edition 1996-02
- [36] IEC, "IEC 617, Graphical symbols for diagrams - part 1 to 13", issued 1983
- [37] IEC, "IEC 617 - Part 1: Conventions concerning electric and magnetic circuits. (issued 1985)
- [38] IEC, "IEC 617 - Part 2: Symbol elements, qualifying symbols and other symbols having general application"
- [39] IEC, "IEC 617 - Part 3: Conductors and connecting devices"
- [40] IEC, "IEC 617 - Part 4: Passive components"
- [41] IEC, "IEC 617 - Part 5: Semiconductors and electron tubes"
- [42] IEC, "IEC 617 - Part 6: Production and conversion of electrical energy"
- [43] IEC, "IEC 617 - Part 7: Switchgear, controlgear and protective devices"
- [44] IEC, "IEC 617 - Part 8: Measuring instruments, lamps and signalling devices"
- [45] IEC, "IEC 617 - Part 9: Telecommunications: Switching and peripheral equipment"
- [46] IEC, "IEC 617 - Part 10: Telecommunications: Transmission"
- [47] IEC, "IEC 617 - Part 11: Architectural and topographical installation plans and diagram"
- [48] IEC, "IEC 617 - Part 12: Binary logic elements (issued 1991)"
- [49] IEC, "IEC 617 - Part 13: Analogue elements (issued 1993)"
- [50] IEC, "IEC 60617 Graphical Symbols for Diagrams - online database", ~ 1850 symbols and 350 application notes in an on-line database (<http://dom2.iec.ch/iec60617>)
- [51] IEC, "IEC 61131-3 Programmable controllers - Part 3: Programming languages", Second edition 2003-01
- [52] ISO, "ISO 14617 Graphical symbols for diagrams - Part 1-15", First edition 2002-09-01
- [53] ISO, "ISO 14617-1 Part 1: General information and indexes", Second edition 2005-07-15
- [54] ISO, "ISO 14617-2 Part 2: Symbols having general application"
- [55] ISO, "ISO 14617-3 Part 3: Connections and related devices"
- [56] ISO, "ISO 14617-4 Part 4: Actuators and related devices"
- [57] ISO, "ISO 14617-5 Part 5: Measurement and control components"
- [58] ISO, "ISO 14617-6 Part 6: Measurement and control functions"
- [59] ISO, "ISO 14617-7 Part 7: Basic mechanical components"
- [60] ISO, "ISO 14617-8 Part 8: Valves and dampers", Corrected version 2003-12-01
- [61] ISO, "ISO 14617-9 Part 9: Pumps, compressors and fans"
- [62] ISO, "ISO 14617-10 Part 10: Fluid power converters"
- [63] ISO, "ISO 14617-11 Part 11: Devices for heat transfer and heat engines"
- [64] ISO, "ISO 14617-12 Part 12: Devices for separating, purification and mixing"
- [65] ISO, "ISO 14617-13 Part 13: Devices for material processing", First edition 2004-11-15
- [66] ISO, "ISO 14617-14 Part 14: Devices for transport and handling of material", First edition 2004-11-15

- [67] ISO, "ISO 14617-15 Part 15: Symbols for use on installation diagrams and network maps", First edition 2002-11-01

On the Noise Modelling and Simulation

Dorel Aiordachioaie Viorel Nicolau Mihai Munteanu Gabriel Sirbu
 “Dunarea de Jos” University of Galati, Electrical and Electronics Engineering Faculty
 Domneasca-47, Galati – 800008, ROMANIA
 Emails: {Dorel.Aiordachioaie, Viorel.Nicolau, Mihai.Munteanu, Gabriel.Sirbu}@gal.ro

Abstract

Physical modeling is referred to as the first representation of a process model and is represented as a set of differential and algebraic equations. Noise added to the model can improve the estimated behavior of the process and it is more close to reality. The work defines the metamodeling models in order to build tools for the management of the noise, i.e. generation and properly use of it. Some models for random sequences and signals are considered also, under various distribution functions. The way of adding noise to dynamic model of the simulated process depends on the structural model of the considered process, i.e. with noise at the output or with noise at the input of the model.

Keywords: Process Modeling, Noise Modeling, Metamodeling.

By applying the first modeling principles, a set of equations are obtained, which could be organized in two subsets: a subset of differential equations describing the dynamics of the process and a subset of algebraic equations describing the outputs and the constraints of the behavior of the considered process. In the context of real experiments and/or simulations of physical systems, where measurements should be considered as well for the purpose of identification and parameter estimation, the model is improved with noise information.

The reason to introduce noise in process's model is mainly related to un-modeled dynamics and disturbances acting on the process. Considering noise, the state-equations of the model of the process could have the form

$$\mathbf{E} \cdot \dot{\mathbf{x}}(t) + \mathbf{F} \cdot \mathbf{x}(t) = \mathbf{B}_u \cdot \mathbf{u}(t) + \mathbf{B}_w \cdot \mathbf{w}(t) \quad (1.a)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{e}(t) \quad (1.b)$$

1 Introduction

Physical based and object-oriented modeling languages offer an interesting and useful approach in process modeling and simulation, very appreciated and useful in the world of engineers and scientists. Examples of such software-based environments are Omola, Dymola or MathModelica. All these environments are connected with basic features of the Modelica modeling language, as a neutral representation of physical processes. More, based on object-matching features, it is used as the standard representation formalism over the distributed simulation platforms.

Perhaps the first reference that emphasizes a strong call to new modeling principles is of [1], which clearly shows the constraints of pure mathematical models. Other examples could be of [2] and [3], the last one - a project under the resources of Foundation for Strategic Research of Sweden.

where a noise component, $\mathbf{w}(t)$, is added for state variables and a noise component $\mathbf{e}(t)$ is added for the output variables. The type of the noise regarding the power and the probability density function depends on the process. Usually, white noise is considered with variance connected with the dynamics of the process.

Noise modeling is an important task in process modeling. There are unmodelled phenomena and unknown parameters. A noise model should describe how the unmeasured inputs and the unmodeled dynamics could change the behavior of the considered system. Noise modeling also stands for the addition of one or more noise components to state variables, in order to model disturbances and/or some random or unknown behavior. Naturally, the physics of the process should indicate which

variables should have noise and which one not, that could be done manually for simple processes. The problem has two aspects: first, there is a complex process, difficult to manage and, second, a software tool that is more efficient and comfortable for any modeler.

Adding noise to all equations can lead to derivatives of white noise and – as results – to non-causal process, as infinite values of some variables.

Details on how the non-causality with respect to the input signal, $u(t)$, can be handled are in [4] and [5]. The problem itself is considered and solved, however [6], where a band limited noise to avoid the problem suggest it.

The present work is looking to noise modeling and appropriate tools to generate different noise models. The reason of the subject is coming from the fact that Modelica, as far we know, does not have any considerations on noise models. The noise modeling tools are considered at the level of the metamodel, i.e. models of the methodology of noise modeling.

The object of the section 2 is related to methods of noise modeling. Section 3 contains the basic theoretical models to generate noise, looking to define the problem, to understand the method and to propose solutions related to the noise modeling. Section 4 is dedicated to a set of noise models, in Modelica language implementation. Simulation results are presented and discussed in section 5.

2 Noise metamodeling

Fig. 1 and 2 are looking to present the methodology of noise modeling in the context of physical modeling, i.e. the integration on noise models into physical process models.

The class diagram in Fig. 1 shows the hierarchy of different models, which is used in the building of the system model with physical constraints and – possible – under different representation formalisms. A model is an abstract representation and a generalization of a process model. A process model is an aggregation of one or more models based on equations. An equation-based model is an aggregation of some models, part with noise and part without noise. A noise model should be compliant with the laws of physics and the variables involved in the model should have a physical meaning. Symbolic tools are used to decide where and how to

add noise to the model of the process, in order to have a valid representation of the reality.

As Fig. 2 shows, a noise model is generalized as a model. A noise model is an aggregation of some model noises, e.g., of white noise model and band-limited noise, i.e. colored noise. The last one has constraints from a physical model concerning the parameters, e.g. the power of the noise and the frequency bandwidth.

Fig. 3 presents the point of view of the signal domain, which is the output of the noise models. The figure describes a hierarchy on classes, as the object-technology is supposed to have, starting with the class signal as generalization of the class noise. In turn, the class noise is an aggregation of three different noise classes, considering probability distribution function, starting with Wiener distribution and going to white noise and finally to colored noise. Each class has specific methods and attributes, e.g. the power of the white noise and the time constant or, equivalently, the frequency bandwidth, as parameter in the transfer function from white noise to colored noise signals. What is not considered here is the type of the signal, discrete or continuous. This is more complicated, mainly because it requires the interactions with a solver and is out of the paper's horizon.

3 Random variables

Random process generation is usually made in two steps: first, generating imitations of independent and identically distributed random variables having the uniform distribution over the interval $(0,1)$ and, second, applying transformations to these variables in order to generate random vectors with arbitrary distributions. These two steps are independent.

The next subsections sketch the theoretical background in order to sustain the declarative models for the generation of various types of the noise. Methods for generating a *sequence* of random numbers have been extensively studied and are well understood. Widely accepted is the method of the *linear congruential* generators. These numbers have the general form

$$U(k+1) = (a \cdot U(k) + c) \bmod m \quad (2)$$

where $U(k)$ is the k -th element of the sequence and $U(0), a, c$ and m are parameters.

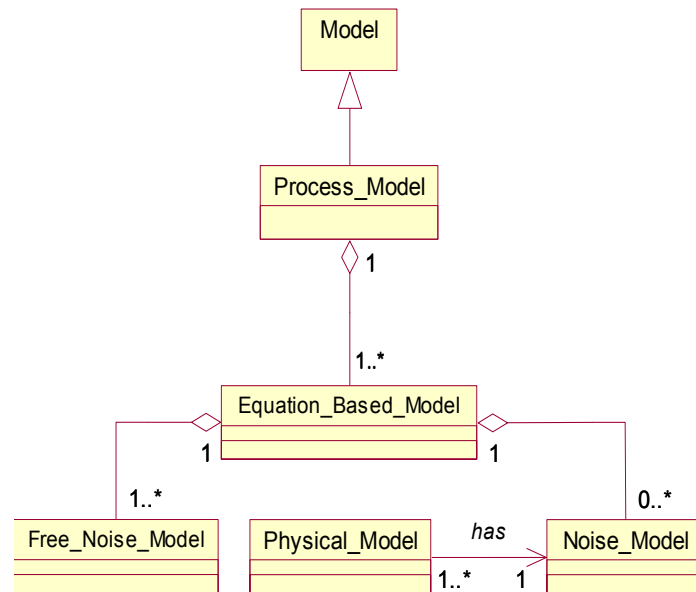


Figure 1: Different types of models

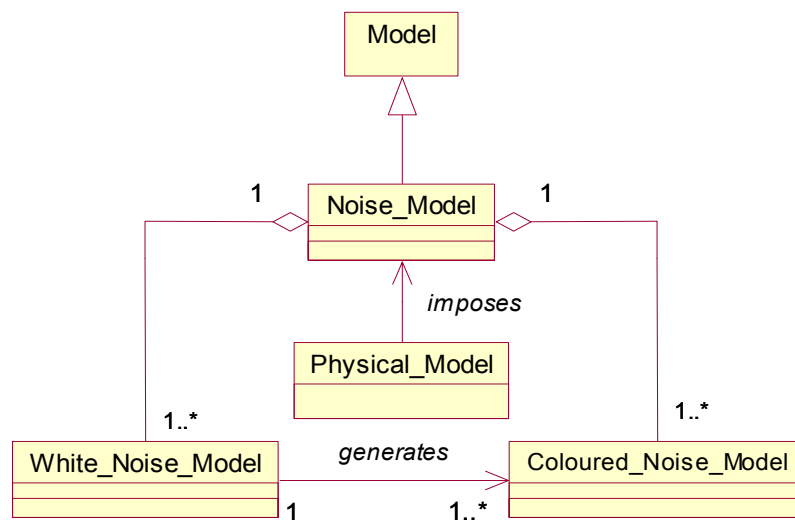


Figure 2: Connections among different types of noise models

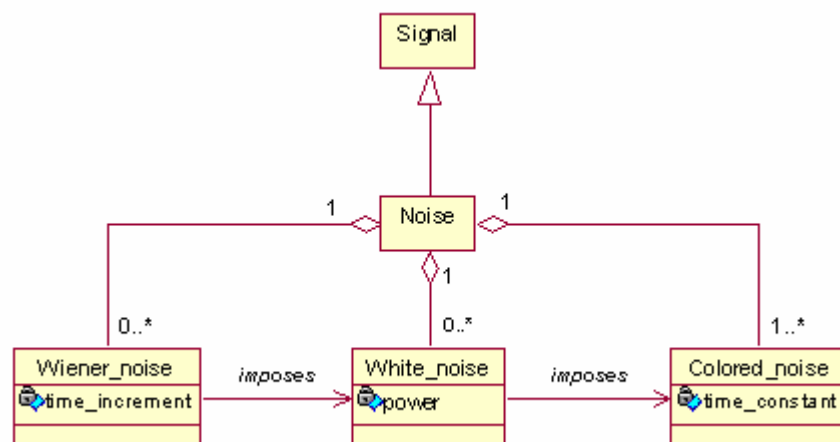


Figure 3: Class diagram of the noise signals

The second variable of the modulus function, m , is a large integer which must be prime; the multiplier a is an integer in the range $(2, 3, \dots, m-1)$; the additive constant c is an integer, often equal to zero. These are chosen to make the sequence look as random as possible. This generator has period $m-1$. Common choices for these values are $U(0) =$ any positive integer, $a = 16807$, $m = 2^{31} - 1$, $c = 0$. Other common choices and optimum algorithms are presented and described, e.g., in [13],[14],[15], [16] and [17].

A Box-Muller method [18] could be used in order to obtain a unit normal random, say X . Two uniform random variables U_1 and U_2 are necessary and the relation

$$X(k) = \sqrt{-2 \cdot \log(U_1(k))} \cdot \sin(2 \cdot \pi \cdot U_2(k)) \quad (3)$$

Given a uniform random variable $U(k)$, a Rayleigh random variable $R(k)$ can be obtained by:

$$R(k) = \sqrt{2 \cdot \sigma^2 \cdot \ln(1/1 - U(k))} \quad (4)$$

where σ^2 is the variance of the Rayleigh random variable.

4 Noise models

Considering an interval $I = [0, T]$, a standard Wiener process is a random variable $W(t)$ which satisfies the properties:

- 1). $W(0) = 0$
- 2). $W(t_2) - W(t_1) \cong \sqrt{t_2 - t_1} \cdot N(0,1)$, with $t_2 \geq t_1$, $\forall t_1, t_2 \in I$. $N(0,1)$ is a random number under normal distribution with zero mean and unit variance.
- 3). For $t_1 \leq t_2 \leq t_3 \leq t_4 \in I$, the samples $W(t_2 - t_1)$ and $W(t_4 - t_3)$ are independent.

White noise $N(t)$, with unit variance, is the formal derivative of a Wiener process $W(t)$, as

$$N(t) = \frac{dW(t)}{dt} \quad (5)$$

It may be possible that the noise in a physical system has correlations that are not satisfied by white noise.

A colored noise, $\xi(t)$, may be calculated from a stochastic equation as

$$\frac{d\xi(t)}{dt} = -\frac{1}{\tau} \cdot \xi(t) + \frac{\sigma}{\tau} \cdot N(t) \quad (6)$$

where $N(t)$ is a Gaussian white noise, with unit variance and zero mean.

5 Modelica implementation

Based on mathematical considerations of above section and on metamodel of Fig. 3, declarative models based on Modelica modeling language will be presented.

Real type numbers makes the interface communication over different models. The Modelica code for interfaces can be as

```
connector PortNumber
  Real n;
end PortNumber;
```

The uniform random number generator needs a function *mod* defined as

```
function mod
  input Real x, y;
  output Real z;
algorithm
  z := x - div(x, y) * y;
end mod;
```

where the assignation in the algorithm section imposes the causality of input and output variables.

A separate model describes global simulation parameters

```
model parameters
  // the numbers of random values:
  parameter Integer n=100;
  // sample period:
  parameter Real dt = 1;
  // start time moment:
  parameter Real start = 1;
  // the index of arrays used in simulation:
  Integer j;
algorithm
  j := integer(time/dt) + 1;
end parameters;
```

The uniform random generator over the interval $(0,1]$ is described by

```

model UNG //uniform_number_generator
  extends parameters;
  constant Integer m = 2^31 - 1;
  constant Integer a = 7^5;
  constant Integer c = 10;
  Real xmax, x[n];
  Integer j;
  PortNumber OUT;
algorithm
  x[1] := 1.0;
  for k in 1:n - 1 loop
    x[k + 1] := mod(a*x[k] + c, m);
  end for;
  xmax := max(x);
  for k in 1:n loop //normalization
    x[k] := x[k] / xmax;
  end for;
  OUT.n := x[j];
end UNG;

```

The model defines the basic bricks of the noise modeling and generations tools and has two outputs, related to two consecutive random numbers over a set of n preimposed values. All variables are part of the generator and the causalities are assigned in the algorithm section.

The normal distribution needs two uniform generators that could be developed by using two independent uniform generators or only one generator but with multiple independent outputs. With the last assumption the model of the normal random generator is as

```

model NNG //normal_number_generator
  extends UNG_MULTI;
  PortNumber OUT;
  Real x[n];
algorithm
  x[j] := OUT.n;
  OUT.n := sqrt(-2*log(OUT1.n))*sin(6.28*OUT2.n);
end NNG;

```

The model for a Rayleigh distribution is

```

model RRG // Rayleigh random generator
  extends parameters;
  parameter Real sigma = 1;
  Real xr;
  PortNumber OUT;
  UNG ung;
algorithm

```

```

  xr := sqrt(2*sigma^2*log(1/(1e-5+1-ung.OUT.n)));
  OUT.n := xr;
end RRG;

```

The normal white noise model needs a normal random numbers. The model is described by

```

model white_noise
  extends parameters;
  parameter Real sigma = 1;
  Real xw[n], xa[n];
  PortNumber OUT;
  NNG nng;
algorithm
  xw[j] := sqrt(dt)* nng.OUT.n;
  xa[j] := sigma*(xw[j] - xw[j-1]) / dt;
  OUT.n := xa[j];
end white_noise;

```

There are two random variables, one is Wiener and another one is of white type.

The colored noise is described by the sequence

```

model colored_noise
  extends parameters;
  PortNumber IN, OUT;
  parameter Real tau = 1;
  parameter Real sigma = 1;
  Real xc;
algorithm
  OUT.n := xc;
equation
  when sample(start, dt) then
    der(xc) = -xc/tau + sigma*IN.n/tau;
  end when;
end colored_noise;

```

and has two parameters with the names *tau* and *sigma*. The function **sample**(start,interval) returns true and triggers time events at time instants ($start + i*interval$).

The final simulation model, in order to generate a colored noise with the imposed parameters, is

```

model sim
  colored_noise cn;
  white_noise wn;
equation
  connect(wn.OUT, cn.IN);
end sim;

```

In addition, the links between the used models are graphically presented in Fig. 4.

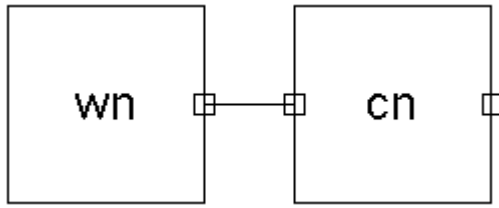


Figure 4: The model for colored white noise

6 Simulation results

For discrete sequences, simulations were conducted for different lengths. Figure 5 presents the result for $m = 2^{31} - 1, a = 7^5, c = 10, x(0) = 1$ as parameters of the uniform number generator. The parameters of the obtained random sequence are 0.4277 for mean and 0.0833 the variance. Raising the length to $n=100$ the distribution's parameters are improved to 0.4350 for the mean and 0.0995 for the variance. It is remembered that the ideal values are 0.5 for mean and 1/12.

Figure 6 shows a normal random sequence. The parameters for the second uniform number generator were $m = 2^{31} - 1, a = 7^3, c = 0, x(0) = 2$. For a length of 50 a sequence of 0.0939 and 1.0741 is obtained, as values for mean and variance. With $n=100$ the parameters are -0.0191 and 0.9885.

Fig. 7 shows samples of the simulation results from the outputs of the continuous noise models, i.e. white and coloured. It seems that the results are satisfactory over the behavior of the signals.

More statistic tests will be developed in order to improve the structure of generators and to check the distances between the real and the imposed behavior.

7 Conclusions

The objective of the work was to define noise models, with different density distribution functions, at the level of metamodels and to implement these in a neutral declarative modeling language, here Modelica.

The study is on the beginning and - as a first trial - the obtained models are compliant with the reference behavior. In the future more study will be done in order to make distinction between continuous time

and discrete time random processes and to build a set of models ready to use in noise modelling.

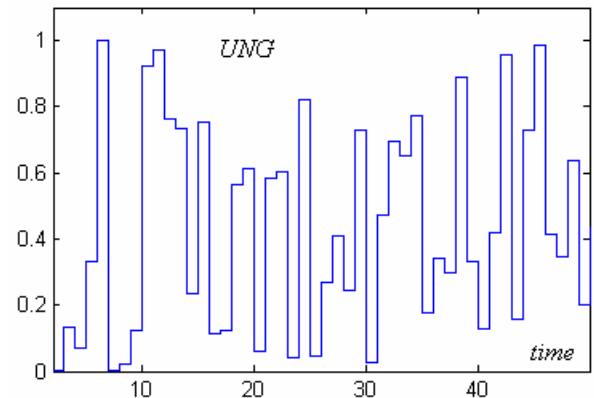


Figure 5: Uniform random number sequence

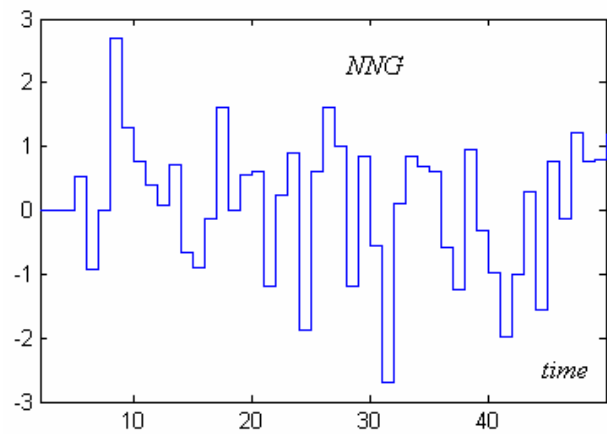
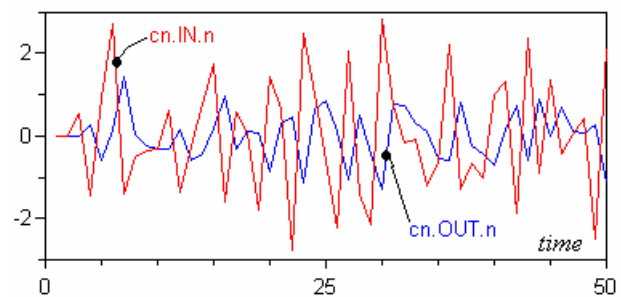


Figure 6: Normal random sequence

Figure 7: White and colored noise signal
($dt = 1, \sigma^2 = 1, \tau = 2$)

Acknowledgements

The authors would like to thank the Romanian National University Research Council for supporting part of the work, under the research grant 1350.

References

- [1] Cellier, F.E., *Continuous System Modeling*, Springer Verlag, New York, 1991.
- [2] Åström, K.A., Hilding Elmqvist, Sven Erik Mattsson, *Evolution Of Continuous-Time Modeling And Simulation*, 1998.
- [3] VISIMOD - A project funded by the Swedish Foundation for Strategic Research, <http://www.visimod.org/>.
- [4] Schein O., and G. Denk, Numerical solution of stochastic differential-algebraic equations with applications to transient noise simulation of microelectronic circuits, *Journal of Computational and Applied Mathematics*, 100, 1998, pp. 77-92.
- [5] Schon, T., Markus Gerdin, Torkel Glad and Fredrik Gustafsson, A Modeling and Filtering Framework for Linear Differential-Algebraic Equations, In *Proc. of the 42nd Conf. on Decision and Control*, Maui, Hawaii, USA, December 2003.
- [6] Campbell, S.L., Descriptor systems in the 90's. *Proceedings of the 29th Conf. on Decision and Control*, Honolulu, Hawaii, USA, Dec., 1990.
- [7] Tiller, M., *Introduction to Physical Modeling with Modelica*, Kluwer Academic Publisher, 2001.
- [8] Fritzson, P., *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, New York, 2004.
- [9] Modelica and Modelica Association, *Modelica Rationale*, <http://www.modelica.org/>, 2004.
- [10] Elmqvist, H., S.E. Mattsson and M. Otter, Modelica-A Language for Physical System Modeling, Visualization and Interaction, *Proc. of the 1999 IEEE Symp. on Computer-Aided Control System Design*, Hawaii, Aug., 1999.
- [11] The Global CAPE-OPEN Consortium, <http://www.global-cape-open.org>, 2000.
- [12] Braunschweig, B.L., Pantelidis, C.C., Britt, H.I., Sama S., Open Software Architecture for Process Modeling: Current Status and Future Perspectives. *Proc. of the FOCAPD'99 Conf.*, Breckenridge, Colorado, USA, 1999.
- [13] L'Ecuyer, P., Random number generation. In Banks, J., editor, *Handbook of Simulation*, Wiley, 1998, pp. 93-137.
- [14] L'Ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. *An object oriented random-number package with many long streams and substreams*, *Operations Research*, 50(6), 2002.
- [15] Knuth, D. E. *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, Mass., 3rd Ed., 1998
- [16] Kahaner, D., *Numerical Methods and Software*, Prentice Hall Series in Computational Mathematics, 1977.
- [17] Stephen K. Park, Keith W. Miller, *Random Number Generators: Good Ones are Hard to Find*, *Comm. of the ACM*, 31-10, 1988.
- [18] Sheldon M. Ross, *A First Course in Probability*, Prentice Hall College, 1997.

Acausal Modelling of Helicopter Dynamics for Automatic Flight Control Applications

Luca Viganò* and Gianantonio Magnani

Politecnico di Milano

Dipartimento di Elettronica ed Informazione (DEI)

Via Ponzio 34/5, 20133 Milano, Italy

e-mail: viganol, magnani@elet.polimi.it

Abstract

In the preliminary design stages of helicopter autopilots computationally affordable and mathematically simple dynamic models are needed to perform approximated performance assessments. In this paper, the structure of a modular, acausal and reconfigurable helicopter simulator is described, showing how the innovative characteristics of Modelica language can be employed in order to simplify the implementation of the single components and to allow the optimization of the simulator architecture. The overall model makes use of the existing Modelica MultiBody and Mechanics.Rotational libraries.

Keywords: helicopter; flight mechanics; simulation; AFCS

1 Introduction

The mathematical modelling of rotorcraft dynamics is a very difficult task that has represented a challenge for many researchers since age '60s. The availability of performant computers and a deeper theoretical knowledge in the late 80's dramatically improved the results achieved in this field. Nowadays, this research area is extremely wide, as it entails advanced studies in computational fluid dynamics (CFD) and flexible multi-body dynamics as well. However, Heffley et al. [11] and successively Padfield [3] underlined how, for handling analysis and for the design of common mid-low bandwidth automatic flight control systems (AFCS), the needed mathematical model of the plant shouldn't be too much complex (at least in the early stages of the project), because the experimental validation turns out to be much simpler with a reduced number of uncertain parameters and because the fundamental aerome-

chanical phenomena can be reasonably matched with simplified, first principle based, models. Furthermore, this kind of model is generally easier to use (requiring a relatively small set of parameters), computationally less expensive and then it's more suitable for real-time applications. This is the reason why a consistent number of researchers have devoted themselves in the last twenty years to the development of so-called minimum complexity helicopter math models (see [8], [11] for more details), which are able to correctly predict the prevailing phenomena involved in helicopter handling and control. The most recent result, in that sense, is given in Padfield's book [3], where the definition of Level 1 helicopter dynamic model is given.

In this paper, a Modelica implementation of a Level 1 helicopter dynamics model is presented, showing how the peculiar characteristics of Modelica language may be profitably used in order to make the implementation as natural as possible. The paper is organized as follows: in section 2 a brief overview of main rotor modelling techniques is given; in section 3 the Modelica implementation of the proposed helicopter model is described, followed in section 4 by simulation studies; at the end of the paper, concluding remarks and future developments are outlined.

2 Overview of basic main rotor dynamic modelling

In this paragraph, a synthetic description of a main rotor dynamic model suitable for flight mechanics simulation is provided, according to Level 1 model definition. The resulting mathematical model may be used not only for control synthesis purposes, but also for preliminary performance calculations. Main rotor model is without any doubt the most important and complex helicopter subsystem, being a fundamental

*corresponding author

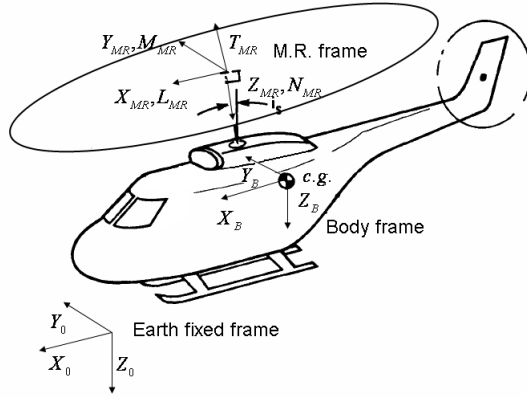


Figure 1: Helicopter reference frames [8]

source of lift and controllability for the aircraft, transferring prevalently aerodynamic forces and moments from the rotating parts (blades) to the non-rotating frame (that is the fuselage); pilot and AFCS have direct control of main rotor thrust amplitude through collective blade pitch and indirect control of thrust direction by means of cyclic blade pitch (because the flapping rotating blades act together as a gyroscope). The pitch of each blade is varied with particular mechanical devices (typically swashplate or “spider” assemblies, [1]). Since control-oriented helicopter simulators usually describe main rotor making use of analytically integrated loads, in order to provide deep understanding of the underlying physics and a significant simplicity, it’s evident that a so-called Level 1 main rotor model is more suitable for a general, modular and reconfigurable simulator than a complex CFD-based “numerical” model, where the aerodynamic loads are obtained integrating the pressure distribution upon each blade and, in the same way, the inertial loads are computed using complex codes for flexible multi-body systems [14]. This last approach becomes necessary when high bandwidth control actions (for example HHC - high-harmonic-control - techniques for the active control of vibrations [5]) or detailed analyses must be performed, as underlined in [15]. In the following, we shall provide a simplified description of the main rotor mathematical model used in our simulator: the interested reader can find much more details in the specialized literature [13], [17]. Let’s assume the following hypotheses:

- Compressibility, stall and reverse flow effects are neglected.

- The possible different blade flap retention arrangements (teetering, articulated and hingeless rotor) are described with the unifying centre-spring equivalent rotor theory.
- Fast lead-lag dynamics is neglected.
- The blades are considered rigid (in case of hingeless rotors, flexibility is concentrated in the centre-spring).
- Aerodynamic loads are computed using simple blade element theory.

Introducing the MBC (Multi-Blade Coordinates) transformation, one can describe flapping dynamics (which is fundamental in helicopter handling and control) with reference to the non-rotating frame. Representing with β_i the flapping angle of i -eth blade, with N_b the blades number and with $\psi = \Omega t$ the rotor azimuth angle in wind-axes system (rotor angular velocity Ω is assumed constant), MBC transformation is defined as:

$$\beta_0 = \frac{1}{N_b} \sum_{i=1}^{N_b} \beta_i \quad (1)$$

$$\beta_{0d} = \frac{1}{N_b} \sum_{i=1}^{N_b} \beta_i (-1)^i$$

$$\beta_{jc} = \frac{2}{N_b} \sum_{i=1}^{N_b} \beta_i \cos j \left[\psi + \frac{2\pi}{N_b} (i-1) \right]$$

$$\beta_{js} = \frac{2}{N_b} \sum_{i=1}^{N_b} \beta_i \sin j \left[\psi + \frac{2\pi}{N_b} (i-1) \right]$$

Neglecting periodic terms, which influences only vibrations, and the differential coning β_{0d} , which is reactionless and, in any case, null for N_b odd, the whole rotor disc configuration can be described using only the so-called coning mode β_0 and the first two cyclic modes β_{1c}, β_{1s} (representing longitudinal and lateral disc tilt angles, respectively) according to the following expression:

$$\beta(\psi, t) = \beta_0(t) + \beta_{1c}(t) \cos(\psi) + \beta_{1s}(t) \sin(\psi) \quad (2)$$

Using the vector representation $\bar{\beta} = \{\beta_0, \beta_{1c}, \beta_{1s}\}$, the flapping equations for the dynamics of a generic N_b -bladed rotor¹ can be expressed in the form [3],[12]:

¹For a two bladed teetering rotor, $\beta_0 = \text{const}$ must be assumed

$$\ddot{\bar{\beta}} + C_f \dot{\bar{\beta}} + D_f \bar{\beta} = H_f \quad (3)$$

The matrix C_f , D_f and the vector H_f are complex function of rotor system parameters (in particular, blade Lock number γ , representing the ratio between blade aerodynamic and inertial load, and equivalent spring stiffness K_β), flight conditions (mainly μ , advance ratio, representing the air velocity lying in rotor disc plane adimensionalized with respect to blade tip speed), blade pitch angle and aerodynamic inflow distribution. These last two contributions deserve more attention; in particular, blade pitch can be expressed in a way similar to (2)

$$\theta(\psi, t) = \theta_0(t) + \theta_{1c}(t) \cos(\psi) + \theta_{1s}(t) \sin(\psi) \quad (4)$$

where the three system inputs θ_0, θ_{1c} and θ_{1s} are, respectively, the collective blade pitch, the lateral cyclic pitch and the longitudinal cyclic pitch. The prediction of the aerodynamic inflow (that is the flowfield induced by the rotor at the rotor disc) is a really complex task, as it involves the dynamic description of a completely three-dimensional aerodynamic field. In flight mechanics applications, anyway, simple mathematical model are often used for this task, varying from the simple theoretical static uniform momentum theory to more complex dynamic wake models. In particular, all the dynamic models derived from the original work of Pitt and Peters [10] represent a good compromise between simplicity, physical consistency and correspondence with experimental flight data and they are suitable for flight mechanics and control applications. These models typically describe dynamic inflow with a three states approximation, in order to correctly predict the first harmonic distribution of induced velocity on the rotor disc in maneuvered flight:

$$\lambda(r, \psi, t) = \lambda_0(t) + \lambda_{1c}(t) \frac{r}{R} \cos(\psi) + \lambda_{1s}(t) \frac{r}{R} \sin(\psi) \quad (5)$$

where R is the rotor radius and the pair (r, ψ) uniquely represents the position of a point on the rotor disc. The equations for the inflow dynamics are commonly expressed in adimensional form as:

$$[M] \begin{Bmatrix} \dot{\lambda}_0 \\ \dot{\lambda}_{1s} \\ \dot{\lambda}_{1c} \end{Bmatrix} + [L]^{-1} \begin{Bmatrix} \lambda_0 \\ \lambda_{1s} \\ \lambda_{1c} \end{Bmatrix} = \begin{Bmatrix} C_T \\ C_L \\ C_M \end{Bmatrix} \quad (6)$$

The coefficients (C_T, C_L, C_M) represent the adimensionalized resultant aerodynamic thrust, rolling moment and pitching moment, respectively, in non-rotating frame. The expressions for the mass-apparent matrix M and the matrix L can be found, for example, in [9],[10]. In order to solve this complex mathematical problem, the expressions for the resultant forces and moments transmitted by the rotor to the fuselage need to be computed. Integrating analytically the aerodynamic and inertial loads acting on each blade and summing over all the blades, closed-form expressions for the rotor forces and moments X_w, Y_w, T, L_w, M_w, N can be derived; these formulas (here non reported for brevity, see [3],[8] for more detail) included vibratory terms, which are usually neglected in flight mechanics analyses, and quasi-steady terms, which are, on the contrary, of chief interest. The prefix w stands for “wind”, in order to remember that these components need to be transformed from the hub-wind system (aligned with relative air velocity) to the shaft-axes system taking into account the sideslip angle β_w , before assembling the overall system dynamics. From this analysis it’s clear that also the implementation of a basic complexity main rotor model in a traditional simulation environment may result a very difficult and time-consuming task, involving the solution of a fully coupled mathematical problem. In the next section, a Modelica implementation of a control-oriented helicopter dynamic model is described, showing the remarkable simplification allowed by the modelling paradigms offered by Modelica language.

3 A Modelica comprehensive control oriented helicopter simulator

In this section, a control-oriented helicopter dynamic model developed with Modelica language is presented. This model has been implemented employing the well-known analytical results published in [3],[8],[11] which constituted the starting point for a significant number of helicopter flight simulators, in military and research fields. The overall helicopter model takes advantage of the unique object-oriented features of Modelica language and it includes the following submodels:

- Atmosphere model
- Main rotor dynamics
- Tail rotor dynamics

- Airframe rigid mechanics
- Fuselage and empennages aerodynamics
- Turboshift engine simplified dynamics

The Atmosphere model is declared with **outer** keyword, in order to make it accessible from all the helicopter components; it implements a U.S. Standard Atmosphere model for the variation of air pressure, density and temperature with altitude, and it allows to define a uniform wind disturbance. The model can be extended in order to include also a stochastic turbulence model (for example based on Dryden's turbulence spectrum [4]). Since all the helicopter aerodynamic components need the updated value of the density ρ and since the small variations of density from a point to another of the aircraft do not justify a "local" evaluation of this quantity with a function, we have decided to compute it once for the whole helicopter, making reference to the altitude of its center of mass. A significant advantage found using Modelica for this application regarded the computation of local airspeed at different points of the helicopter; as pointed out by Looye and Moorman [2] speaking about their Flight-Dynamics library, the local airspeed is given not only by the inertial velocity of the center of mass and by the wind components, but it's also influenced by aircraft angular velocities. Local airspeed at a generic point p of the helicopter can be expressed as:

$$\vec{V}_a(p) = \vec{V}(p) - \vec{V}_w(p) - \vec{V}_{dw}(p) \quad (7)$$

where \vec{V}_a is the local airspeed, \vec{V} is the local inertial velocity, \vec{V}_w is the local windspeed (in our case it's assumed to be uniform) and \vec{V}_{dw} represents the velocity of the airflow determined by downwash effects. Employing the **FixedTranslation** components of the MultiBody library [7], this problem is easily solved, since \vec{V} , derived from the MultiBody frame relative to the considered aerodynamic component, takes automatically into account the velocity transport effect due to rigid rotations. On the other side, this procedure avoids user to manually specify (possibly introducing implementation bugs) the transport laws of forces and moments from aerodynamic components (for example from the center of pressure of the different lifting surfaces or from the hub reference system of main and tail rotor) to the center of mass reference system, as commonly happens in flight simulators implemented with low-level languages (such as Fortran or C). Eventual downwash effects (as those due to main

and tail rotor inflow) have been included introducing a dedicated **inflow connector**, which is declared as following:

```
connector Inflow_in
"External velocity field with harmonic
distribution"
input Modelica.SIunits.Velocity v0
"Average inflow component";
input Modelica.SIunits.Velocity vs
"Sinusoidal inflow component";
input Modelica.SIunits.Velocity vc
"Cosinusoidal inflow component";
end Inflow_in;
```

The same holds for the dual connector `inflow_out`, but the inflow components are declared as `output`. The causal nature if this connector is obviously made necessary by the simple mathematical modelling of downwash effects carried out in flight mechanics. Thanks to the innovative features of Modelica language, rotor dynamics can be implemented in the most natural way, declaring the equations as found in technical reports and specialized books (as those summarized in the preceding section) without requiring any error-prone and time consuming by hand manipulation of the analytical expressions. For example, let's consider the tail rotor dynamics: tail rotor is simpler to describe than main rotor, because it has no cyclic inputs but only a collective pitch input; moreover, its high rotating speed makes the flapping and inflow dynamics so fast that they are usually neglected. Static inflow theory is therefore suitable for the computation of tail rotor induced velocity and thrust:

$$\lambda_i = \frac{V_i}{\Omega_{tr} R_{tr}} = \frac{C_T}{2\sqrt{\mu^2 + \left(-\frac{v}{\Omega_{tr} R_{tr}} - \lambda_i\right)^2}} \quad (8)$$

$$C_T = C_T(\lambda_i, \mu, \theta_{tr})$$

where $V_i, C_T, \Omega_{tr}, R_{tr}, \mu, v$ represent, respectively, tail rotor induced velocity, thrust coefficient (adimensionalized thrust), rotating speed, radius, advance ratio and transversal airspeed (directed as the helicopter y-axis); furthermore, θ_{tr} symbolizes the tail rotor pitch input. The expression (8) is an implicit equation, because the inflow depends on the thrust and the thrust depends on the inflow; using traditional, causal, simulation tools or low level languages, the user should solve this equation implementing by hand the code for a Newton's iterative scheme or (as suggested in [11]) introducing

cyclic pitch and t.r. collective pitch) are visible, while on the right the output signals correspond to measured Euler/Cardan angles, pitch rates and airspeeds. Light thick connections represent aerodynamic interferences (downwash effects). Eventual primary mixer (for the decoupling of pitch and roll control channels) and interlink subsystems can be easily added to this basic simulator. Furthermore, specialized graphical shapes have been employed for the 3D visualization of the helicopter trajectories (Figure 3).

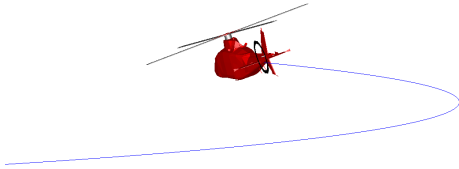


Figure 3: A109 helicopter performing a steady coordinated turn

4 Simulation study

As a case study, the technical data of an *Agusta A109mkII* helicopter have been considered [11]. The complete helicopter system is fully specified in our simulator inserting only 69 parameters (excepting the coefficients of look-up-tables). In the following, the results deriving from two different simulation studies are reported.

4.1 Trim analysis

In this test, the helicopter model has been trimmed for different values of airspeed in both longitudinal and lateral flight conditions. Figures 4,5 report plots showing trim data values for angular attitudes, command inputs and required power for different trim conditions. These results have been compared with the real flight data of the same helicopter model [6],[11], observing generally a very good agreement. Only a little overestimation of tail rotor authority has been observed for high speed flight, due probably to complex aerodynamic interactions between main rotor wake and tail rotor flowfield.

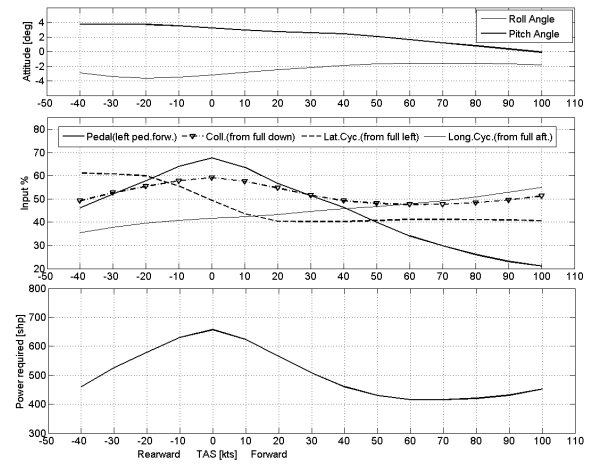


Figure 4: Longitudinal flight - trim data

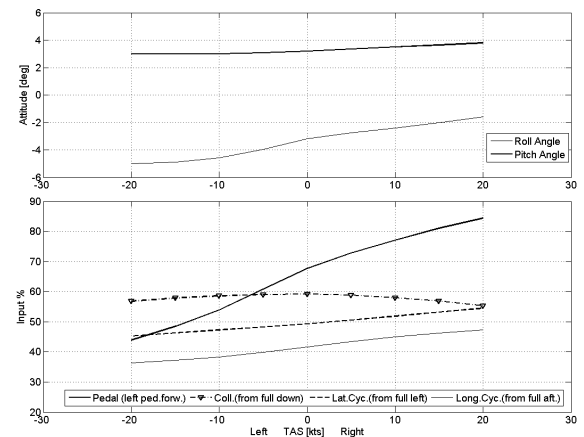


Figure 5: Lateral flight - trim data

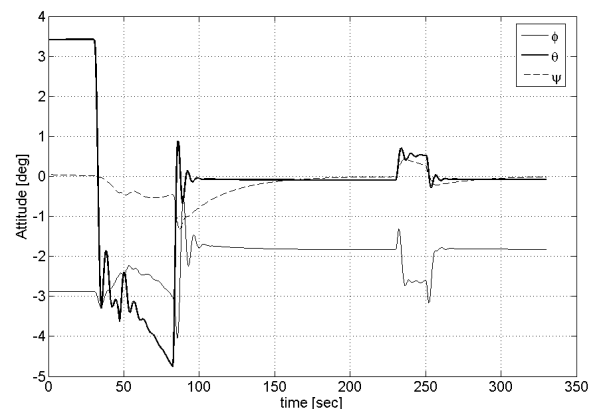


Figure 6: Helicopter attitudes

4.2 Piloted flight

Introducing a very simple AFCS control structure [3], we could easily reproduce the presence of a standard autostabilisation equipment plus a pilot (or an autopilot alone), in order to perform realistic transients. As an illustrative example, a simulation of a flight transient is reported: the helicopter starts hovering, then, for $t = 30$ sec the helicopter begins a longitudinal flight (sideslip $\beta_w = 0$) accelerating up to 100 knots (51.4 m/s, Figure 7). At $t = 230$ sec the helicopter starts a climb and it increases its altitude of 60m in 20 sec. The time history of three attitude angles is reported in Figure 6; the velocity increase from hover to 100 knots is achieved putting the helicopter nose down (the angle θ decreases) by means of a forward longitudinal stick displacement, as shown in Figure 8, where the longitudinal cyclic pitch reaches its steady-state trim value for 100 kts.

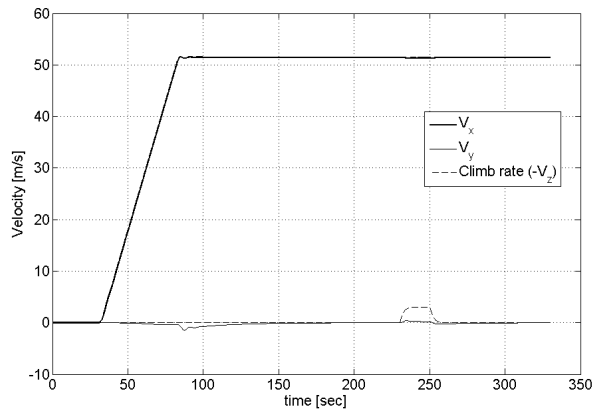


Figure 7: Helicopter velocities in earth-fixed frame

Correspondingly, the virtual pilot uses pedals (tail rotor pitch) in order to hold heading during the transient, compensating main rotor torque. At $t = 230$ sec the collective is pulled up in order to impress a positive climb rate of about 3 m/s to the helicopter, until the new altitude is reached. Figure 9 reports the time history of the dimensional inflow states: as speed increases, the main rotor wake passes from a configuration where the inflow is approximately uniform ($v_s = v_c = 0$) to a harmonic distribution consistent with high speed forward flight. Finally, Figure 10 shows the required engine power during the transient.

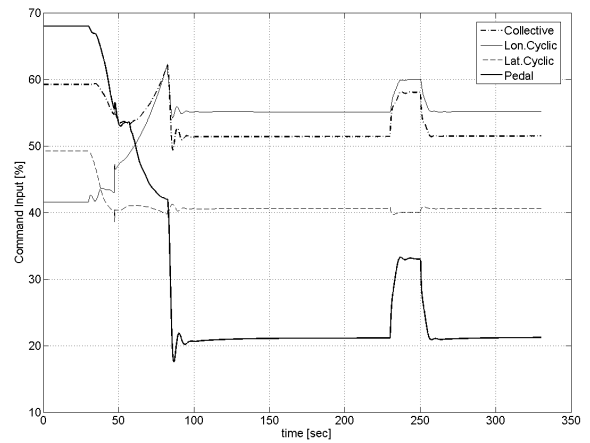


Figure 8: Helicopter inputs

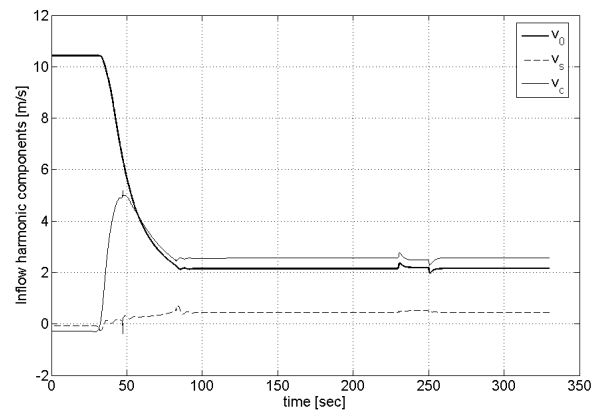


Figure 9: Main rotor inflow - harmonic components

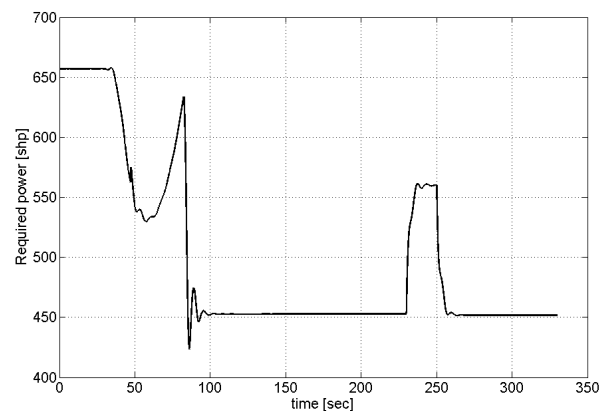


Figure 10: Required power

5 Conclusions and future work

In this paper, a simple parametric and reconfigurable helicopter dynamic model has been proposed, underlining the great advantages in terms of code readability and reusability deriving from the use of Modelica language paradigms and Dymola environment. As next work, the model validation will be performed. The validated model will be integrated with detailed electrohydraulics actuator models (under development) and realistic redundant FCC (Flight Control Computer) architectures. The resulting simulator will be used for the assessment of performance degradation in case of actuators failures and for the support in analysis and design of new specific linear and nonlinear control algorithms.

References

- [1] D. Balmford A.R.S. Bramwell, G. Done. *Bramwell's Helicopter Dynamics*. Butterworth - Heinemann, 2001.
- [2] D.Moorman and G.Looye. The Modelica Flight Dynamics Library. In *2nd Modelica Conference*, DLR, Oberpfaffenhofen, Germany, March 18-19, 2002.
- [3] G.D.Padfield. *Helicopter flight dynamics : the theory and application of flying qualities and simulation modelling*. Oxford : Blackwell, 1996.
- [4] J.D.McMinn. Extension of a Kolmogorov atmospheric turbulence model for time-based simulation implementation. In *AIAA Guidance, Navigation, and Control Conference*, New Orleans, USA, August 1997.
- [5] P. Colaneri M. Lovera and R. Celi. Periodic analysis of Higher Harmonic Control techniques for helicopter vibration attenuation. In *American Control Conference*, Denver, Colorado, June 2003.
- [6] G. Bonaita M.M. Eshow, D. Orlandi and S. Barbieri. Results of an A109 Simulation Validation and Handling Qualities Study. Technical Report USAAVSCOM 88-A-002, Aeroflightdynamics Directorate (U.S. Army Research and Technology Activity), Agusta SpA and Italian Air Force/D.A.S.R.S.-R.S.V. , May 1989.
- [7] M. Otter, H. Elmqvist, and S.E. Mattsson. The New Modelica MultiBody Library. In *3rd Modelica Conference*, Linköping, Sweden, November 3-4, 2003.
- [8] W.A.Decker P.D.Talbot, B.E.Tinling and R.T.N.Chen. A mathematical model of a single main rotor helicopter for piloted simulation. Technical memorandum NASA 84281, NASA Ames Research Center, Moffett Field, California, September 1982.
- [9] D.M. Pitt and N. HaQuang. Dynamic Inflow for Practical Applications. *Journal of the American Helicopter Society*, 33:64–68, 1988.
- [10] D.M. Pitt and D.A. Peters. Theoretical prediction of dynamic-inflow derivatives. *Vertica*, 5(1):21–34, 1981.
- [11] R.K.Heffley and M.A.Mnich. Minimum-complexity helicopter simulation math model. Contractor report NASA 177476, Aeroflightdynamics Directorate, U.S. Army Research and Technology Activity (AVSCOM), April 1988.
- [12] R.T.N.Chen. Effect of primary rotor parameters on flapping dynamics. Technical paper NASA 1431, NASA Ames Research Center, Moffett Field, California, January 1980.
- [13] R.W.Prouty. *Helicopter Performance, Stability, and Control*. Malabar, FL: Krieger Publishing, 1990.
- [14] C. Theodore and R. Celi. Helicopter Flight Dynamic Simulation with Refined Aerodynamics and Flexible Blade Modelling. *Journal of Aircraft*, 39(4):577–586, 2002.
- [15] M.B. Tischler. Digital Control of Highly Augmented Combat Rotorcraft. Technical report 87-A-5, Aeroflightdynamics Directorate, U.S. Army Research and Technology Activity (AVSCOM), May 1987.
- [16] J.C. Wilson and R.E. Mineck. Wind-tunnel investigation on helicopter-rotor wake effects on three helicopter fuselage models. Technical memorandum NASA TM X-3185, NASA Langley, Hampton, Virginia, March 1975.
- [17] W.Johnson. *Helicopter Theory*. Dover Publications, New York, 1994.

Dynamic modeling and control of a 6 DOF parallel kinematics

M. Krabbes Ch. Meißner

Leipzig University of Applied Sciences

Institute of Process Information Technology and Control Systems

Wächterstraße 13, 04107 Leipzig, Germany

Abstract

An object-oriented modeling structure as utilized by Modelica is well suitable for the simulation of the dynamic behavior of parallel kinematic structures. Especially application of the simulation system *DYMOLA* based on this language enables an easy dynamics simulation of parallel kinematics up to creation of inverse models in the purpose of control. Based on the inverted simulation of closed loop behavior in connection with a real-time implementation new concepts of multi-axis control become feasibly. *Keywords: model inversion; inverse disturbance observer; motion control, parallel kinematic machine*

1 Introduction

Serial and parallel kinematic structures has been one of the first and best examples to explain the new quality of modeling and simulation, which is possible by means of the object oriented modeling language *MODELICA* and corresponding simulation tools like *DYMOLA*. The non explicitly solved description scheme simplifies the handling of such complex systems dramatically. However, the philosophy and architecture of Dymola permits also a change of the signal direction through complete closed loop systems. So in connection with its real-time abilities, Dymola extends his purpose from an analyzing tool by potentials in control design and code generation. This can be shown very impressively at an example of a so called Parallel Kinematics Machine (PKM). This hexapod is a 6 DOF movable mechanical system for handling or other machining with high structural stiffness and small dead load, because all drives are fixed with the machine frame (Fig. 1).

For the control of almost any multi-dimensionally actuated production machine the appropriate control architecture replaces within the control loop the part of the kinematic chain behind the respectively last

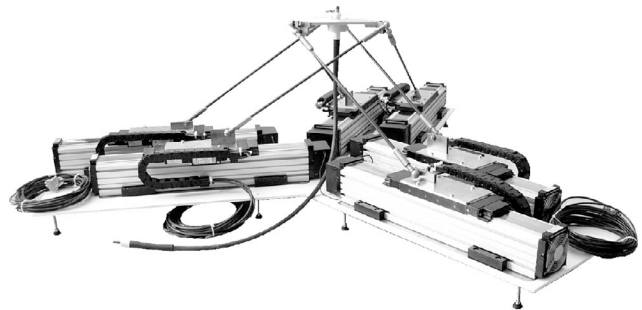


Figure 1: Parallel kinematic machine *Black Beetle*

drive position sensor by an appropriate static and/or dynamic model. Drive position sensors and based on it short control loops remain indispensable for high-dynamic performance. Only thus the electrical servo controllers can be dimensioned sufficiently rigidly, which are decentralized in subordinated SISO structures.

The model of kinematics connected to the control loops is assumed in particular with PKM normally as rigid and decoupled, so it considers only the corresponding static behavior. This simplifies substantially the necessary steps to its structural design, inverting and integration into the control system. Only the experimental identification of a machine-individual kinematics model with the necessary accuracy is further subject of scientific work and is referred as calibration. In contrast to this, development need exists for every of the mentioned steps concerning the dynamic effects of the open chain: starting at the model design over its inverting and control integration up to the experimental identification. Beyond that, the subordinated SISO drive control is to be maintained in face of extensions by centralized multi-axis controllers. Current publications are going to solve these problems by the mainstream concepts of multivariable control [3, 2]. Utilization of the simulation system Dymola represents a promising approach for a more tool based solution of these tasks by means of object oriented modeling based on Modelica. First results of investigations

to that effect are presented by this contribution.

2 Inverse Models for PKM control

The integration of a dynamic model into the control system of a PKM (as well as any other multi-axis kinematic structure) can be decomposed into two fundamental problem fields: on the one hand non-ideal and load-sensitive tracking behavior $\mathbf{G}_{drive} = \theta_{meas}/\theta_{ref} = f(\{\mathbf{F}; \tau\}_{ext})$ of the drive positions θ_{meas} does arise. The non-orthonormal action of the drives expresses itself in unavoidable, but directly *measurable* contouring errors. On the other hand also the elastic dynamics of the structure $\mathbf{G}_{elast} = \{\mathbf{X}_{meas}; \mathbf{I}_{meas}\} / \{\theta_{meas}; \{\mathbf{F}; \tau\}_{ext}\}$ lead to Cartesian position errors of the tool center point (TCP) \mathbf{X}_{meas} . These error portions are *only model-based assessable* and require imperatively the consideration of external perturbing loads $\{\mathbf{F}; \tau\}_{ext}$.

This contribution suggests a compensation of these effects in the described decomposition, whereby common structures and model prototypes are used. Appropriate inverse models of the closed loop system are used by means of the so called Inverse Disturbance Observer (IDOB), in order to produce a new reference signal with an error minimizing pilot control component [1, 7].

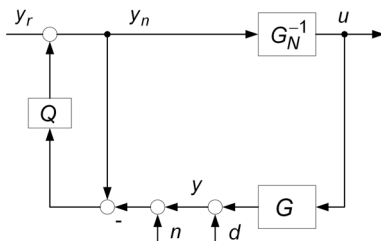


Figure 2: IDOB control scheme

The fundamental structure as pictured in figure 2 is based on a (nominal) inverse plant model and a feedback structure, which approaches within the bandwidth of the (unity gain low pass) filter Q the total behavior of the series connection of plant G and inverse model G_N^{-1} to 1. In the case of IDOB an upstream model-inverse \tilde{G}_N^{-1} produces accurate tracking, because the inverted model behavior is matched to the real plant. This correction effect makes it possible to work also at unstable plants with a stable approximated model-inverse [1].

According to the decomposition as introduced above, IDOB is used cascaded into two structures as in figure 3 based on inverted model components [4]. Within an

inside loop ideal tracking of the drives is effectuated by an inverting of the position controlled drives. By implementation of in this case rigidly assumed kinematics all changing inertia effects can be considered as well as influences of the coupling of the struts and external perturbing load. Hence, this model is quite accurately and permits a feedback filter Q_0 of high bandwidth for good performance.

For the tracking of the Cartesian position, the overall system is enclosed by a further IDOB. The ideal coordinates transformation is used here for the inverse model according to the rigid geometrical model. In order to close the control loop, a measuring signal of the TCP position is required, which is not actually present however. Therefore a further, partially inverted model is used for its estimation, which supplies apart from the TCP position also the external perturbing force and torque based on the measurable values of drive position θ_{drive} and drive load \mathbf{I}_{meas} . In this structure the outside loop is subject to various restrictions. On the one hand strong deviations between rigid model \mathbf{G}_{rigid}^{-1} and position controlled plant are possible. Therefore the outside filter can be dimensioned possibly only with small bandwidth. On the other hand the overall system works with the errors of the flexible model, because the structure depends on an approximated control variable.

Also the additional estimated signals are required for a safe total behavior, since theoretical stiffness values can be impressed, which would make excessive demands of the machine structure. Therefore reference inputs with defined, homogeneous compliance are to be produced by means of this load estimation.

3 Modeling of a DOF 6 Laboratory Machine

The novel PKM named *Black Beetle*, developed at HTWK – Leipzig University of Applied Sciences in cooperation with Fraunhofer Institute IWU [6], offers the possibility to test quickly new concepts in calibration, control or machining. It is build with six in one plane pairwise arranged linear drives, which are respectively joined with the tool mounting by struts of same length (Fig. 1 and 4). With this mounting a therein fixed tool or spindle can moved in 6 directions (DOF 6).

The extreme lightweight construction and the high process speed causes not negligible tracking errors and elastic deformations, which enforces a dynamic treatment in controller design. Realtime-simulations

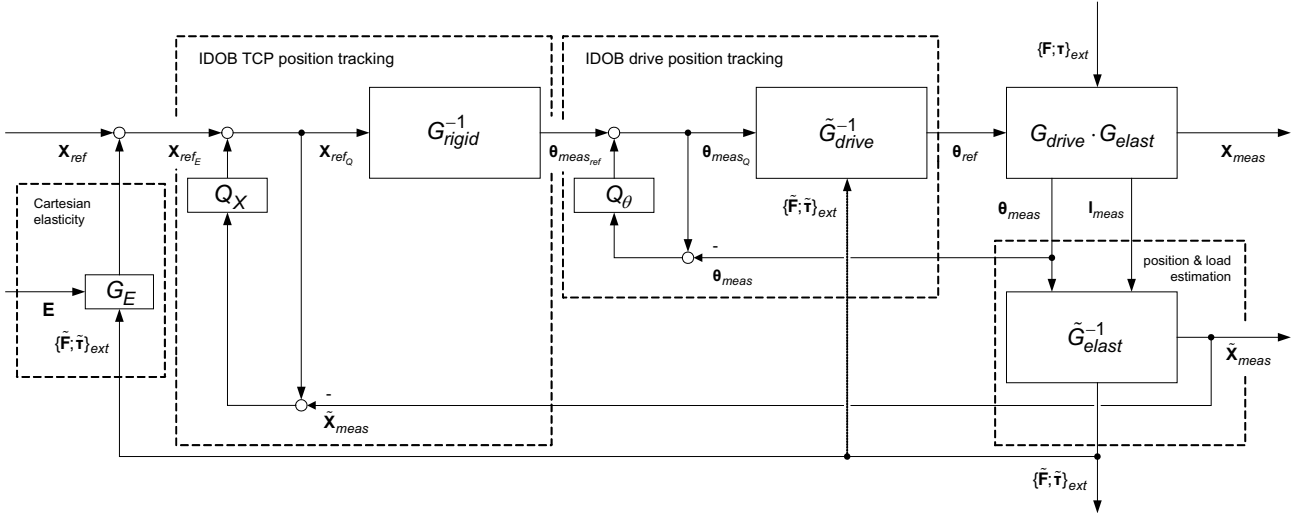


Figure 3: Cascaded control scheme of a PKM

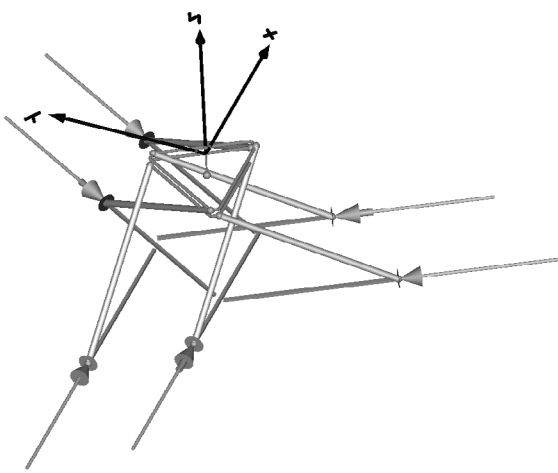


Figure 4: PKM model visualization.

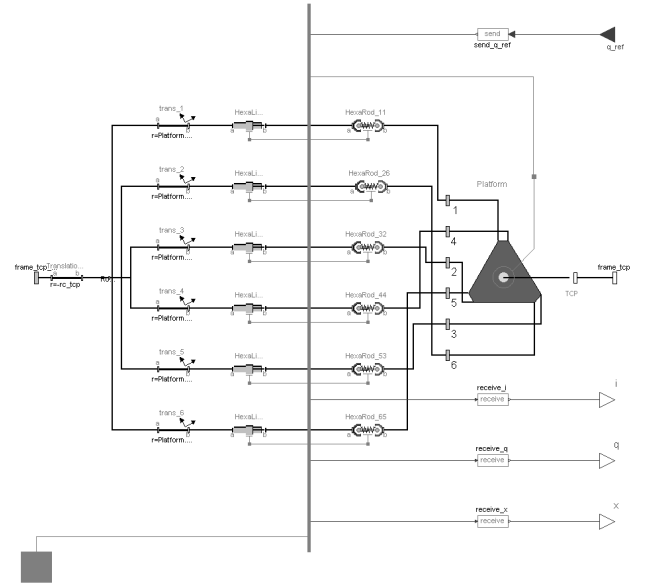


Figure 5: PKM overall model.

of such a system by means of Dymola offers now an easy way to do that. The overall model $G_{drive} \cdot G_{elast}$ (Fig. 5) is able to simulate the six linear drives including their friction effects and integrated controllers as well as the physical layout in a simplified structure of struts, which are represented as one-dimensional spring-damper-systems with mass-point. For this, many standard models of the Modelica library and a few own special models are used. Input signals of this element are the reference trajectories of the drive controllers and the perturbing load, output signals are the measurable drive position and force and the actual cartesian TCP position. The control structure has now to be completed with inverted and semi-inverted versions of the overall model.

4 Derivation of Inverse and Semi-Inverse Models

As described above the overall model has to be inverted in different variations. For this, we only have added some TwoInputs / TwoOutputs - blocks to the input and output connectors of the overall model, but nothing else (Fig. 6) The Modelica translator derives signal directions on it's own [4, 5].

In some cases these changes lead to problems because often derivatives of some expressions are necessary, which can't be differentiated e.g. in case of the direction dependent static friction. In respect of velocity v there is a not derivable discon-

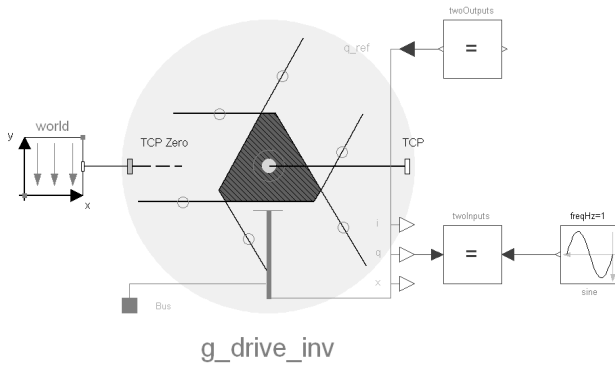


Figure 6: Inversion example of the overall model.

tinuity of Coulomb friction at $v = 0$ m/s which is problematic during model inversion. The solution is to filter the friction force signal. This filter is implemented as series of n first order filters with a overall characteristic of critical damping. Thus the force-signal can be differentiated n times (see `Modelica.Blocks.Continuous.CriticalDamping-Block`). Of course the signal is being smoothed and is shaped like a hysteresis, but this can be neglected with a adequately high cut-off filter frequency.

Three variants of the inverted model (Fig. 2) are utilized as follows. For use in outer IDOB a rigid version \mathbf{G}_{rigid}^{-1} is generated, which only transforms the coordinates from Cartesian space to axis configurations space. This system is based on a PKM model, where simply no dynamic effects will be mentioned.

The inner IDOB loop uses a more detailed inverse model $\tilde{\mathbf{G}}_{drive}^{-1}$ which is intended for drive position tracking. As described later, the whole drive internal controller structure, its friction effects and additional the coupling between the drives and therewith upcoming position dependent loads are mentioned.

Last but not least the TCP position and load estimation model $\tilde{\mathbf{G}}_{elast}^{-1}$ has been derived. Measured drive positions and currents enable this variant to estimate a actual TCP position and perturbing loads (force / torque). Unfortunately using current as drive load indicator is not appropriately. If the static (Coulomb) friction force is higher than the required drives force in hold-up position, the model is not able to determine the machine load, because the drive will not try to move in this case and no current is necessary to keep position. So it's recommended to use a more sensible measuring signal (of virtual sensors), but the concept remains the same. This inversion variant is primary used to register the elastic effects of the machine structure and to provide an in reality not or not easy to

measure TCP position signal. A secondary advantage is that this model can be used as source to apply homogeneous compliance at the TCP, which relies on this force / torque measurements. As consequence we can establish a respective sensitive tool.

5 IDOB drive position tracking

During the development of the complete DYMOLA implementation, realization of executable systems has shown to be a remarkable challenge. In a first step of development the inner IDOB loop had to be realized. As mentioned, the linear drives are working with decentralized encapsulated control by means of cascaded PI-velocity and P-position tracking. Caused by the drive couplings and a changing load, there is no optimal control configuration as for a single axis with fixed load. Hence, the aim is to improve the tracking performance, but only by varying the input signal of the drive controllers using a feedforward element. This is done as in the IDOB architecture by means of an inverse model. Because the model will never match the real linear drives behavior, a feedback loop generates an error minimizing signal.

While all models and their inverse derivatives could be realized quite easily, the multidimensional connection to the intended control structure in one Dymola-model overextended all available compilers. Only in single axis configurations the structure could be verified completely. In this example the performance was improved by multiple decades with a filter cut-off frequency of 1000 Hz. In this case the plant model and the inverted nominal model had very different friction parameters.

The connection of all IDOB elements succeeded only within Simulink by multiple import elements (Fig. 7). However, the performance of this implementation is affected by the required simple fixed step solver and the necessary input filters in all Dymola import elements in order to get the input signals with sufficient order of differentiability. With this environment the inner IDOB was tested by a spacial test trajectory (Fig. 8) and compared with the original control structure and only feedforward control by the inverse model (Fig. 9). As it can be seen from the absolute error values of one axis in figure 9 the tracking errors are reduced with the feedforward model up to 50 %, but with complete IDOB control the errors are lower than 10 % than before. However, on the other side there are considerable spikes, which are resulting from chattering of the non

	Original system	Feedforward control	IDOB control
No perturbative load	100 %	75 %	12 %
Perturbative load: 4 kg	101 %	70 %	11 %

Table 1: Relative mean error values.

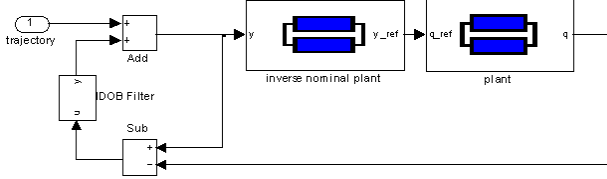


Figure 7: Simulink implementation of the inner IDOB loop.

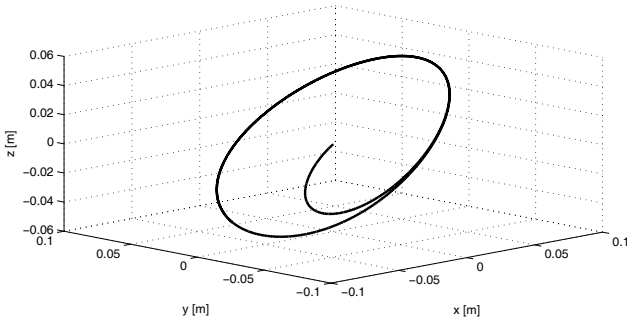


Figure 8: Spatial test trajectory.

ideal friction model in the plant model at velocity zero-crossing. They are also the reason for a quite limited cut-off frequency of the feedback filter at 50 Hz. Table 1 gives the relation of the summarized mean error values of all axes over the test trajectory with respect to the original system. Obviously, the control structure shows also robust behavior in case of added parameter differences between plant and inverse nominal model.

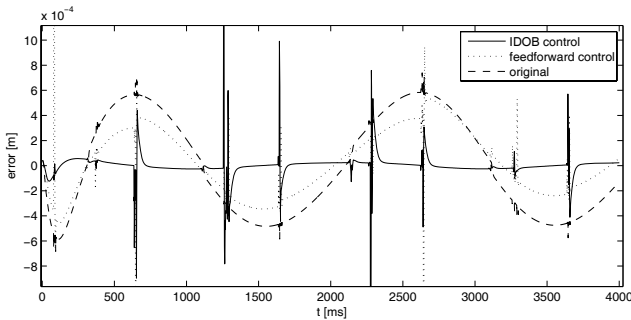


Figure 9: Comparison of tracking errors.

6 Real-time Environment

To control the laboratory machine by this approach, we need to involve the derived models in a real-time environment. In this case MATLAB-xPC-Target is chosen, because it is more easy to integrate a Dymola model into Simulink with xPC-Target real-time extension. In a capable model, which can be downloaded on a xPC-Target machine, the overall plant model $G_{drive} \cdot G_{elast}$ is replaced by connections to the drive controllers of the machine by a SERCOS-interface (signals on left and bottom side). To be able to use the SERCOS bus from xPC-Target, a appropriate Simulink block was created [6]. The integration of a Dymola model in a Simulink model is possible with the interface block provided by Dymola itself. With some options one is able to compile and link the Simulink model with the Dymola model translated by the Dymola translator for a PC-based target.

In order to calculate the complex cascaded control structure with its different models it is necessary to use a high performance target machine. Because of real-time purposes also here the model must be calculated with a fixed step solver. Hence with Dymola one have to chose the Euler algorithm (if necessary with inline integration). This fact implies that the calculation does not converge as good as a variable step solver would do. However, complete test of cascaded IDOB architecture is still in future work, where the potentials of real-time optimization have to be developed.

7 Conclusion

This paper presents first practical results of controlling a laboratory PKM by the approach of IDOB control. It could be shown, that on the way of generating real-time simulations Dymola is not only an analysis tool but moreover can be used for powerful control design and target code generation. While the main domain of such processing is multi-axis control, further improvements are required in the solution process of tasks with such high complexity.

References

- [1] N. Bajcinca and T. Bunte: A novel control structure for dynamic inversion and tracking. IFAC World Congress 2005.
- [2] C. Breche, T. Ostermann, D.A. Friedrich: Control concept for PKM considering the mechanical coupling between Actors. 5th Chemnitz Parallel Kinematics Seminar 2006.
- [3] B. Denkena and C. Holz: Advanced position and force control concepts for the Linear driven hexapod PaLiDA. 5th Chemnitz Parallel Kinematics Seminar 2006.
- [4] G. Looye, M. Thümmel, M. Kurze, M. Otter, J. Bals: Nonlinear inverse model for control. 4rd Int. Modelica Conference Hamburg-Harburg 2005.
- [5] M. Krabbes, Ch. Meißner: Dynamic modeling of a 6 DOF parallel kinematics by means of Modelica. 5th Chemnitz Parallel Kinematics Seminar 2006.
- [6] Th. Schröder, T. Otto-Adamczak, J. Müller M. Krabbes: A laboratory 6-DOF parallel kinematic for evaluation of new methods. 5th Chemnitz Parallel Kinematics Seminar 2006.
- [7] T. Umenco and Y. Hori: Robust speed control of DC servomotors using modern two degrees-of-freedom controller design IEEE Trans. Ind. Electron. 38(5), 363-368, 1991.

Modelling of Alternative Propulsion Concepts of Railway Vehicles

Holger Dittus Jörg Ungethüm
 Deutsches Zentrum für Luft- und Raumfahrt
 Pfaffenwaldring 38-40, 70569 Stuttgart
 holger.dittus@dlr.de joerg.ungethuem@dlr.de

Abstract

Hybrid power trains as increasingly used in road vehicles become more and more interesting for railroad vehicles. Short-distance passenger traffic on non-electrified lines is a domain where brake energy recuperation might reduce the total energy consumption significantly. In this paper a simulation model of a light diesel-powered railcar is presented. Model components are adapted from standard Modelica and Powertrain library. The potential improvement of fuel economy regarding different application settings is evaluated.

Keywords: model, simulation, railroad, power train, hybrid, energy consumption, drive strategy

1 Introduction

In contrast to road vehicles there were no compulsory emission limits for railroad vehicles in the past. In the domain of diesel driven railway vehicles there is only a voluntary emission limitation for nitric oxide and particles by the International Union of Railway (UIC), which is obligatory for its members. With the beginning of the year 2006 new tightened up restrictions by the European Union for new and repowered railcars take effect. Further decrease of emissions is settled for the year 2012. In this context effort is also spent on increasing the fuel efficiency of railcars.

In this work a model for dynamic simulation of diesel driven railroad vehicles is developed. This contains models for driving resistance, longitudinal section of the track, combustion engine, electric motor and generator, as well as gearboxes. Energy storages like flywheels and ultracapacitors are implemented for the simulation of hybrid vehicles.

The component models are combined to exemplary railcar power train configurations for local and regional traffic. The effect of different power train configurations and different driving strategies is demonstrated. Pure electric vehicles are not in the

scope of this work. However, the electric components might also be used to model those vehicles.

2 State of the Art

2.1 Railway vehicle power trains

In diesel railcars and locomotives hydromechanic or electric power transmissions are used. For light railcars most often the hydromechanic power transmission is used. The automatic transmission and the diesel engine are frequently adapted from road vehicle mass products. The electric power transmission consists of the diesel engine, which is rigidly coupled with an electric generator, and the electric traction motors. It is most often used in high speed railcars and heavy diesel locomotives. However, the electric power train can easily be transformed into a serial hybrid power train by adding an electric storage which makes it attractive even for short-distance railcars.

2.2 Comparison of commercial and rail vehicles

In commercial and public transport vehicles hybrid power trains are primarily used in two different applications. A significant number of busses for local urban traffic and light delivery vans, used in post and express service, are featured with alternative or hybrid power trains, predominantly in the USA. In most cases, serial or power split hybrid power trains are in use. In some cases also fuel cells or battery powered electric drives are in trial and in regular use.

Up to now, hybrid power trains in railroad vehicles are rare. However, some railroad manufacturer conducted experiments to use energy storages in railroad vehicles for braking energy recuperation. A current mass product is the “GreenGoat” which is a diesel-electric shunt locomotive built by the Canadian manufacturer Railpower Technology Corp. This lo-

comotive has a traction power of 2000kW and is featured with a 1200Ah lead battery.

Electric brakes for electric and diesel-electric railroad vehicles are well known. Braking energy is either converted into heat by brake resistors or it is fed back into the contact wire. Electric brakes are used as service brake because they are wear-free and non-exhaustible.

In the domain of short-distance passenger traffic on non-electrified lines light railcars with a tare weight between 23t (e.g. DWA LVT/S) and 120t (e.g. ALSTOM Corodia LIREX) are in use. A typical example is the RegioShuttle RS1 (former Adtranz, now Stadler) which has a maximum total weight of 56t. This railcar is the most commonly used diesel powered railcar in Germany. The power train of this railcar is based on a conventional road bus power train.

2.3 Driving cycles and driving styles

In a typical driving cycle a railroad vehicle is accelerated with the maximum tractive force up to the admissible maximum speed (acceleration phase). This speed is kept until short before the next halt (constant speed phase), where the vehicle is braked with the maximum service deceleration (deceleration phase). Changes of the admissible maximum speed are also realized with maximum acceleration or deceleration. This driving style leads to the shortest possible travelling time which is possible for a given vehicle.

The "energy saving" driving style uses the recovery margin of the timetable. The recovery margin is about 5 to 10% of the minimum travelling time and is considered in timetables to recover delays. It can be used to reduce the maximum speed or to join a roll out phase between constant speed and deceleration phase.

2.4 Driving strategies of serial hybrid power trains

The driving strategy of a serial hybrid power train determines how each component of the power train is driven. One category of driving strategies lets the power of the diesel engine follow the actual power of the electric traction motor. There is still a variation in the diesel engine rotating speed possible, which can be optimized to reduce total fuel consumption. In the classical diesel-electric power train without any energy storage this is the only possible driving strategy. Another category of driving strategies decouples the power of the diesel engine from the actual power of the traction motor. Obviously, an energy storage is required in this case. As the working conditions of

the diesel engine can be shifted towards its optimum a lower fuel consumption can be expected.

2.5 Energy storages

Railroad vehicles are commonly built for a lifetime of 25 to 30 years. The time between two general overhauls is 8 to 10 years. The huge number of charge/discharge cycles within this period would lead to high battery masses to obtain sufficient battery life-time, which makes batteries not yet suitable for railcar hybrid power trains.

Flywheels are used in railroad applications as stationary energy storage to buffer peak loads of contact wires. As flywheels are proved to be usable in mobile applications, e.g. road busses, they can also be used in railroad vehicles. Flywheels are nowadays built from fibre composite which enables very high rotation speed and reduces the impact in case of burst. Flywheels are best for energy storage in the time-range of several minutes.

Ultracapacitors are known for high power density and huge number of charge/discharge cycles. They are already applied in railroad applications for brake energy recovery (Bombardier MITRAC Energy Safer).

3 Modelling of the components

3.1 Driving resistance

The driving resistance of a railroad vehicle consists of the rolling resistance, climbing resistance, drag resistance, curved track resistance, and acceleration resistance.

$$F_W = F_{\text{Roll}} + F_{\text{Slope}} + F_{\text{Drag}} + F_{\text{Curve}} + F_a$$

The rolling resistance of a railroad vehicle is small compared to road vehicles. It is calculated from the total weight of the vehicle using the rolling resistance coefficient.

$$F_{\text{Roll}} = m_{\text{Vehicle}} \cdot \vec{g} \cdot f_R \cdot \cos(\alpha_{\text{Slope}})$$

$$f_R = 0.0015 \dots 0.0040$$

The climbing resistance is calculated from the mass of the vehicle and the ascent angle. For longer trains the mass of the train must be described as mass strap, where the ascent might vary between different parts of the train. In this work, only short railcars are considered, so the train is treated as a single mass point. It is common to specify the ascent in tenth of percent.

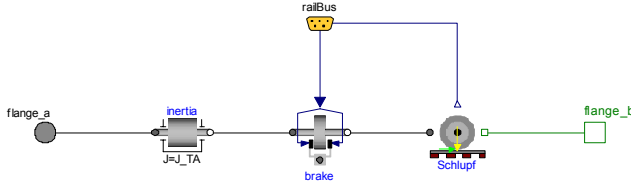


Figure 1: Model of the driven axle

$$F_{\text{Slope}} = m_{\text{Vehicle}} \cdot \vec{g} \cdot \sin(\alpha_{\text{Slope}})$$

$$= m_{\text{Vehicle}} \cdot \vec{g} \cdot \sin(\tan^{-1}(p))$$

The drag resistance is assumed to be proportional to the square of the driving velocity. For longer trains (esp. freight trains) the lateral air resistance of the wagons is dominant. For short railcars which are the focus of this work, the lateral air resistance is not calculated separately.

$$F_{\text{Drag}} = \frac{1}{2} \cdot \rho_{\text{Air}} \cdot c_w \cdot A \cdot v_{\text{Vehicle}}^2$$

The curved track rolling resistance is commonly calculated by an empiric formula. However, there are a number of different approaches. In this work, a simple formula, which is valid in average cases, is used:

$$F_{\text{Curve}} = \frac{0,75 \cdot m_{\text{Vehicle}} \cdot \vec{g}}{R_{\text{Curve}}}$$

The translatory acceleration resistance is calculated by Newton's law.

$$F_a = m_{\text{Vehicle}} \cdot a_{\text{Vehicle}}$$

The longitudinal section of the track including gradient, curvature data and admissible maximum speed is read from a data file.

3.2 Axles

The models of the axles are one of the main components. They are connected to the model of the chassis. There are models for driven and non-driven axles which are very similar. The models might be used for bogies, too. The model of the driven axle is shown in Figure 1. Its connectors are the abstractions of the driving shaft and the axle box which transmits the driving force to the model of the driving resistance. The component 'Schlupf' calculates friction and slip between the track and the wheel in the contact point. The actual slip is fed as input signal for the wheel sliding protection and the wheel skid protection.

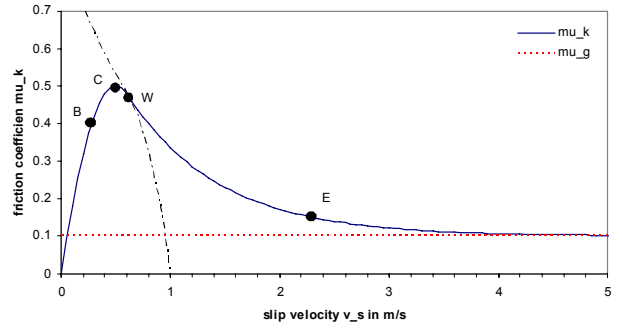


Figure 2: Approximation of friction coefficient as a function of slip velocity

3.3 Friction and slip between track and wheel

The small friction coefficient between track and steel wheel makes it necessary to calculate the slip, as acceleration and deceleration might be limited by the transmittable force in the contact point. Neglecting track gradient and vertical acceleration of the chassis, the perpendicular force is constant. The maximum friction coefficient is a function of vehicle speed and track condition. For the acceleration phase, the approach

$$\mu_{T,\max} = k_1 + \frac{k_2}{k_3 + v_{Fzg}}$$

is commonly used. The values of the constants were empirically investigated by Curtius and Kniffler. Even though modern vehicles do achieve higher friction coefficients, these values are still used in vehicle design to ensure sufficient friction even under worse conditions.

k_1	k_2	k_3
0,161	2,083 m/s	12,222 m/s

The actual value of the friction coefficient is a function of the actual slip velocity.

$$v_{\text{slip}} = R_{\text{wheel}} \cdot \omega_{\text{wheel}} - v_{\text{vehicle}}$$

The function $\mu_K(v_{\text{slip}})$ is built from a combination of a 2nd order polynomial and an exponential function (Figure 2). This curve is an approximation of three different friction mechanisms. In the range 0 to B the so-called microslip dominates. The friction coefficient is nearly linear to the slip velocity. Between B and the maximum friction coefficient at C a rapid alternation of minimal slipping and sticking (slip-stick-movement) dominates. For higher values of

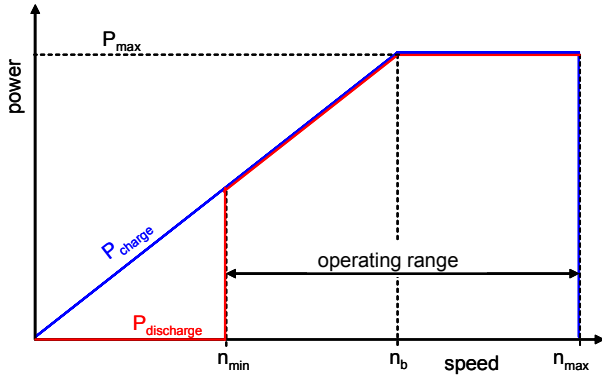


Figure 3: Flywheel operating characteristic

slip velocity, the friction coefficient decreases dramatically leading to a significant heating of the material in the contact point followed by further decrease of the friction coefficient. In most cases, measured values for μ_B and μ_W are not available. A reasonable approach is [1]:

$$\mu_K = \begin{cases} 2 \cdot \mu_C \cdot \frac{v_{slip}}{v_C} \cdot \left(1 - \frac{v_{slip}}{2v_C}\right) & \forall v_{slip} \leq v_W \\ D \cdot \exp(-\omega \cdot v_{slip}) + \mu_\infty & \forall v_{slip} > v_W \end{cases}$$

$$\omega = -\frac{T}{\mu_W - \mu_\infty} \quad T = \frac{2 \cdot \mu_C}{v_C} \left(1 - \frac{v_W}{v_C}\right)$$

$$D = (\mu_W - \mu_\infty) \cdot \exp(-\omega \cdot v_W)$$

$$\alpha = \frac{\mu_W}{\mu_C} \approx 0.95$$

$$\mu_\infty \approx 0.1$$

As input for the wheel sliding and wheel skid protection a variable slip is defined as the ratio:

$$\text{slip} = \frac{v_{slip}}{v_C}$$

3.4 Wheel skid and wheel slide protection

In this model, the actual slip is available as input signal for the skid and slide protection. This enables the implementation of a perfect slip and slide protection. In reality, in most cases only the wheel rotating speed is available. However, the internals of the skid and slide protection are beyond the scope of this work. For the aim of energy consumption prediction the approach of a perfect controlled system is sufficient. The skid protection reduces the driving power if the slip exceeds 0.8 until zero at slip 1.0. The slide protection works analogue in reducing the brake force.

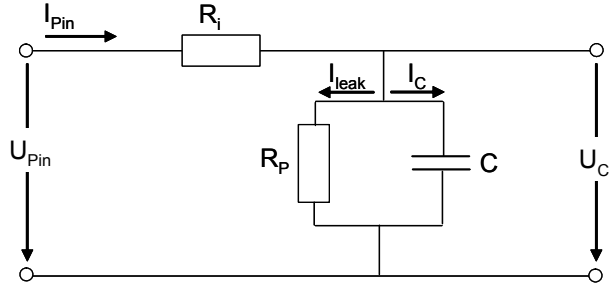


Figure 4: Equivalent electric circuit of the ultracapacitor

3.5 Internal combustion engine

The model of the internal combustion engine is a modified map-based model from the Modelica PowerTrain library. Apart from the stationary characteristic map the fuel consumption at idle speed is needed. As there is no specific data available an empirical formula is used. As a rule of thumb, a diesel engine needs in idle conditions 4mg of fuel per work cycle per 500 cm³ displacement. The engine friction is determined using a Willans curve methodology [3].

3.6 Flywheel

While the capacity of a flywheel is determined by the inertia and the maximum rotation speed of the wheel, the power is determined by the electric drive. As a result, a typical characteristic of a flywheel is shown in Figure 3. Major components of the model are a map-based electric motor and a standard Modelica inertia. Losses of the flywheel due to air friction, cooling and the vacuum pump are considered in sum as function of the rotating speed. To determinate the status of the flywheel, a state of energy is defined:

$$\text{SoE} = \frac{n^2 - n_{\min}^2}{n_{\max}^2 - n_{\min}^2}$$

3.7 Ultracapacitors

The model of the ultracapacitors is an implementation of the model by van Mierlo et al in [4]. The ultracapacitor is reduced to the equivalent circuit shown in Figure 4, which can easily be modelled using standard components. Corresponding to the flywheel a state of energy is defined:

$$\text{SoE} = \frac{U_c^2 - U_{\min}^2}{U_{\max}^2 - U_{\min}^2}$$

4 Control Strategies

The control strategy of the vehicle is divided into three layers. The first layer controls the higher-level functions of the vehicle independently from the power train components. It controls the velocity of the vehicle and communicates the desired acceleration or deceleration to the second and third control layers. The second layer manages the distribution of power and energy among the different power train and storage components. The third layer acts on the level of single components and controls their operation status depending on the desired power.

4.1 First control layer

The first control layer defines the way of driving implemented in the control strategy. The general aim of the control strategy is to achieve the shortest driving time. This means maximum acceleration until the speed limit is reached followed by a section driven with constant speed. Each change in the speed limit leads to maximum operational acceleration or deceleration. When approaching the next stop the vehicle is decelerated with the maximum operational braking force.

The first control layer uses three different states during an operational cycle. The first state is the driving state, which is used during acceleration and normal driving. In this state, the desired velocity given in the track description is controlled by a PI-Controller. When the vehicle reaches a certain distance to the next stop the state controller switches to the second state and the deceleration phase starts. This distance is dynamically calculated by means of the maximum operational deceleration and the current velocity of the vehicle. A PI-controller controls the braking signals for the power train components and, if needed, for the wheel set brakes. When the velocity reaches zero at the stopping point given in the track description the state controller switches to the third state. After a predefined stop-time the next operational cycle starts and the state controller automatically switches to the driving state.

The current state is provided to the other control systems via the signal bus of the vehicle. Therefore three Boolean signals are used which are called driving, braking and halt.

4.2 Second control layer

The second control layer determines the current electrical power of each component of the electrical drive train and storage system. In the diesel-hydraulic power train there is no choice which component has to deliver the required power. In this case the second control layer is not necessary; the power demand is transmitted directly to the controller of the internal combustion engine (DH).

In the case of the diesel-electric power train two operating strategies for the energy management are discussed. The first one (DE1) is based on the assumption that the internal combustion engine operates intermittently in its most efficient operating point [8]. If the SoE of the storage is greater 70 %, the ICE is switched off. As soon as the SoE-level falls short of 25 % the ICE is switched on again. The electrical energy is stored in the flywheel or delivered directly to the driving motors. The operation in this manner requires an energy storage with high power performance to ensure the delivery of the requested power when the ICE is stopped. On the other hand the power capability helps reducing fuel consumption due to the fact, that great amounts of braking energy can be recuperated.

The second strategy (DE2) assumes that the ICE runs permanently on a characteristic curve with low fuel consumption. In this case, the operating point of the engine depends on the current power demand of the first control layer. Whenever high power is needed, the engine runs faster and the generator demands more torque. In times of low power demand, engine speed decreases. The energy storage is primarily used to buffer braking energy; as a secondary effect it helps levelling out the engine dynamics.

4.3 Third control layer

The third control layer is implemented in each component. It controls the way how the requested power is distributed or stored, i.e. for the ICE the point or characteristic curve of operation. As this is very specific for the different components, it is not discussed any further here.

5 Simulation results

In the simulation three power train variants are compared. The first one is the model of the diesel-hydraulic (DH) power train. The others are diesel-electric power trains with identical component parameters. There is only a difference in the control

strategy. The main parameters of the railcars are given in the table below.

	DE	DH
Vehicle mass	54.2t	51.3t
Engine power	2 x 257kW	
Flywheel usable capacity	4.5kWh	-
Flywheel max. power	500kW	-

Figure 5 shows the structure of the DE-Model with two diesel-electric generators and two electrically driven axles which are mechanically connected to the driving resistance model. The DC voltage link connects the electrical components like flywheel, braking resistance and auxiliary equipment. The controllers manage the flow of energy within the electrical and mechanical system.

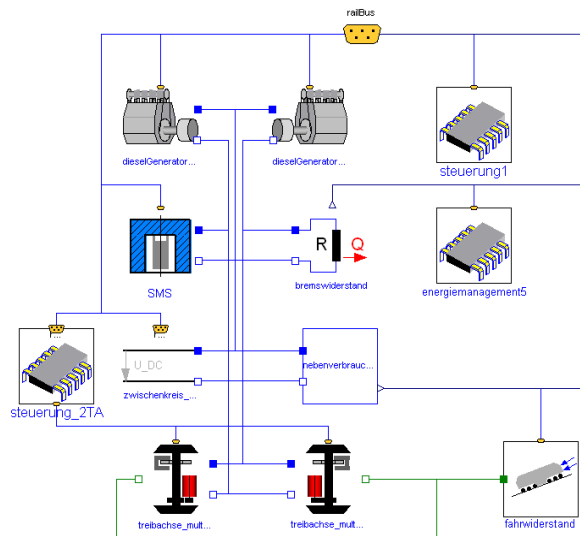


Figure 5: Model of the diesel-electric power train

The three different power train systems are simulated on a test track, which is a short sequence of a mountainous track with relatively low speed levels. The admissible maximum speed is 80km/h, the slope is up to 25%. There are three stops within the route. Figure 6 shows the track related resistance forces for the diesel-hydraulic railcar. The rolling resistance F_{Roll} is almost constant, while the drag resistance F_{Drag} is obviously varying with vehicle velocity. The slope resistance is the most significant force and is by far the dominant resistance force on parts of the track. Compared to the other resistance forces, the force F_{Curve} due to the curvature of the track is relatively small. Figure 7 shows the acceleration force F_a . As the railcar travels with maximum ac-

celeration, this force is temporarily up to ten times of the sum of the other forces.

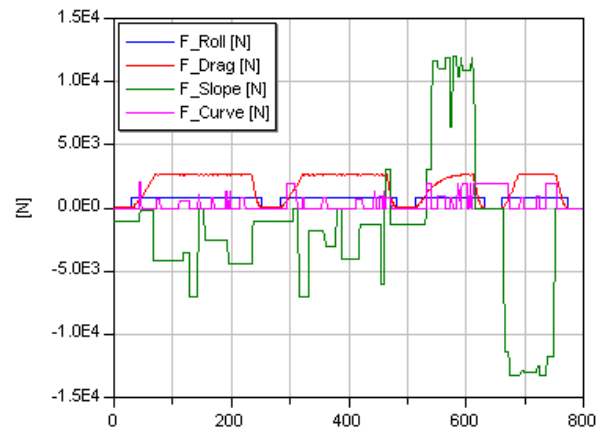


Figure 6: Resistance forces simulated with the DH-model

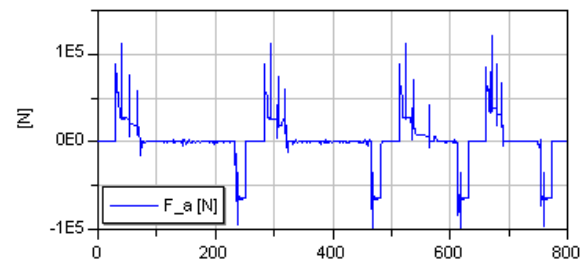


Figure 7: Acceleration force of the DH-model:

Figure 8 compares the vehicle speeds of the two diesel-electric variants and the diesel-hydraulic railcar. There are slight differences in the time needed to reach the last stop. The DH is the fastest, while the DE1-variant needs about 12 seconds longer to accomplish the track. At $t = 600$ s the velocity of the three variants differs significantly. Although the two DE-railcars are driven by electric motors with the same power, they don't achieve the same velocity. This is the result of the different control strategies. In the DE2 variant the power of the driving motors is limited by the controller of the energy management. This is done because the amount of energy in the flywheel is depleting even though the ICEs are running at full power. Eventually this reduction of motor power leads to the longer driving duration of this variant.

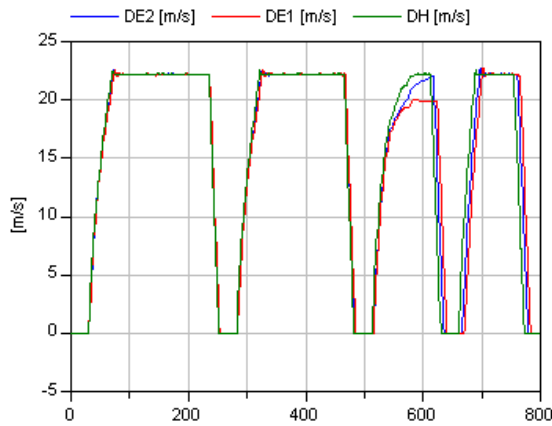


Figure 8: Comparison of the vehicle velocities

Figure 9 illustrates the state of energy SoE of the flywheel. Obviously the utilisation of the storage is much higher with the DE1-strategy. This leads to greater amounts of energy stored in and recovered from the flywheel. The capacity of the storage seems to be a good choice for the DE1 variant, while the DE2 variant does not necessarily need such a big storage. In this variant at least 25 % of the capacity is unused. While it is easily possible to determine a well-sized capacity for the DE2-variant, it is difficult to find the correct capacity of the DE1-variant. In the end, the capacity is responsible for the on/off-timing of the ICE. Especially if the ICE runs while the traction motors need power, the energy efficiency of the system rises. Instead of storing the electric energy in the flywheel with conversion losses, it is instantly used by the traction motors with higher overall efficiency. An optimisation of the storage capacity strongly depends on the driving cycle, the SoE at start and the on/off-timing of the ICE.

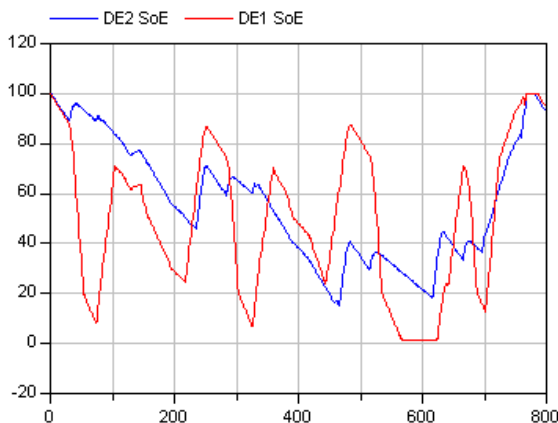


Figure 9: State of Energy of the flywheel

Differences between the DE-variants are also in the amount of power the flywheel must be capable of. Figure 10 shows the power curves for both variants.

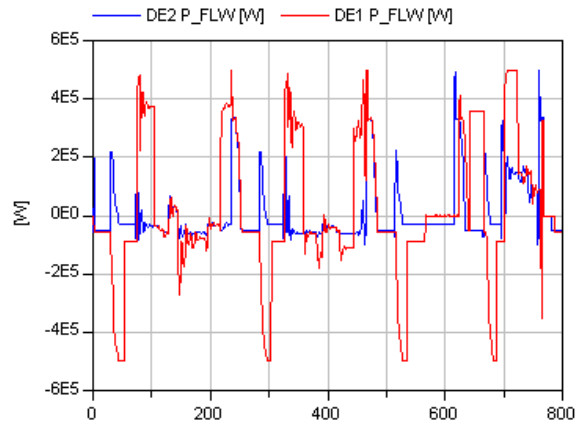


Figure 10: Power at the flywheel

While the DE1 variant needs a flywheel with power up to 500 kW, the DE2 variant could be equipped with a flywheel with less power. The DE2 strategy covers a great amount of the demanded power by the ICEs, the DE1 variant prefers the storage to cover the vehicles power requirements.

The energy flow within the electrical system of the DE-models is shown in Figure 11 and Figure 12. The energy flows of the generators and the traction motors are slightly different, but in the same dimension. The differences in the flywheels energy are remarkable. As already discussed with the state of energy curve, the DE1 variant stores plenty of energy in the flywheel. The figure shows that not only the recuperated braking energy is stored but also a great amount of the electrical energy produced by the diesel-generators.

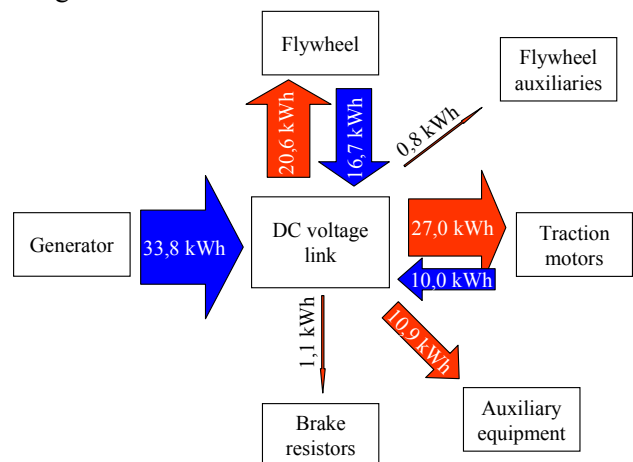


Figure 11: Electrical energy flow of DE1

The DE2 variant does not even store all the recuperated energy. Partly the braking energy is consumed by the auxiliary equipment, and some parts are transformed to heat in the brake resistors.

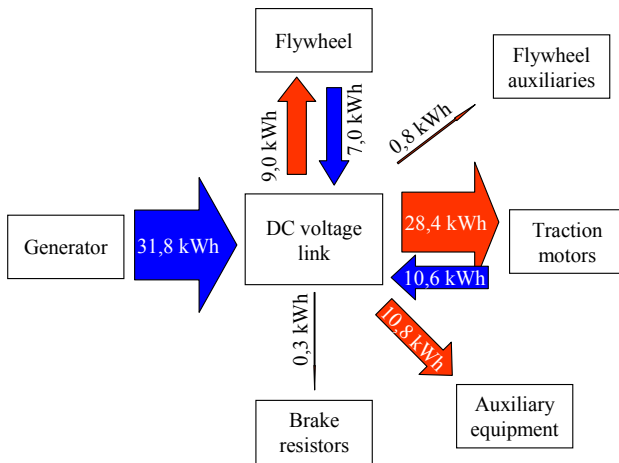


Figure 12: Electrical energy flow of DE2

The fuel consumption of the three variants is shown in Figure 13. The best energy efficiency is obtained with the diesel-electric variant DE1, which uses the intermittent control strategy for the ICE. The fuel consumptions of DE2 and DH are close together at the last stop. During driving there are partly strong differences between all strategies.

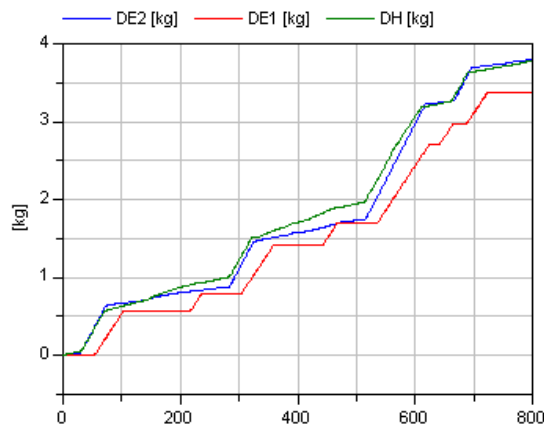


Figure 13: Fuel consumption of one diesel-engine

The comparison of the fuel consumption and the energy flows in the DC voltage link shows that DE1 consumes less fuel than DE2. This is especially remarkable because the amount of electrical energy produced by the generator is greater with DE1 than with DE2. This result proves the enhanced efficiency of the ICE as a consequence of the usage in its most efficient operating point. It has to be stated that these conclusions are specific for the chosen line charac-

teristics and vehicle parameters. Especially the dominance of the slope resistance requires customizing the control strategy by needs of the line topology.

6 Conclusions

The model presented in this work is a starting point of hybrid railcar simulation in Modelica. It turns out, that a number of models that were primary developed for automotive simulations can be adapted to railroad vehicles. Therefore the effort to set up the simulation is moderate. The simulation gives an idea of the energy saving potential of hybrid power trains even in railroad vehicles. However the shown serial hybrid is the most simple hybrid power train, parallel or power-split hybrids should be investigated in further. As the simulations with the different control strategies demonstrated, there is a strong influence of the control strategy on the achievable fuel consumption of a predefined vehicle.

References

- [1] Wende, D. Fahrdynamik des Schienenverkehrs. B.G. Teubner Verlag/GWV Fachverlage GmbH, Wiesbaden, 2003
- [2] Filipovic, Z. Elektrische Bahnen. Springer Verlag, Berlin, Heidelberg, New York, 1992
- [3] Kuhlmann, P. Grundlagen der Verbrennungsmotoren, lecture notes Universität der Bundeswehr, Hamburg, 1990
- [4] van Mierlo, J., van den Bossche, P., Maggetto, G. Models of energy sources for EV and HEV: fuel cells, batteries, ultracapacitors, flywheels and engine-generators. Elsevier Journal of Power Sources No. 128, 2004
- [5] Wallentowitz, H. Längsdynamik von Kraftfahrzeugen Schriftenreihe Automobiltechnik Forschungsgesellschaft Kraftfahrwesen Aachen mbH, 2002
- [6] Göhring, M. Betriebsstrategien für serielle Hybridantriebe Dissertation, RWTH Aachen, 1997

Modelling Automotive Hydraulic Systems using the Modelica ActuationHydraulics Library

Peter Harman
Ricardo UK Ltd.
Leamington Spa, UK
Peter.Harman@ricardo.com

Abstract

This paper describes applications in the automotive industry which require modelling and simulation of hydraulic systems and introduces a new library developed specifically for use in these applications.

Issues encountered whilst modelling hydraulic systems are discussed, from the numerical issues of simulating mixed hydraulic-mechanical systems to the more practical aspects such as the availability of parameter data for valve models.

A selection of case studies are described.

Keywords: hydraulics, automotive, power steering, transmission actuation, lubrication, braking systems, active 4wd

1 Introduction

An increasing number of automotive systems use some form of electro-hydraulic control, and with increasing levels of complexity in these devices simulation is an important part of the development process. Modelica provides an ideal environment for such simulation as it is necessary to include mechanical, electrical and hydraulic components in the same system, utilising Modelica's multi-domain capabilities.

2 Applications of Hydraulics Simulation in Automotive Industry

Key applications of hydraulics within automotive engineering are in the power steering, transmission, driveline and braking systems.

2.1 Power Assisted Steering

Power steering is standard on virtually every road car, racing car and off-highway vehicle, and in the majority of cases is a hydraulic system. A spool valve, either translational or rotational, translates the deflection of the steering wheel relative to the steering rack into a hydraulic flow into a piston to provide additional steering force. The dynamics of the hydraulic system affect the response of the vehicle to the driver input and therefore simulation is used to predict the effect of parameter changes such as the spool geometry.

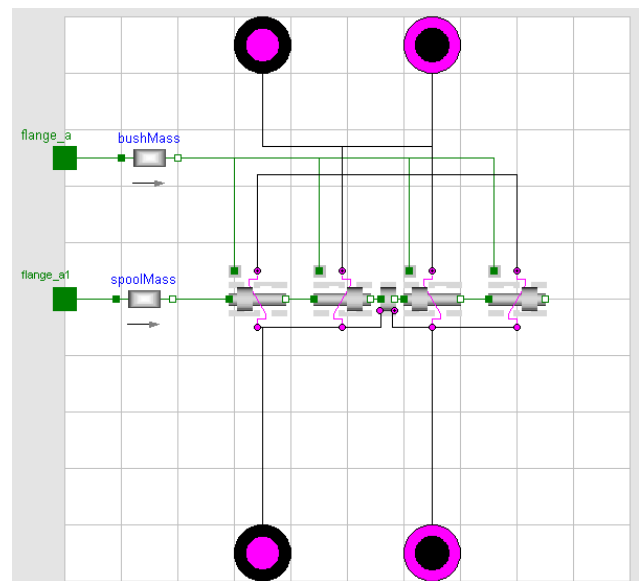


Figure 1: Power steering spool valve model

Figure 1 shows a Modelica model of a spool valve within a power steering system model. The spool model is built up of submodels, each representing one flow path through the valve. These “SpoolPort” models interface with translational mechanics components from the Modelica standard library, and the flow through each is a function of the spool displacement, the pressure drop and the port geometry.

This makes it easy to quickly build complex spool arrangements for this type of application.

2.2 Transmission Actuation and Lubrication

The trend in transmission technology is towards actuated transmissions such as Automated Manual Transmissions (AMT) and Dual-Clutch Transmissions (DCT or DSG), and these plus the continuation of development of conventional Automatic Transmissions and Continuously Variable Transmissions (CVT) lead to an overall growth in hydraulically actuated systems. Development of valve-blocks and the control system requires combined simulation of the hydraulics and the transmission dynamics [1]. These transmissions often also have lubrication systems combined for which simulation is required to determine flows at each outlet.

2.3 Driveline Control and Braking Systems

Active four wheel drive systems and Torque Vectoring systems for the enhancement of vehicle dynamics are increasingly popular. Development of these systems requires combined simulation of driveline components and vehicle dynamics, [2], and further extending this with models of the hydraulic system allows the effect of hydraulic dynamics on the vehicle behaviour to be studied.

Likewise the effect of antilock braking systems and brake force distribution on vehicle dynamics can be studied.

3 Issues with Hydraulic Simulation

A library previously developed [3] in Simulink has been successfully used for a number of projects, but requires an expert user. Lessons learnt from this work which have been applied in the ActuationHydraulics library include accounting for numerical issues and the availability of parameter data.

3.1 Numerical Issues

Numerical issues in hydraulic simulation are well discussed in [4], small volumes around valves lead to numerically stiff systems and non-linear systems of equations containing discontinuities arise due to in-

terconnected components with non-return valves and shuttle valves. These characteristics exploit well the capabilities of Modelica.

3.2 Data Availability

The ideal source of data is a disassembled component as in figure 2, as specifying orifice areas for valves is not necessarily compatible with the data made available by manufacturers. Data for hydraulic components is available in a wide range of forms, such as flow areas, lookup tables of flow against pressure or special forms such as the Lohm unit devised by Lee Hydraulics. The ActuationHydraulics library uses “replaceable” flow-models for orifice flows, each compatible class defines flow as a function of pressure drop and parameters according to the data available.



Figure 2: Typical hydraulic spool

For some components a pressure drop across the component is the only data available. With some modelling tools it would be necessary to reverse-engineer an orifice size and flow coefficient to achieve this pressure drop. However the ActuationHydraulics library exploits the acausal capabilities of Modelica and allows the choice of a flow-model where pressure drop is specified as a function of flow.

4 ActuationHydraulics Library

The modelling library was developed specifically for the applications described and with the aim of overcoming the issues described. Although hydraulics libraries in Modelica already existed [5,6,7], these were either unavailable or did not fulfil all the needs of the target applications.

The library components are isothermal, the effects of temperature can be studied by running the models with different fluid parameters, but thermodynamic effects within the fluid are not taken into account.

All components in the library extend from a base class, `PartialHydraulicComponent`, which defines an “outer” instance of `HydraulicsGlobal`. This allows the fluid used in the model to be selected in one location and works in a similar manner to the “World” component within the Modelica MultiBody library.

A wide range of components are included in the library, including valves, pipes, accumulators, pumps and actuators.

4.1 Valves

As described earlier, complex spool arrangements can be modelled, with variation of parameters as illustrated in figure 3. Flow through spools utilise the improved flow formula described in [8].

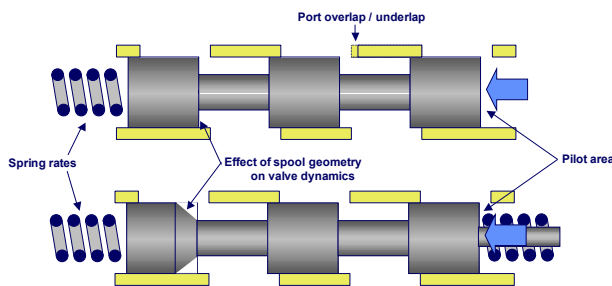


Figure 3: Modifiable parameters for spool modelling

Simple valves are modelled using the data available as described earlier. A range of simple valve types such as non-return valves are included.

4.2 Pipes and Accumulators

Models of pipe flow including inertia, compressibility and friction effects, including pipe wall expansion [9], are included. Lumped or distributed models can be used depending on the level of detail required. Gas-charged and spring-charged accumulator models are also included.

4.3 Pumps

The pump and motor models include volumetric losses due to leakages and compressibility, and mechanical losses due to friction and inertia. Variable swashplate models are included, as shown in figure 4.

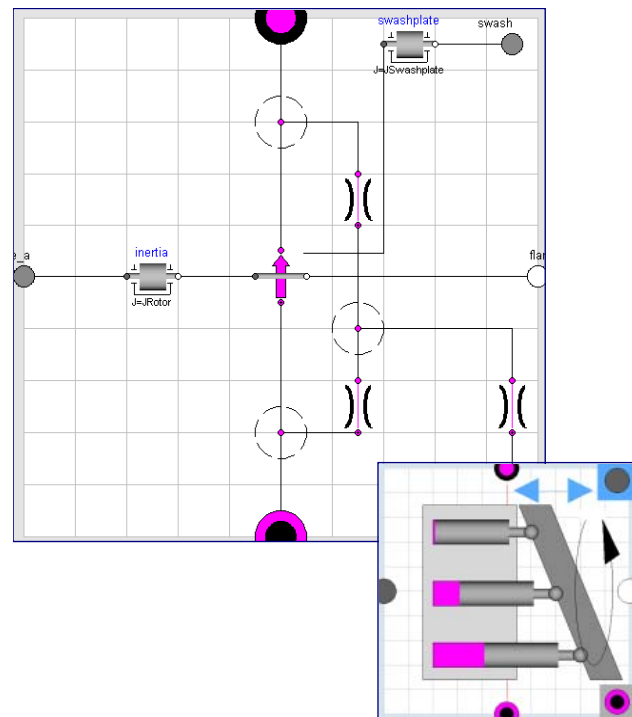


Figure 4: Diagram and Icon of Variable Swashplate Pump model

Flow ripple due to the number of pistons, teeth or lobes in the pump can be included, this can be useful for analysing noise and vibration aspects of hydraulic systems and ensuring volumes around pumps provide enough damping of these vibrations. Figure 5 shows the different magnitudes and frequencies of ripple for different pump configurations.

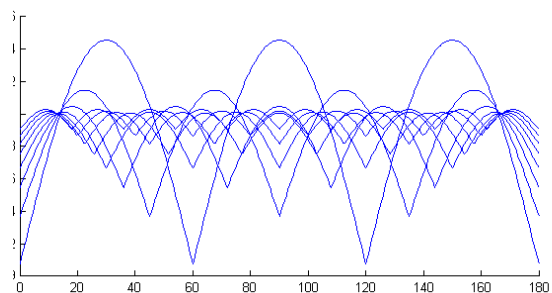


Figure 5: Pump flow ripple against angle for range of pumps with different numbers of pistons / lobes

4.4 Actuators

Translational and rotary actuators with inertia, friction, fluid compressibility and leakage effects are included.

Actuator, pump and spool components all interface with the Modelica standard library mechanics components, allowing connection to mechanical models from other libraries. Current signals to valves are applied using interfaces from the Blocks package within the standard library.

5 Case Studies

These case studies are recent typical uses of the library.

5.1 Continuously Variable Transmission

The most common form of Continuously Variable Transmission (CVT) utilises hydraulic actuators on conic “variators” to control the radius of a belt on either variator.

A CVT of this design was simulated using the ActuationHydraulics library in order to identify solutions to improve the poor controllability of the pressure at the primary variator. The relationship between current duty ratio at the solenoid valve and the pressure at the variator was validated against test data.

A parameter study was performed for spring rates and port overlaps in the main spool valve, resulting in a 100% increase in duty ratio range over which pressure is controlled and 25% reduction in hysteresis.

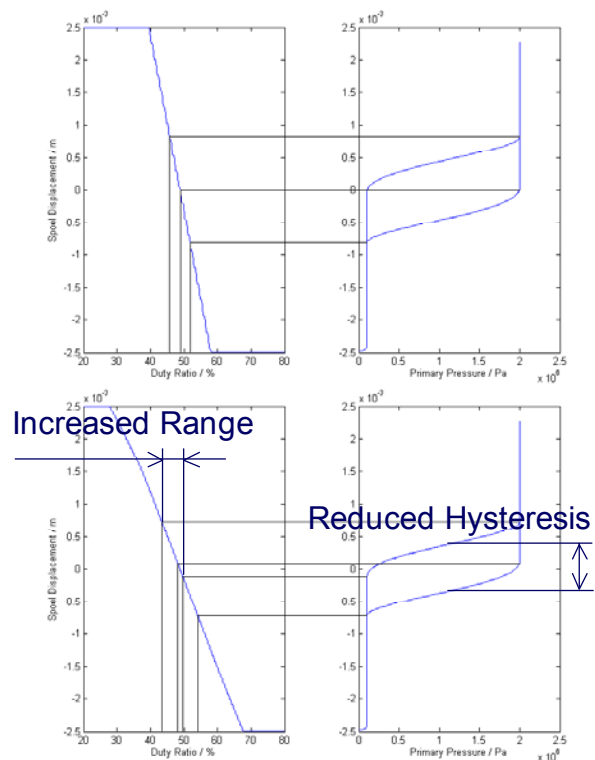


Figure 6: Increased controllability of CVT

5.2 Dual Clutch Transmission

Simulation using the ActuationHydraulics library was performed on the actuation and lubrication systems for the 750Nm 7 Speed Dual Clutch Transmission developed for the Chrysler ME4-12 vehicle.



Figure 7: Chrysler ME4-12 with Ricardo DCT

A key aim during design of the transmission was to minimise complexity and cost of the hydraulic system in order to show a transmission suitable for mass production. One result of this was the use of a shared pump between the lubrication system and the gear-shift actuation system. The sizing of the pump and

accumulator needed to be sufficient to allow the vehicle to change gear sequentially from 7th through to 1st whilst undergoing maximum deceleration at the minimum engine speed and hence minimum pump flow. It also had to provide enough lubrication flow to each of the bearings in the transmission.

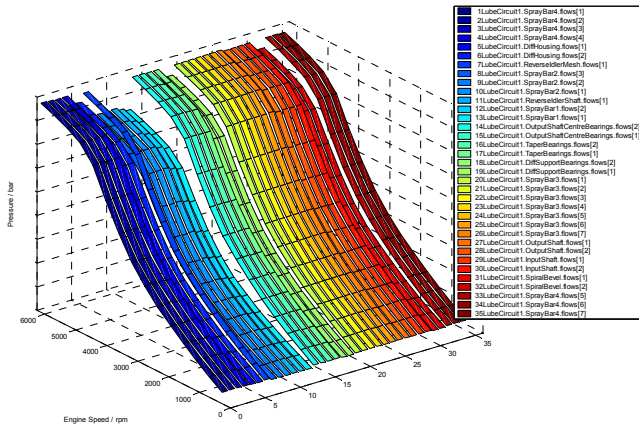


Figure 8: Lubrication flow against shaft speed at each bearing

The simulation was used, together with optimisation techniques, to define the pump and accumulator sizes and the sizes of each hole in the lubrication spray bars.

5.3 Formula 1 Hydraulics

Calculation of component sizes for F1 hydraulics is not a straightforward task as the pump and accumulator provide fluid for the gearshift, clutch, throttle, differential, power steering and fuel flap. A solution for this and control system testing and development has been provided by using Modelica simulation models which can either connect to race simulations or be driven by replays of test data, allowing engineers to study the effects of parameter changes.

6 Conclusions

The ActuationHydraulics library provides a solution for all automotive hydraulics applications, with connectivity to other Modelica packages. Successful use of the library has been shown for a number of applications.

References

- [1] Brandao, F., Harman, P., “An Integrated Approach: Ricardo Transmission and Driveline Simulation Library”, IMechE IPDS 2006
- [2] Brandao, F., Barnbrook, R., Shuttlewood, D., “An Integrated Approach Using IPG CarMaker and Dymola for Transmission and Driveline Modelling”, IPG User Conference 2006
- [3] Harman, P., “Dynamic Analysis of Transmission Hydraulic Systems”, 2004, Ricardo Technical Support Agreement (TSA) presentation
- [4] “IMAGINE SA Technical Bulletin 114: Numerical Challenges posed by Modeling Hydraulic Systems”, 2000
- [5] Beater, P., Otter, M., “Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator”, Proceedings of the 3rd International Modelica Conference 2003
- [6] Tiller, M., “Development of a Simplified Transmission Hydraulics Library based on Modelica.Fluid”, Proceedings of the 4th International Modelica Conference 2005
- [7] Beater, P., “Modeling and Digital Simulation of Hydraulic Systems in Design and Engineering Education using Modelica and HyLib”, Modelica Workshop 2000
- [8] Ellman, A., Piche, R., “A Modified Orifice Flow Formula for Numerical Simulation of Fluid Power Systems”,
- [9] Watton, J., “Fluid Power Systems”, Prentice Hall, 1989

Vehicle model for transient simulation of a waste-heat-utilisation-unit containing extended PowerTrain and Fluid library components

Max Eschenbach Jörg Ungethüm Dr. Peter Treffinger

German Aerospace Center; Institute of Vehicle Concepts

Postfach 800320, 70503 Stuttgart

max.eschenbach@dlr.de joerg.ungethuen@dlr.de peter.treffinger@dlr.de

Abstract

DLR Institute of Vehicle Concepts uses MODELICA for vehicle simulation. One major application is the power train analysis with regard to fuel consumption, emission and performance. The multi-domain approach of MODELICA has already been a benefit when describing alternative vehicle concepts like fuel cell vehicles and hybrid vehicles.

This paper describes the first approach modelling another concept of low-emission vehicles, i.e. a vehicle powered by an internal combustion engine (ICE), where a waste-heat-utilisation-unit in the exhaust system is applied. The recovery and conversion of waste heat in the exhaust system of ICE-powered vehicles is very attractive, because as a rule of thumb about one third of the fuel energy is emitted with the exhaust to the ambient. There are several approaches to convert exhaust energy into useful energy, e. g. applying a thermal engine, which is powered by the exhaust. Another possibility are thermoelectric devices, which convert heat into electricity. However, in almost all concepts for waste heat utilisation a heat exchanger in the exhaust system has to be integrated and additionally a heat sink has to be provided to run the complete process.

As an example for a waste-heat-utilisation-unit a thermoelectric device is considered in this paper. To predict the performance of such a thermoelectric device in vehicle operation the PowerTrain library [1] has been extended with elements of the Fluid library for representation of the exhaust system. Most important a special heat exchanger model has been developed which describes the thermoelectric device.

The paper gives an overview over the modelling approach of the complete system and the component models respectively. Finally, first results of simulation runs are given.

Keywords: model, dynamic simulation, waste heat recovery, thermal management

1 Modelling approach

The vehicle model has been based on components of the PowerTrain library. Figure 1 shows the object diagram of the vehicle model.

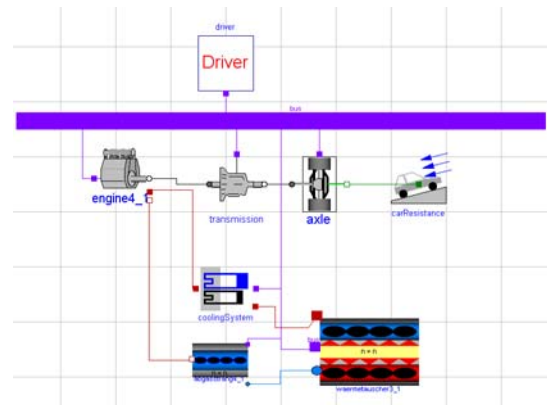


Figure 1: Object diagram of the PowerTrain vehicle model

It consists of the following main parts:

1. the vehicle model, including driver, engine, drivetrain, axle, and vehicle resistance as known from the PowerTrain examples.
2. the engine HeatModel, which determines the heat flow to the exhaust system and the cooling system (see heat flow connection to the respective sub systems). It also provides mass flow rate of the exhaust.
3. the cooling system, with its connection to the cooling system as a thermal heat flux.
4. the exhaust system, whereby actually no direct fluid connection to the engine has been realized. Instead the exhaust flow is generated in an exhaust gas generator as a first component of the exhaust line by using the heat flow, air and fuel mass flow provided by the engine.

Several signals from the bus are used as well in the physical models, e.g. the vehicle speed is taken into

account when calculating heat transfer from the exhaust pipes to the ambient.

1.1 Engine Model

As already indicated above the engine model must provide the heat flow to the exhaust system and the cooling system. Having in mind that the waste-heat-utilisation-unit is installed in the exhaust stream also the flow dynamics of the exhaust stream could be of interest. However, at the current stage of the development it was decided to concentrate on the thermal aspects. Precise determination of the heat flows in internal combustion engines by simulation requires CFD-Modelling of the coolant flow through the engine when the boundary conditions of the combustion are known to determine local velocities and heat transfer coefficients. This is mainly done during engine development and out of the scope of this work.

To obtain thermal engine models for thermo-management issues still semi-empirical approaches are used. These are based on simplified distributed mass models of the engine.

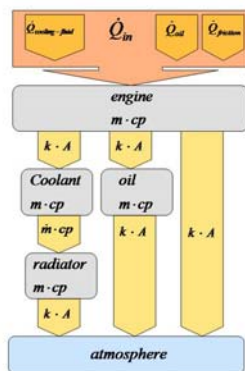


Figure 2: Thermal engine model with 4 masses

Figure 2 shows as an example the scheme of a thermal engine model with 4 masses. Road and roller dynamometer testing are used to determine the different thermal mass allocations by dynamic testing. Measurements on engine test benches can be used to quantify specific heat sources, e.g. by a successive component strip down, to identify the component friction as a heat source into the oil. As a result the energy transferred in cooling system and engine oil can be described depending on engine rotational speed and load. The engine-speed depending HeatModel included in the PowerTrain library has been extended to consider these dependencies. Thereby it turned out that the quadratic approximation of the

total heat release into the cooling system, which was already implemented, did not fulfil the special requirements of this application. The HeatModel was therefore refined regarding the distribution of the different thermal flows.

For instance the engine temperature is now considered when determining the heat flow into the engine coolant. A simplified engine heat up is thereby considered in the heat model. In [2] it is stated that a temperature rise of the cylinder wall temperature from room temperature up to 150°C reduces the heat flux through the walls by approximately a third.

Convective heat flux over the engine surface has been neglected so far. This holds, when low power cycles, e.g. the New European Drive Cycle (NEDC) are investigated. The new approach ensures that the fuel energy is either converted into mechanical energy, a heat flux into the coolant system (by friction or conduction) or released as sensible heat with the exhaust gas.

1.2 Exhaust system model

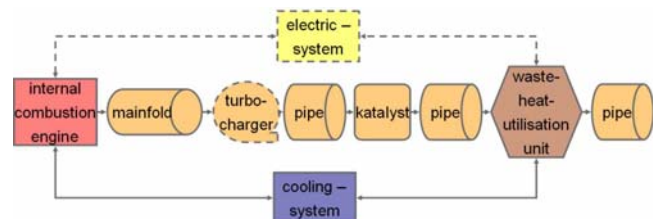


Figure 3: Scheme of exhaust system

Figure 3 shows a schematic representation of an exhaust system with a waste-heat-utilization-unit integrated. First element is the exhaust flow from the internal combustion engine into the manifold. The behavior of the exhaust system regarding thermal and flow characteristic is accounted for by combining components like pipes and catalysts. The position of the waste-heat-utilization-unit within the exhaust line is important, because the temperature of the exhaust decreases along the exhaust line due to heat release to the ambient.

Due to the requirement to reach the operation temperature of the catalyst quickly, such a device will be most likely placed behind the catalyst. The heat transfer from the exhaust line to the ambient depends on the geometry of the exhaust line, the geometry of the air ducts around the exhaust line, the ambient temperature, the velocity of the vehicle and many more parameters. It was out of the scope of this study to implement a complex model, which takes all these effects in account.

A correlation derived from experiments [3], which gives the heat release of exhaust line components to the ambient as function of the vehicle velocity has been applied to the components of the fluid library (e.g. pipe).

1.3 Waste-heat-utilization-unit

Figure 4 shows a scheme of the waste-heat-utilisation-unit. It is basically a heat exchanger, where heat is transferred from the hot exhaust gas to the engine coolant, whereby a temperature difference over the thermoelectric material occurs, which is placed between the walls of the exhaust gas channel and the cooling channel.

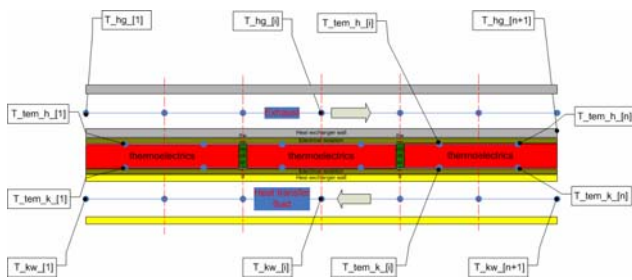


Figure 4: Scheme of the waste-heat-utilization-unit.

Depending on the Seebeck-coefficient and the temperature difference a voltage is generated.

$$U_{th} \approx (\alpha_n + \alpha_p) \cdot (T_h - T_c) \text{ with}$$

α_n/α_p abs. of neg./pos. Seebeck-coefficient, V/K

T_h/T_c hot/cold side temperature of thermoelectric material, K

Peltier-Effect and the heat release caused by electric resistances in the thermoelectric material are taken into account as proposed in [4]. Basics about thermoelectrics can be found e.g. in [5]. Traditional thermoelectric materials are PbTe, Bi₂Te₃, SiGe, BiSb or FeSi₂. Thermoelectric devices based on these materials allow efficiencies – i.e. electrical power divided by the heat flow dragged through the thermoelectric material - of around 5%, whereby temperature differences around 200 K are required. Due to recent innovations in thermoelectric materials, which promise higher efficiencies – higher than 10% - their application for waste heat conversion is gaining more interest [6].

As indicated in figure 4 the waste-heat-utilisation-unit is modelled as a distributed exhaust heat exchanger, which contains thermal masses and considers heat conduction between the different layers and along the layers as well. Due to the varying engine loads during the cycle a wide range of flow velocities on the hot side were observed. How-

ever, the basic “DistributedPipe_thermal”-model of the Fluid library considers only a constant heat transfer coefficient. Therefore a heat transfer correlation was implemented, which gives the heat transfer coefficient as function of Reynolds- and Prandtl-number, namely the Gnielinski-correlation [7].

2 Simulation results

The complete simulation model has been tested applying the NEDC, which is equipped with an 80 kW-Gasoline engine. The parameters of the vehicle model have been estimated mainly from experimental investigations, which are taken from the literature [8]. The parameters of the waste-heat-utilisation-unit have been estimated by analyzing similar devices, which have been built for automotive application [9].

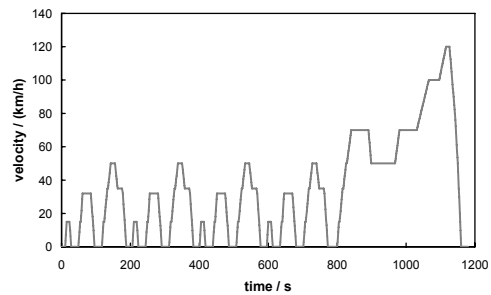


Figure 5: New European Drive Cycle (NEDC).

Figure 5 shows the velocity profile of the NEDC, which is a relative low power cycle. This leads to a slow warm up phase.

Most simulations or testing procedures of waste-heat-applications are performed under steady state conditions at higher engine loads and vehicle speeds, defining the optimum operation point. This simulation is focused on the dynamic power output under disadvantageous conditions for waste-heat-applications regarding the lower thermal energy supply to the exhaust system. Under these driving conditions noticeably gradients in cooling and exhaust temperature lead to a dynamic power output of the device.

As can be seen in figure 6, which shows the development of the hot (red) and cold side temperature (blue) over the NEDC, starting from ambient temperature the heat up of the waste heat utilization unit takes considerable time. A temperature difference near 100 K is only reached after around 1100s, where the highest load of the cycle occurs.

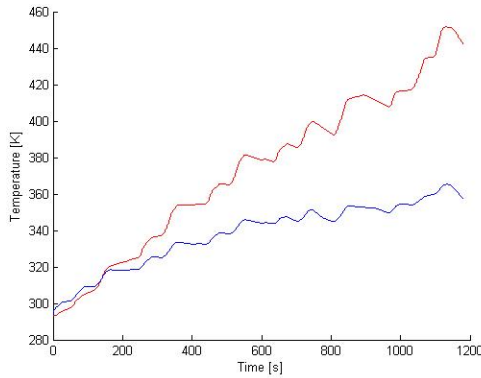


Figure 6: Hot and cold side temperature of the thermoelectric device over the NEDC.

The hot side temperature increases almost steadily over the cycle. The cold side temperature seems to approach a maximum, which means that the cooling system comes near to its operation temperature. As indicated above a temperature difference of around 200 K would be necessary to achieve a good performance of the thermoelectric device, which has been assumed in this work. The thermoelectric device delivers therefore only a fraction of its specified electrical power output under nominal conditions. Figure 7 shows the efficiency of the conversion of the thermal energy into electricity by the thermoelectric device over the NEDC, i.e. when the temperature difference shown in figure 6 is applied.

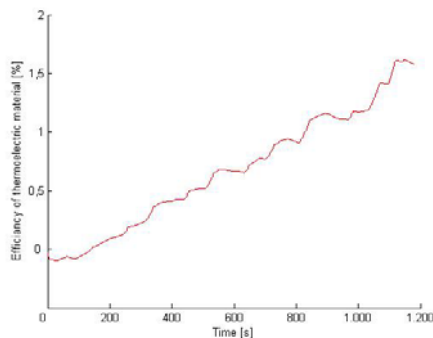


Figure 7: Efficiency of thermoelectric material over the NEDC (see also figure 6).

The efficiency is negative in the beginning, caused by a reverse heat flow through the heat exchanger. It reaches 1,6 % at the maximum, when the highest temperature differences of that cycle are present. However, the temperature difference should be even higher to approach the nominal value of the design point for this type of device. As we have seen in figure 6, the hot side temperature is almost steadily increasing over the NEDC. Therefore reducing the mass of the waste-heat-utilisation-unit should not

only improve the response time, but also yield to higher temperature differences.

The effect of a 50 % mass reduction of the waste-heat-utilization-unit on the temperature difference over the NEDC is shown in figure 8 (top). The smaller response time is indicated by sharper temperature drops in the periods, when no load is applied in the NEDC. However, also the temperature difference is considerably higher compared to the reference waste-heat-utilization-unit with 100 % mass. Consequently the electrical power output during the complete cycle is higher for the device with 50 % mass (see figure 8, bottom).

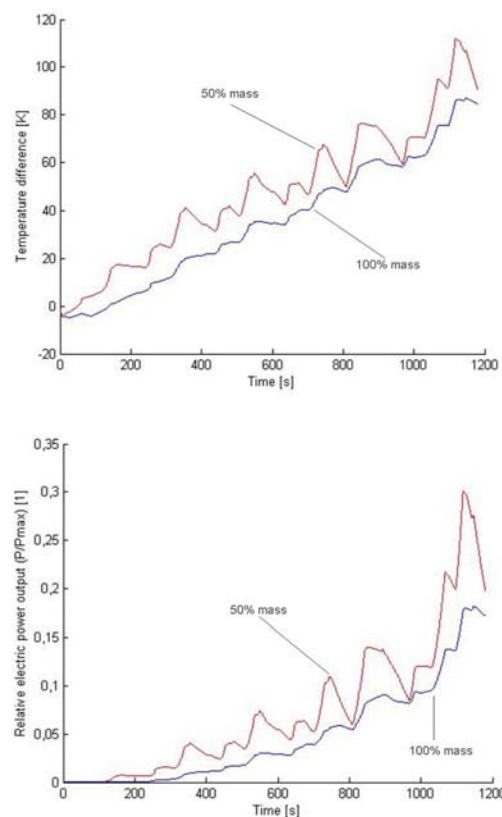


Figure 8: Top: Temperature difference between hot and cold side of the thermoelectric device with 50% (red) and 100% (blue) mass over the NEDC; bottom: Relative electrical power output of the thermoelectric device with 50% mass (red) and 100% mass (blue) over the NEDC.

The electrical output of the waste-heat-utilization-unit is reduced by the auxiliary power required to drive the coolant pump and possibly the cooling fan. An experimental investigation of a vehicle on a roller dynamometer equipped with a prototype waste-heat-utilization-unit showed even negative net power at low loads, i.e. low vehicle velocities [10].

The reason is the power demand of the cooling pump, which has been operated continuously over all load conditions. This coolant mass flow is covered for an optimal power output at high temperatures.

Considering the following constraints:

1. Maximum temperature of the thermoelectric modules
2. Boiling temperature of the coolant fluid

a flow controller and an electrical cooling pump were implemented in the cooling system model. This allows decreasing the flow rate of the cooling fluid to a minimum value at low engine loads to avoid unintentional heat flux into the cooling system at operation points with negative net power.

Figure 9 shows a comparison of the electrical power output of the 100 % mass waste-heat- utilization-unit w/o and with a flow controller. After approx. 750 s the controller switches from a minimum flow rate to the maximum flow rate due to rising coolant temperatures.

The electrical power output of the configuration with controller is behind the configuration w/o controller for the first 750 s. After applying the optimum flow rate the electrical power output rises quickly and reaches even higher values than the configuration w/o controller, which is due to the conversion of thermal energy stored in the unit.

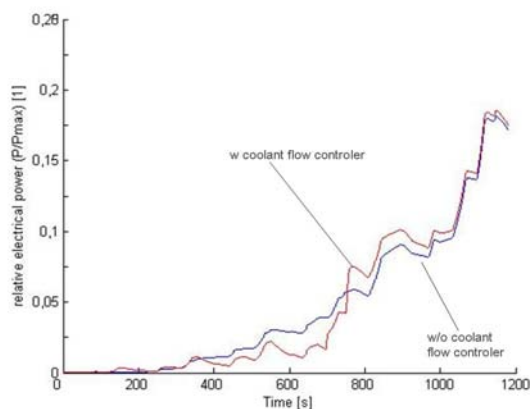


Figure 9: Relative electric power of the 100% mass waste-heat-utilization-unit: a) w/o flow controller (blue line), b) w flow controller (red line).

Over the complete NEDC the electrical output of the unit with flow controller is 2% than the one of the configuration w/o flow controller. However, this is overcompensated by the reduction of the power consumption of the cooling pump and yields to a higher net efficiency.

3 Conclusions

A model of a vehicle with a waste-heat-utilization-device in the exhaust system has been built based on the PowerTrain library. Therefore some components of the PowerTrain library have been extended with respect to the modelling of their thermal behaviour. Components of the Fluid library have been taken and improved to describe the exhaust system. Thereby it was possible to set up a simplified vehicle model. However, looking on the capabilities of specialised automotive tools for the simulation e.g. of the exhaust system or the cooling system, the additional effort to realize more detailed models becomes apparent. Nevertheless the multi-domain approach of MODELICA allows simulation with one environment and avoids co-simulation.

The usage of the model has been underlined by some simulation results, whereby the investigation of control strategies with respect to optimised net power output has been indicated. From the given examples it can be seen, that a suitable control strategy and optimized design supports the electric power output.

As already mentioned above the model should be refined, especially the representations of the exhaust system and the cooling system are expandable. Having in mind the unsteady electrical power output of the waste-heat-utilization-unit also the electrical system of the vehicle should be added in order to investigate, whether the power supply fits to the power demand.

References

- [1] Otter, M., Dempsey, M., Schlegel, C.; Package PowerTrain. A Modelica Library for Modeling and Simulation of Vehicle Power Trains. Proceedings of the 1st Modelica Workshop, Lund, Sweden, October 2000, Modelica Association, pp. 23–32.
- [2] Guzzella, L., Onder, C.; Introduction to Modelling and Control of Internal Combustion Engine Systems. Springer-Verlag Berlin Heidelberg, 2004.
- [3] Kandylas, I., Stamatelos, A.; Berechnung des Wärmeübergangs in Motorabgassystemen. Motortechnische Zeitschrift (MTZ), Ausgabe 1998-08.
- [4] Müller, E.; Thermoelectric materials for waste heat recovery, linerized thermal sensors and Peltier cooling. German Aerospace Center (DLR), Institute of Materials Research, at Steinfurter Keramiktage, Dezember 2002.

- [5] Nolas, G.S., Sharp, J., Goldsmid, J.; Thermoelectrics: basic principles and new material developments. Springer-Verlag Berlin Heidelberg, 2001.
- [6] Venkatasubramanian, R., Sivola, E., Colpitts, T.; Thin-film thermoelectric devices with high room-temperature figures of merit., Nature Magazin, 2001 Volume 413, pp. 596-602.
- [7] VDI-Wärmeatlas, 7.Auflage, 1994, Abschnitt Gb
- [8] Poruba, C., Seider, G., Kröner, M.; Energiebilanierung im Entwicklungsprozess von Motoren. Wärmemanagement des Kraftfahrzeuges V, Haus der Technik Fachbuch, Düsseldorf 2006, pp. 54-80.
- [9] Vázquez, J., Sanz-Bobi, M.A., Palacios, R., Arenas, A.: State of the Art of Thermoelectric Generators Based on Heat Recovered from the Exhaust Gases of Automobiles. Universidad Pontificia Comillas Escuela Técnica Superior de Ingeniería. Instituto de Investigación Tecnológica, Madrid, Spain.
- [10] Thacher, E.F., Helenbrook, B.T., Karri, M.A., Richter, C.J.; Testing an automobile exhaust thermoelectric generator in a light truck. Clarkson University, Potsdam, New York, USA, Department of Mechanical and Aeronautical Engineering, Delphi Corporation, Lockport, New York, USA, 2005.

Modeling, Calibration and Control of a Paper Machine Dryer Section

Johan Åkesson¹ Ola Slätteke²

¹Department of Automatic Control, Faculty of Engineering, Lund University, Sweden
(jakesson@control.lth.se)

²ABB Ltd., Finnabair Industrial Park, Dundalk, Co. Louth, Ireland
(ola.slatteke@se.abb.com)

Abstract

Following increased efforts during the last decade to formulate mathematical models for the paper drying process, this paper presents a Modelica library, *DryLib*, which enables users to rapidly develop complex models of paper machine dryer sections. In addition, parameter optimization, model reduction and moisture control by means of Non-Linear Model Predictive Control is treated. These applications have in common that they are based on numerical optimization schemes. Since the nature of the particular optimization problem dictates the requirements of the numerical code, the paper also serves as an illustration of the need for flexibility in terms of *i*) means for the user to express optimization problems, *ii*) and choice of numerical algorithms.

Keywords: paper machine, paper drying, parameter optimization, model reduction, non-linear MPC

1 Introduction

The topic of this paper is modeling, model reduction, parameter optimization and control of a paper machine drying section. The dryer section is the last part of the paper machine and consists of a large number of rotating steam heated cast iron cylinders. The moist paper is led around these cylinders and the latent heat of vaporization in the steam is used to evaporate the water from the web. The cylinders are divided into separate dryer groups where the steam pressure can be individually controlled in each group. By adjusting the steam pressure in the dryer groups, and thereby the heat flow to the paper, the moisture in the paper web is controlled.

Based on the work [11], a Modelica library, *DryLib*, has been developed. *DryLib* implements

the physical phenomena involved in the drying process, as well as convenient components and connectors which enables rapid development of dryer section models. An important feature of *DryLib* is its ability to express models which are scalable, in the sense that the complexity of the models can be easily changed. This feature is quite useful, since the need for granularity depends on the application – a high fidelity model may be suitable for simulation, whereas a coarse model capturing the main behavior may be appropriate for control design.

The present paper gives three main contributions. Firstly, the Modelica library *DryLib* is presented. Secondly, important issues such as parameter optimization, model reduction and optimization based control schemes (Non-linear Model Predictive Control (NMPC)), are treated. Some of these topics have a general character, while others are dealing specifically with dryer section issues. Thirdly, the applications of the paper serves as examples of the wide range of relevant optimization problems that naturally follow the availability of high-fidelity models.

The paper is organized as follows. In Section 2 the structure and implementational details of *DryLib* are treated. The Sections 3, 4 and 5 treats parameter optimization, model reduction and moisture control by means of non-linear MPC. In Section 6, the software used to solve the optimization problems presented in the paper is described. The paper ends with with conclusions and future work in Section 7.

The current paper is a condensed version of [2], where additional details can be found.

2 DryLib

The model library, *DryLib*, that is developed and used in this paper is built upon physical relations in

terms of mass and energy balances, in combination with constitutive equations for the mass and heat transfer. The objective is to obtain a non-linear model that captures the key dynamical properties for a wide operating range. The core of the model is based on [16] and [12], and it is also given in [11]. The model for the paper web is based on [16] whereas the model for the cylinder, and steam system is taken from [12]. The mathematical model used as a basis for `DryLib` is identical to the model described in the references, except that convective heat transfer from the paper web has been added.

The objective of building the Modelica library `DryLib` has been to create a user friendly and extensible platform for modeling of paper machine dryer sections. In particular, the aim has been to design the library so that, at the user level, the appropriate level of model detail can be easily selected. The current implementation of `DryLib` contains a few examples of components where the level of detail can be specified by the user. More importantly, the library classes are designed to enable advanced users to add new behavior to key components in order to extend the functionality of the library. An important concept in the design process has been that of *model scalability*, which means that the granularity of the model behavior should be easy to change, without the need to re-build the model.

2.1 Hierarchical Structuring

Having formulated a mathematical model for the paper machine dryer section, the issues of structuring the equations into Modelica classes and definition of interface classes (connectors) need attention. A paper machine dryer section model can be assembled using very few basic component types. In essence, there are only two fundamental entities, namely a steam heated cylinder and a sheet of paper. These two component types may then be combined, in large numbers, into a complete dryer section model. However, it is convenient to introduce additional hierarchical levels. As discussed above, the cylinders of a typical dryer section are organized into steam groups, in which a number of cylinders are operated at the same pressure. The introduction of steam groups into the library provides a convenient hierarchical level for the user, since many decisions regarding e.g. operating points and control design and evaluation are made at the steam group level. For basic usage of `DryLib`, it is also sufficient to utilize only classes defined at the steam group level in order to create a fully working dryer section model.

In order to increase the flexibility of the library, the

boundary conditions of the physical entities have been factored out and modeled as separate classes. As an instructive example we consider a paper sheet, where the boundary conditions of the surfaces defining the sheet depends on the environment. For example, different boundary conditions are imposed on the surface if the paper is in contact with the air or a cylinder shell. The key to building a flexible Modelica libraries using this principle of separation is the design of generic connector classes. This topic will be discussed in detail below.

From a user's perspective, `DryLib` is intended to enable easy modeling of a dryer section. However, the user should remain in control of the implementational details of key components, e.g. paper sheets and cylinders. Also, advanced users should have the possibility to introduce new behavior of existing components. Two key features of Modelica have been used to satisfy these requirements. In the first case, extensive use of parametrized types (`replaceable/redeclare`) has been used to propagate type information downwards in the component hierarchy from the main user level (which is the steam group level) to lower level components. This strategy enables the user to easily select the appropriate level of detail for e.g. the cylinder dynamics. In the second case, inheritance has been used in order to simplify introduction of new component behavior. For the basic components such as cylinders and paper sheets, generic base classes have been introduced, which in turn serve as super classes for particular implementations. `DryLib` currently provides a few alternative implementations for key components, and additional behavior is easily added using the pre-defined base classes.

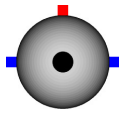
2.1.1 Connectors and Variable bindings

The interface structure in `DryLib` is based on three connector classes. While the connectors for heat flow and mass flow (for connecting components with steam flow) are straight forward, the connector class for a paper surface deserves to be discussed. The paper web is modeled by separate mass balances for water and fiber, and an energy balance, as described above. Natural flow variables are thus mass flow of water and fiber, q_w [kg/s] and q_f [kg/s], and energy flow Q [W]. As for the potential variables, there are several feasible choices. However, since `DryLib` is likely to be used by domain experts in the field of paper drying, it was decided to use the standard variables within this domain. The natural choices are then moisture ra-

tio, u [kg water/kg dry substance], dry basis weight, g [kg/m²] and temperature T [°C].

A particular feature of Modelica that has been used to simplify the propagation of parameters and variables between components in DryLib is name look-up in the instance hierarchy (*inner/outer*). For example, the machine speed is used in various components, but is common for the entire dryer section. Implementation using *inner/outer* constructs is thus convenient. Examples of variables that may be assumed to be shared by the components of a steam group are ambient temperature and air moisture, which are also implemented using *inner/outer*.

2.1.2 Cylinder Models



The (partial) cylinder base class `CylinderBase` contains mainly connector components and serves as a unifying class for particular implementations of dynamic behavior. The cylinder base class has two mass flow connectors corresponding to steam inlet and outlet, and one heat flow connector. Two particular implementations of the cylinder dynamics is included in DryLib.

2.1.3 Paper Models



The paper web base class contains essentially four paper connectors corresponding to the cross section areas and the upper and lower surfaces. This design enables separation of the actual paper web behavior, and the physical phenomena defined by the boundary conditions of the paper.

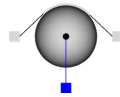
2.1.4 Interfaces

Having defined the connector structure and the basic cylinder and paper classes, modeling of the interfaces between components is straight forward. The `PaperPaperInterface` class models the interface between two paper cross section areas perpendicular to the machine direction. In `CylinderPaperInterface`, the heat transfer between a cylinder and a paper in contact is modeled. Finally, the evaporation of water from the paper surface is modeled in the class `Evaporation`.

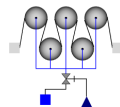
2.1.5 Steam Group Models

The classes described above have the character of *specifying physical behavior*. We shall now turn our

attention to classes which are mainly used as *structuring entities* in the sense that they introduce new hierarchical levels, and that they contain instances of behavior classes. Basic usage of DryLib may involve only classes introduced at this level.



In order to efficiently explore the strong repetitive character of a typical dryer section, the class `CylinderUnit` was introduced. This class combines a steam cylinder and a paper sheet which is attached to an evaporation component. While different cylinders may have different physical parameters, the structure of `CylinderUnit` is valid in most cases.



The actual control system typically consisting of a valve, a pressure sensor and a PID controller is encapsulated in the class `SteamGroup`, which also contains an array of `CylinderUnit` components. The `SteamGroup` class has four connectors corresponding to incoming and outgoing paper, the steam header and an input signal representing the reference value of the pressure controller.

2.1.6 Miscellaneous

Apart from the classes presented above, DryLib also contains classes which is used to drive a dryer section model, referred to as sources and sinks. In addition there are some miscellaneous classes for e.g. valves and sensors.

2.2 PM7, Husum, Sweden

To demonstrate the capabilities of DryLib, a dryer section model corresponding to that of PM7 located at the M-real mill, Husum, Sweden, has been developed. The PM7 paper machine is a multi-cylinder machine producing copy paper. The dryer section of the machine is divided into a pre-dryer and an after dryer section with the surface sizing in the middle. The objective of the after-dryer section is only to dry the mixture added by the surface sizing and it cannot take care of moisture problems from the pre-dryer section. Only the pre-dryer is modeled here. The PM7 drying cylinders are divided into six groups, consisting of one, two, two, three, ten and twelve cylinders respectively. For a detailed description of the plant, see [5].

In Figure 1, the top level of the PM7 dryer section model is shown, including six steam groups, a paper source, a paper sink, a mass flow source representing the steam header and a set point distribution for calculation of pressure set points for the groups. The final

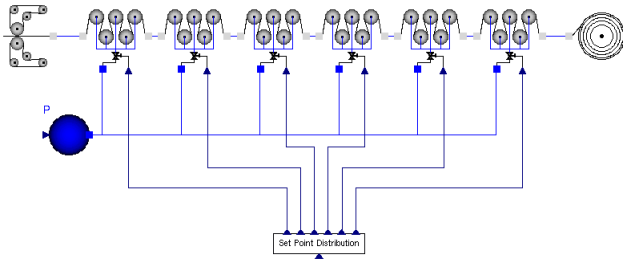


Figure 1: The top level of a complete dryer section model.

model consists of 7453 equations and 312 dynamical states when translated with Dymola.

2.3 Extensions

Possible extensions of `DryLib` can be sorted mainly into two categories. Firstly, the library may be extended by adding components modeling process equipment or physical phenomena not covered by the current implementation. For example, modeling of systems in direct connection with the dryer section, such as the condensate system, the steam production and the ventilation system would enable simulation of a larger part of the process. Also, adding this functionality would simplify connection of the dryer section model to models of other important parts of the paper machine, e.g. the press section, the wire section or other process units utilizing the same steam header.

Secondly, `DryLib` may be extended by introducing components which enables simulation of the drying process at an increased level of detail. The current design of `DryLib` is based on a particular choice of discretization of the underlying PDE:s (describing mass and energy transport), which yields a model with a reasonable level of detail, while maintaining acceptable simulation times. While this choice of discretization is suitable for analysis of moisture, temperature and pressure profiles in the machine direction, other applications may require different levels of detail.

3 Parameter Optimization

It is desirable that the behavior of the model is similar to that of the real plant, in order for results obtained from using the model to be applicable on the plant. A common method to minimize the plant-model mis-match is to select one or more parameters of the model, and then tune these until a satisfactory model

response is obtained. This procedure of tuning parameters while leaving the structure of the model unchanged is referred to as gray-box identification, see [4]. Parameter tuning may in simple cases be done by hand, but more complex problems requires structured methods for finding the parameter set which yields the best result. One such method is parameter optimization, which, in addition to selection of parameters to optimize, also includes definition of a performance criterion to minimize.

When selecting parameters to optimize, parameters which are uncertain are attractive choices. However, it should be kept in mind, that the parameter optimization procedure does not necessarily produce the physically correct parameter values. Rather, the selected parameters are used to compensate for all types of model-data mismatch given a particular performance criterion. This implies that the actual parameter values obtained from optimization should not be interpreted as the true physical values, but rather those that achieves the best model-data match. On the other hand, it is usually desirable to ensure that parameters have physically feasible values.

3.1 Problem Definition

Setting up a parameter optimization problem requires insight into which aspects of the model are most important. In this case, both the dynamic and static model response is of importance. However, in a first step, only the static behavior has been considered. Specifically, cylinder and paper temperatures of the paper machine, as well as the output moisture, have been measured during stationary operation conditions. The aim of the optimization has been to improve the stationary response of the model in the sense that the difference between simulated temperatures and moisture and measured temperatures and moisture, should be minimized.

A reasonable cost function to minimize is then

$$J = \gamma_{T_m} \sum_{i=1}^{N_{cyl}} (T_{m,i}^m - T_{m,i}^s)^2 + \gamma_{T_p} \sum_{i=1}^{N_{cyl}} (T_{p,i}^m - T_{p,i}^s)^2 + \gamma_u (u_{out}^m - u_{out}^s)^2 \quad (1)$$

where N_{cyl} is the number of cylinders, super-script m indicates measured quantities, super-script s indicates simulated quantities and γ_{T_m} , γ_{T_p} and γ_u are weights. While the measurement method used to determine cylinder temperatures is reliable, the measurements of paper temperatures should be regarded as uncertain.

In particular, the paper temperature is varying considerably in the machine direction depending on the position, relative to a cylinder contact area, at which the measurement is done, [11]. Therefore, the weight γ_{T_p} was set to a small value. The moisture, on the other hand, is an important quality variable that should be matched with high accuracy. Accordingly, γ_u was set to high value.

Four parameters were selected for optimization – three heat transfer coefficients and one mass transfer coefficient.

3.2 Solving the Problem

The minimization of (1) should be performed subject to the constraint constituted by the DAE representation of the model. Since the minimization is performed in stationarity, all derivatives may be set to zero, and the model is then represented by a purely algebraic constraint, $F(x, y, p) = 0$, where x is the state vector, y represents the algebraic variables and p are the parameters.

The problem was solved by a custom made application coded in C, which is based on the `dsblock` interface for accessing the model description generated by Dymola, and the NLP code IPOPT, see [14], which is dedicated to solving large scale algebraic optimization problems. The software is described in detail in Section 6.

3.3 Parameter Optimization Results

Minimizing (1) yields an optimal cost of 277, compared to the cost 61869 for the nominal parameter values. The optimal temperature profiles are shown in Figure 2. For comparison, the nominal profiles are plotted. As can be seen, there is a significantly improved fit between simulated and measured responses. In particular, the output moisture in the nominal case is unrealistically low too early in the dryer section. It can also be noted that the fit of the cylinder temperature profile is better than that of the paper temperature profile. This phenomenon is expected, since the weight of the paper temperature errors was set to a low value.

The matching of the profiles can be improved further by introducing additional optimization parameters. This strategy is explored in [1] for a slightly different parameter optimization problem.

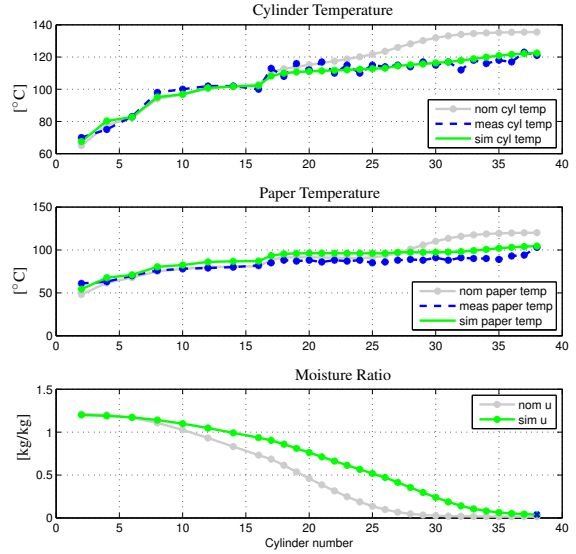


Figure 2: Stationary temperature and moisture profiles. The x-axis shows cylinder numbers.

4 Model Reduction

Dryer section models built using DryLib results in large scale models, even though a sparse discretization scheme for mass and energy balances has been applied. For control design and evaluation purposes, however, a model describing the dynamic relationship between the inputs and the quality variables at the last free draw is usually sufficient. In practice, low order models (e.g. KLT-models with a gain, a time delay and a time constant) valid at a specific operating point are commonly used for dryer section control. In this section, a reduced model targeted towards moisture control design is developed. Since the moisture measurement signal available for feedback control is usually obtained at the end of the dryer section, the aim of the reduction scheme is to develop a simpler model, which captures the non-linear dynamical behavior relating the steam pressure reference signal, input moisture, input temperature and dry basis weight (from the press section) to output moisture. Accordingly, accurate simulation of the paper temperature and the moisture profile can be compromised in order to obtain a lower order model, describing only the phenomenon of interest, i.e. the behavior of the moisture, accurately.

4.1 The Equivalent Dryer

In this paper, the structure of the dryer section will be exploited, in order to obtain a model of lower order.

A previously reported concept is that of the equivalent dryer, which is described in [10]. Instead of modeling each cylinder as a separate unit, the equivalent dryer concept suggests that one, larger cylinder can be used to approximate an entire steam group. This approach has several attractive features. *i)* It preserves the structure of the dryer section, since each steam group is replaced by its corresponding equivalent dryer, *ii)* each equivalent dryer has an intuitive physical interpretation *iii)* and the reduction potential is large, especially for large steam groups.

4.2 The Reduction Problem

At the steam group level, the reduction problem can be stated as *“Find the dimensions of one steam cylinder, including associated incoming and outgoing free draws and contact paper, which approximates as well as possible, the behavior of a given steam group”*. Given our experiences from simulation of dryer section models, we suggest that the dynamic and stationary response of the equivalent dryer cylinder may be treated separately. As for the dynamics, we assume that the mass and volume of the equivalent cylinder can be set to N_{cyl} times those of an individual cylinder in the steam group, where N_{cyl} is the number of cylinders. Now, simulation experiments reveal that the time constant of an equivalent cylinder, constructed based on this assumption, corresponds well to the time constant of the full steam group. However, the same result does not seem to hold for the stationary gains, where there is a significant mismatch. Intuitive ways to set the lengths of the free draw and contact papers, using the same reasoning as for mass and volume, does not produce acceptable results. A more sophisticated way of finding the physical dimensions and parameters is thus necessary.

4.3 Reduction of One Steam Group

A static model for a paper sheet in contact with a steam cylinder, can be formulated using algebraic versions of the dynamic mass and energy balances of the mathematical dryer section model. The algebraic systems of equations for each cylinder and paper web may then be duplicated and put together to formulate a static model for a steam group.

As stated in the introduction of this section, the most important quality variable, at least for moisture control, is paper moisture. Therefore, a reasonable objective is to minimize the deviation between the moisture in the last free draw of the cylinder group, and

the moisture in the outgoing free draw of the equivalent cylinder. In addition, as a secondary objective, it was decided to minimize the deviation in steam consumption. This objective was added since it may be desirable to limit the steam consumption during moisture control. Performing this minimization for a single operating point is not sufficient, however. In order to obtain a good fit over a wider operating range, a set of operating cases was introduced, over which the optimization was performed. Each case consists of a specification of the operating point in terms of steam pressure, input moisture, input temperature and basis weight.

It remains to define the optimization parameters, over which the minimization is performed. Six parameters of the equivalent dryer were selected for optimization, namely the length of the free draws, the length of the contact paper, and in addition two heat transfer coefficients and one mass transfer coefficient. The number of variables that are actually needed to obtain a good fit is not unambiguous, however. For small steam groups, or if few cases are used, some of the suggested optimization variables may well be fixed, without any increase in the approximation error. In fact, it is desirable to find an appropriate trade-off between the number of optimization variables and optimization performance, in order to avoid over-parametrization.

4.4 Reduction of a Dryer Section

A straight forward approach for deriving a reduced order dryer section would be to simply apply the method described in the previous section for each individual steam group. Recalling our main objective, which is to predict the moisture in the last free draw, this approach would not explore the full potential of the method. Instead, a larger optimization problem, incorporating all groups, may be formulated where most attention is given to minimizing the deviation of the last group. This means that *all* groups are reduced at the same time, and that the full reduction potential is used according to the main objective, which is to predict the final moisture. It may, however, be advantageous to include the deviations, with small weights, of all groups in the optimization criterion, in order to avoid a physically unrealistic model.

4.5 Solving the Optimization Problem

The resulting algebraic optimization problem is challenging, both due to its size and its non-linear char-

acter. The final problem consists of 9536 free variables and 9504 equality constraints, of which 8568 are non-linear. Efficient solution of large scale NLP problems of this type require state of the art numerical algorithms, exploring the sparse structure of the problem as well as analytical Jacobian and Hessian information.

The problem definition was programmed in AMPL, which is a language for mathematical programming, [6]. AMPL enables encoding of linear and non-linear algebraic optimization problems, using optimization oriented language constructs. The problem description, i.e. the AMPL code, is then executed within the AMPL tool, which in turn interfaces several numerical solvers. In this application, the NLP code KNITRO, [15], has been used. The combination of AMPL and KNITRO is extremely powerful, since the AMPL interface to numerical solvers offers analytic evaluation of Jacobians and Hessians as well as sparsity information. This enables KNITRO to operate in its most efficient mode, resulting in acceptable execution times also for large systems. The reduction problem formulated in the previous section is solved in about 2-5 minutes, depending on initial starting point.

The proposed method has the distinct drawback of requiring complete re-encoding of the the model description. This was necessary, however, in order to enable utilization of the appropriate symbolical and numerical algorithms.

It is important to note, however, that the problem is non-convex, and that only local optimality can be expected. However, in this case, the solution to the reduction problem seemed to be robust with respect to different starting points. Also, the obtained solution is reasonable in the sense that the optimized parameter values lies within physically feasible limits.

4.6 Model Reduction Results

As mentioned above, a set of operating conditions need to be specified, in order to complete the problem formulation. Clearly, the operating range over which the reduced model is valid, is influenced by this choice. As the nominal case, values for steam pressures, input moisture, input temperature and dry basis weight corresponding to a typical grade were chosen. Based on the nominal case, additional 35 cases were defined by varying the nominal parameters.

The result of the reduction procedure was evaluated by means of step responses in input moisture, dry basis weight and pressure set point, see Figure 3. As can be seen, there is a good match between the stationary responses of the original and reduced models. Also,

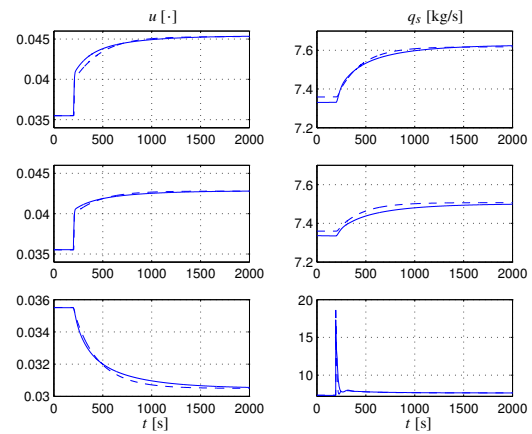


Figure 3: Step responses for moisture (left) and steam flow rate (right) of the original (dashed) and the reduced (solid) models. The responses corresponds to, from above, steps in input moisture, input dry basis weight and pressure reference, applied at 200 s.

the (slow) dominating time constant is captured well by the reduced system. A detailed study of the initial part of the step response reveals that the reduced model does not fully capture the transport delay of the original model. This is, however, to be expected. The original model consists of a large number of paper components, which together forms a high dimension compartment system. The reduced model consists of significantly fewer segments, and cannot approximate the time delay with the same accuracy

The original motivation for performing the model reduction was to obtain a model of lower complexity. Indeed, the reduced model has fewer dynamical states, namely 85, as compared to 318 for the original model. Also, the simulation time for a typical scenario was approximately 85% shorter for the reduced model.

5 NMPC of Output Moisture

Paper moisture is usually controlled using a cascade structure, where the inner loop controls the steam pressures and the outer loop controls the actual moisture. The controllers of the inner loop are commonly PID-controllers, whereas the outer loop is controlled by a Model Based Control (MBC) scheme, e.g. IMC (Internal Model Control), a Dahlin controller, or linear MPC (Model Predictive Control).

The MBC controller is usually based on a low order linear model of the dryer section. While a well tuned controller works well at a given set-point, the non-

linear character of the dryer section dynamics results in degraded performance if the set-point is changed. Since the plant is operated at several different set-points, corresponding to different grades, a traditional control system maintains several parameter sets for the MBC controller. Switching of controller parameters is then done after a grade change.

In this paper, we consider a different approach to moisture control. Based on the reduced non-linear dryer section model derived in Section 4, a basic Non-Linear Model Predictive Control (NMPC) scheme is implemented. The main benefit of using a non-linear model in the control design is that the operating range of the controller may be increased. Also, successful implementation of a controller which achieves good performance in a wide operating range may serve as a unifying strategy for stationary and transition (grade change) control, whereas common practice today is to use separate controllers for these two control modes.

A realistic implementation of an MPC controller consists of three main parts – reference target calculation, state estimation and solution of the optimal control problem. In this paper, the problem of solving the optimal control problem is addressed. The resulting controller is evaluated under the assumption of full state information.

5.1 Model Predictive Control

MPC refers to a family of controllers which are based on the receding horizon principle. At each sample, a finite horizon open loop optimal control problem is solved, and the first part (corresponding to the first sample) of the resulting optimal control profile is applied to the plant. At the next sample, the procedure is repeated and a new optimal control problem with the horizon shifted one sample is solved. Two of the most important advantages of using MPC is that it works well for MIMO plants and that it takes state and control bounds into account *explicitly*. However, an MPC controller, including the on-line solution of an optimization problem (at least in the case of a non-linear model), is computationally demanding, which makes application to processes with fast dynamics troublesome. During the last decade, MPC has emerged as a major control strategy, mainly in the process industry, see [9] for an overview.

5.2 Dynamic Optimization

Traditionally, optimization problems incorporating constraints imposed by dynamic systems have been

addressed by dynamic programming, or the maximum principle. During the last two decades, however, a new family of methods, referred to as direct methods have emerged. These methods are based on discretization of the original optimization formulation, transforming the infinite dimensional problem into a finite dimensional one. The discretized problem is then solved by means of algebraic non-linear programming, see [13] and [3] for two examples of direct methods.

Optimization of Dymola models has previously been considered in the work [7], where the Simulink interface provided with Dymola was used to access the model. The main difference between the approach used in [7] and this work lies in the methods of accessing the model, where the `dsblock` interface has the advantage of offering evaluation of an analytical Jacobian.

The algorithm used to solve the dynamic optimization problem described in this section is a straight forward implementation of a sequential single shooting algorithm, see [13].

5.3 The Optimal Control Problem

An integral part of an NMPC controller is the formulation of the open loop optimal control problem to be solved in each sample. Since the aim of the control scheme in this application is to control the moisture ratio, it is natural to penalize deviations from the target moisture. The control trajectory in the optimization problem is parametrized by a piece-wise constant function with N_u segments. In order to avoid violent control moves, which may introduce disturbances in the steam system, a term penalizing the deviation between two successive control moves is introduced in the cost function. In addition, there are hard limits acting on the control variable. This yields the following optimization problem

$$\min_{\hat{p}_i^{sp}} \int_0^{T_f} \gamma_u (u_{out}^{sp} - \hat{u}_{out}(t))^2 dt + \sum_{i=0}^{N_u-1} \gamma_p (\Delta \hat{p}_i^{sp})^2 \quad (2)$$

subject to

$$F(x, \dot{x}, y, p^{sp}) = 0 \text{ (DAEdynamics)}$$

$$466 \text{ kPa} \leq p^{sp} \leq 596 \text{ kPa} \text{ (control constraint)}$$

where T_f is the prediction horizon, u_{out}^{sp} is the target moisture, $\hat{u}_{out}(t)$ is the predicted moisture profile, \hat{p}_i^{sp} is the predicted pressure set point trajectory and $\Delta \hat{p}_i^{sp} = \hat{p}_i^{sp} - \hat{p}_{i-1}^{sp}$. γ_u and γ_p are weights. In the simulation, the parameters were set to $\gamma_u = 10000$, $\gamma_p = 0.01$, $N_u = 4$ and the sampling interval to 5 s.

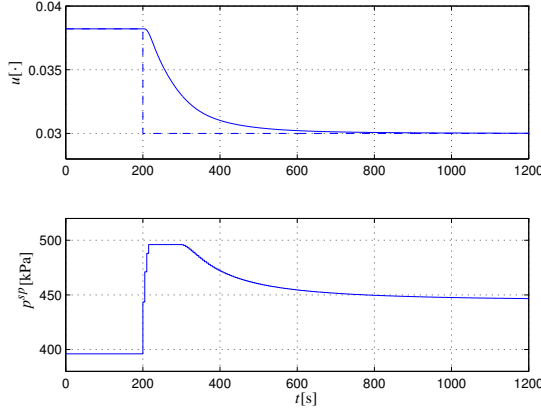


Figure 4: Step response of the NMPC controller.

5.4 Results

A simulation where the NMPC controller is applied to the reduced dryer section model derived in Section 4 is shown in Figure 4. In the simulation, a reference step, from $u_{out}^{sp} = 0.038$ to $u_{out}^{sp} = 0.03$ is applied at $t = 200$ s. As can be seen, the moisture reaches the desired set-point, while the control signal respects the specified constraints.

An important, and often limiting, factor when using MPC controllers, is the execution time for solving the on-line optimization problem. In this case, execution times ranged from 10 s to 80 s, with a mean of 13.5 s. Typically, execution times are longer when reference changes and disturbances occur, while shorter and more predictable execution times are obtained during stationary operation. Assuming a sampling interval of $h = 5$ s, it is clear that the execution times must be decreased. There are several approaches to reducing execution times, e.g. modifying the lengths of the control and prediction horizons, reducing the complexity of the model or using a more efficient optimization algorithm. This is, however, beyond the scope of this paper.

In addition, the problems of reference target calculation and state estimation needs to be solved in order for the control scheme to be useful in practice.

6 Software Tools

The dryer section model has been implemented, as mentioned above, in Modelica and Dymola. The parameter optimization problem and the NMPC problem, however, were solved by integrating several software packages into custom applications, which uti-

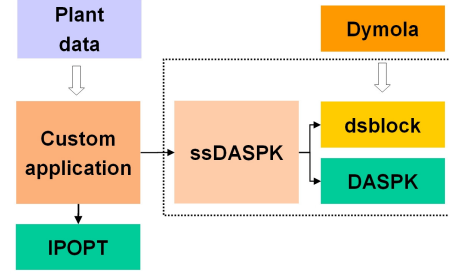


Figure 5: Software application structure.

lized the C-code representing the model generated by Dymola. The results were then fed back to Dymola and verified on the original simulation model.

The software packages used in the development of the custom applications are: *i)* A **C programming interface** which enables access to routines generated by Dymola, `dsblock`. Using this interface, custom applications can be developed for e.g. simulation or like in this case, optimization. The interface provides basic routines for acquiring information about model parameters and initial state, evaluation of the right side of the resulting ODE (DAE) and the associated Jacobian. *ii)* A DAE-solver, **DASPK 3.1** [8]. This code solves DAE:s as well as calculates sensitivities required for optimization. The code is written in Fortran and was translated to C using `f2c`. *iii)* An NLP-code, **IPOPT** [14]. This code implements a primal-dual interior point method and was used to solve the NLP resulting from the parameter optimization and NMPC problems. *iv)* A package for managing the communication between the Dymola C interface and DASPK, which was developed in order to enable convenient development of optimization applications based on models generated by Dymola. This package, referred to as **ssDASPK**, provides e.g. simulation and sensitivity calculation for use in custom applications.

These packages were compiled and linked with the code representing the model generated by Dymola, into applications which was used to set up and solve the particular optimization problems. The structure of the applications is shown in Figure 5.

In addition, AMPL and KNITRO was used to solve the model reduction problem, as described above.

7 Summary and Conclusions

In this paper, modeling, model reduction, parameter optimization and NMPC control design for a paper machine dryer section has been considered. It has been

demonstrated how Modelica models of high complexity can be used for purposes other than simulation. The resulting optimization problems are challenging and require state of the art numerical solvers. In particular, solution of the model reduction problem, which has more than 9000 free variables, is dependent on algorithms exploring the problem structure. Our experience from this project is that there is no single tool or software that can address all problems arising in simulation and optimization. Rather, in order to solve problems effectively, it is essential that Modelica tools are designed to be interfaced with software for solution of optimization problems. In general, it is highly desirable that software for complex systems is provided with interfaces so that they can be combined.

There are several possible extensions of the paper. The `DryLib` library may be extended as outlined in Section 2, and the parameter optimization scheme would benefit from including also time series data. Regarding the model reduction scheme, it may be desirable to derive models with further reduced complexity valid over a wide operating range. Finally, the NMPC scheme outline in Section 5 needs to be further elaborated in order to be applicable to the real plant.

References

- [1] J. Åkesson and J. Ekvall. “Parameter optimization of a paper machine model.” In *Proceedings of Reglermöte 2006*, Stockholm, Sweden, May 2006.
- [2] J. Åkesson and O. Slätteke. “Drylib - a modelica library for paper machine dryer section modeling, and some applications of dynamic optimization.” Technical Report, Department of Automatic Control, Lund University, 2006.
- [3] L. Biegler, A. Cervantes, and A. Wächter. “Advances in simultaneous strategies for dynamic optimization.” *Chemical Engineering Science*, **57**, pp. 575–593, 2002.
- [4] T. Bohlin and A. J. Isaksson. “Grey-box model calibrator and validator.” In *13th IFAC Symposium on System Identification*, Rotterdam, The Netherlands, August 2003.
- [5] J. Ekvall. “Dryer section control in paper machines during web breaks.” Licentiate thesis ISRN LUTFD2/TFRT--3236--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, November 2004.
- [6] R. Fourer, D. Gay, and B. Kernighan. *AMPL – A Modeling Language for Mathematical Programming*. Brooks/Cole — Thomson Learning, 2003.
- [7] R. Franke, M. Rode, and K. Krüger. “On-line optimization of drum boiler startup.” In *Proceedings of Modelica’2003 conference*, 2003.
- [8] T. Maly and L. R. Petzold. “Numerical methods and software for sensitivity analysis of differential-algebraic systems.” *Applied Numerical Mathematics*, **20:1-2**, pp. 57–82, 1996.
- [9] S. J. Qin and T. A. Badgwell. “A survey of industrial model predictive control technology.” *Control Engineering Practice*, **11**, pp. 733–764, 2003.
- [10] M. Rao, Q. Xia, and Y. Ying. *Modeling and Advanced Control for Process Industries – Applications to Paper Making Processes*. Springer-Verlag, New York, 1994.
- [11] O. Slätteke. *Modeling and Control of the Paper Machine Drying Section*. PhD thesis ISRN LUTFD2/TFRT--1075--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, January 2006.
- [12] O. Slätteke and K. J. Åström. “Modeling of a steam heated rotating cylinder - a grey-box approach.” In *Proc. 2005 American Control Conference*, Portland, Oregon, USA, June 2005.
- [13] V. Vassiliadis. *Computational solution of dynamic optimization problem with general differential-algebraic constraints*. PhD thesis, Imperial Collage, London, UK, 1993.
- [14] A. Wächter and L. T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming.” *Mathematical Programming*, **106:1**, pp. 25–58, 2006.
- [15] R. Waltz. *KNITRO User’s Manual*, Version 4.0. Zienna Optimization Inc., 2005.
- [16] B. Wilhelmsson. *An experimental and theoretical study of multi-cylinder paper drying*. PhD thesis, Department of Chemical Engineering, Lund Institute of Technology, Lund, Sweden, 1995.

System and Component Design of Directly Driven Reciprocating Compressors with Modelica

Thomas Bödrich

Dresden University of Technology, Institute of Electromechanical and Electronic Design

01062 Dresden, Germany

Thomas.Boedrich@mailbox.tu-dresden.de

Abstract

Directly driven reciprocating compressors have the potential to be used at large scale for refrigerant compression in domestic refrigerators and freezers and to improve the energy efficiency of these cooling devices considerably. Compared to their well-established conventional counterparts with rotational motor and rotation/translation transducer they offer advantages such as (I) higher efficiency due to the absence of gear friction losses, (II) broad-range modulation of the compressor's mass flow rate, e.g. by control of the piston stroke and (III) a simpler mechanical structure.

With typical strokes of approximately 15...20 mm and piston peak forces up to approximately 150 N, electrodynamic and electromagnetic linear motors are predestined to be used as direct drives for those compressors when operated under resonance conditions with one or more springs. Appropriate motor and system configurations are currently developed and evaluated at Dresden University of Technology. Highly nonlinear compression forces, strong interactions between multiple physical domains and a large number of design parameters to be chosen properly are challenges during system and component design that call for the usage of appropriate models and efficient simulation approaches within the design process. With its multi-domain paradigm, Modelica is excellently suited for the model-based design of directly driven refrigerant compressors, their power supply and control prior to detailed design, manufacture and test of prototypes.

This paper is intended (I) to present an overview of the above compressor technology, (II) to enlighten the beneficial use of Modelica for the design of heterogeneous systems such as the above compressors and (III) to illustrate the design of electro-magneto-mechanical converters by means of lumped magnetic network models and the Modelica Magnetic library.

Keywords: linear compressor; system design; magnetic network; electrodynamic actuator

1 Directly driven reciprocating compressors for refrigeration

1.1 Principle structure and operation

The piston of a directly driven reciprocating compressor is connected to the armature of its driving linear motor without any additional motion transducer or gear element (Fig. 1). This results in greatly reduced friction compared to conventional reciprocating compressors with rotational motor and rotation/translation transducer and hence in a higher overall efficiency of the compressor. In conventional compressors of domestic refrigerators, the motion transducer accounts for approximately 60 % of the total friction losses of the compressor.

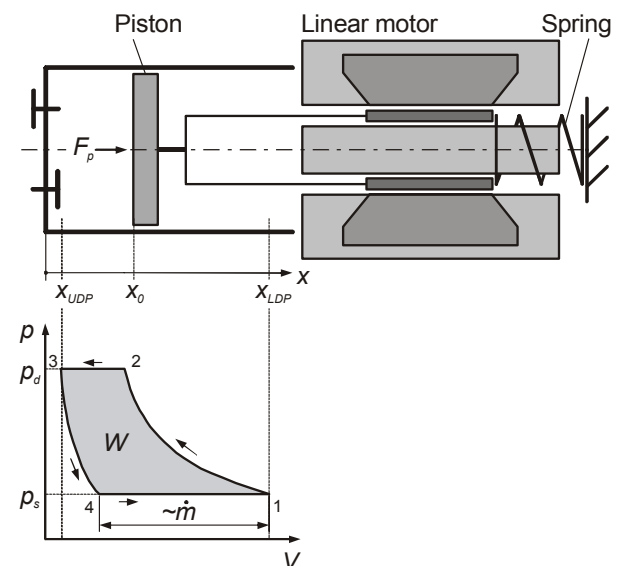


Fig. 1 Principle structure of a directly driven reciprocating compressor (not scaled) and idealised indicator diagram $p(V)$

For reasons of efficiency and size of the linear motor, directly driven reciprocating compressors are normally operated at or close to their mechanical resonance frequency. The mass of the arma-

ture/piston assembly in conjunction with at least one helical or diaphragm spring forms a mechanical resonant system. If diaphragm springs are used, they also guide the armature in radial direction. It must be noted that the compressor load, i.e. the suction pressure p_s and discharge pressure p_d for a particular piston stroke $h = x_{LDP} - x_{UDP}$ influences the resulting net resonance frequency due to its properties of a nonlinear gas spring and hence the overall efficiency at operating frequency. The influence of this gas spring must be taken into account during system design (section 2).

As for conventional reciprocating compressors, an idealised compression cycle of one stage of a compressor consists of four phases (numbers according to idealised indicator diagram of Fig. 1):

- 1→2 isentropic compression,
- 2→3 isobaric discharge,
- 3→4 isentropic re-expansion of the residual gas,
- 4→1 isobaric suction of gas for the next cycle.

In reality, the compression process differs from the above idealised thermodynamic changes of states due to (I) complex heat exchange between the gas and the cylinder assembly, (II) pressure drop and pressure pulsations in the valve assemblies and pipes and (III) deviations of the properties of refrigerants (real gas behaviour) from those of ideal gas as assumed in the above listing of compression cycle phases [1]. The area within the closed pressure-volume curve $p(V)$ of Fig. 1 represents the work per cycle that is exerted on the gaseous refrigerant.

Whereas the stroke of a conventional compressor is constant due to its crankshaft mechanism, the stroke of a directly driven reciprocating compressor can be varied. Thus it is possible to adapt the mass flow rate \dot{m} of directly driven reciprocating compressors in refrigerators to varying thermal operation conditions of the cooling device: \dot{m} is directly proportional to the suction length $l_{suc} = x_{LDP} - x_4$.

1.2 Usage for refrigeration

A successful attempt to utilise an oscillating electromagnetic motor for the direct propulsion of a reciprocating compressor's piston is already reported for the year 1908 [2]. Those attempts continued throughout the last century and resulted in a number of different designs of directly driven reciprocating compressors for different applications [2]. While some of these compressors are experimental prototypes up to now, others are already commercially utilised, e.g. in small air compressors, in small mo-

bile refrigerators for medical and recreational use [3] and in Stirling and pulse tube cryo-coolers.

For refrigerant compressors of domestic refrigerators and freezers however, problems with collisions between the free moving piston and the cylinder head and valve assemblies at varying suction and discharge pressures prevented from the broad utilisation of this concept in the past (beside durability of springs used for resonance operation). To a great extent, these collision problems were due to premature controller realisations. However, situation changed during the last years due to the broad availability of low-cost microcontrollers and power electronic components. This enables now for the realisation of powerful yet cost-effective nonlinear controllers for stroke control and hence for save compressor operation at varying compression loads. For that reason, increased research and design efforts have been made during the last years for the industrial utilisation of directly driven refrigerant compressors. As a result, the Korean manufacturer LG Electronics, Inc. offers domestic refrigerators with directly driven compressors sized for the Korean market since 2004 [4] [5]. The linear motor of these compressors is based on a design of the US company Sunpower, Inc. [6].

1.3 Research objectives and sample application

The current research on directly driven reciprocating compressors for refrigerators at Dresden University of Technology focuses on:

1. analysis of the state of the art of motors for directly driven reciprocating refrigerant compressors and their control,
2. development and evaluation of principle motor designs for a chosen sample application,
3. conceptual control design for compressor operation under varying thermal loads (i.e. cooling powers) for the sample application,
4. establishment of a design methodology for directly driven reciprocating compressors based on modelling and simulation,
5. detailed design, prototype manufacture and test of the linear motor found as optimum under 2. for the sample application.

This sample application is specified as follows:

- domestic one-compartment refrigerator,
- refrigerant: R600 (n-Butane),
- continuous compressor operation, i.e. no intermittent operation for temperature control as in conventional refrigerators,

- specified cooling power for minimum/ nominal/ maximum thermal load: $\dot{Q}_0 = 5/25/50\text{ W}$ with accordingly specified evaporator and condenser temperatures and refrigerant superheat,
- one-stage, one-sided compressor design, i.e. only one compression chamber at one side of the piston,
- compressor operation at mains frequency, i.e. at 50 Hz or 60 Hz,
- adaption to varying cooling power requirements by means of control of the piston's suction length (due to projected continuous operation at constant frequency).

2 Simulation-based system design

2.1 Compressor specification

One of the most important tasks during initial system design is the specification of the compressor's main parameters stroke and compression force versus position from given specifications of the refrigerator's vapour compression process to be realised. This is done by calculation of refrigerant mass flow rates, enthalpy differences and compression work per cycle for specified thermal load scenarios by means of the pressure-enthalpy diagram of the used refrigerant. For the sample design process described in this paper, refrigerant data from an external software package [7] has been used so far for this task. In the future, however, Modelica [8] libraries that are related to thermo-dynamic systems and refrigerant properties such as the commercial AirConditioning LibraryTM [9] and the free Modelica.Media library can be used advantageously, assumed that these libraries are available and contain data for the required refrigerants, respectively. Doing so would incorporate yet another physical domain and model components from its respective Modelica libraries into the resulting system model and would make model-based design of directly driven reciprocating compressors for refrigerators with Modelica even more seamless.

2.2 Dynamic system simulation

Currently, both different principle operating and control concepts and principle motor designs are established and evaluated and the interactions between the major subsystems controller, power supply, linear drive and compression load are designed for the chosen sample application. The elaborated design method is strongly based on dynamic simulations and analysis of the results at steady-state oscillations.

Compared to modelling and design methods based on mechanical and electrical impedances (e.g. [10]), the approach based on transient simulation enables for consideration of all important nonlinear effects in the $p(V)$ relationship and the derived piston force versus position relationship $F_p(x)$. Whereas description of the compression process with an equivalent, stroke dependent spring constant $c_{gas}(h)$ and an equivalent damping coefficient $d_{gas}(h)$ might work for double-sided compressors with one compression chamber at either side of the piston, this approach can not be used for modelling of one-sided compressors as is the case for the chosen sample application. This is due to the shift of the piston's average position x_{avg} away from its rest position x_0 in presence of unbalanced, one-sided compression forces (Fig. 4). This shift of average piston position in turn influences the required stroke that is necessary to realise a desired suction length. Thus it must be considered carefully during conceptual design.

Fig. 2 shows the graphical representation of a Modella model that is used for analysis of fundamental system behaviour and for prediction and evaluation of motor performance to be expected for particular motor designs.

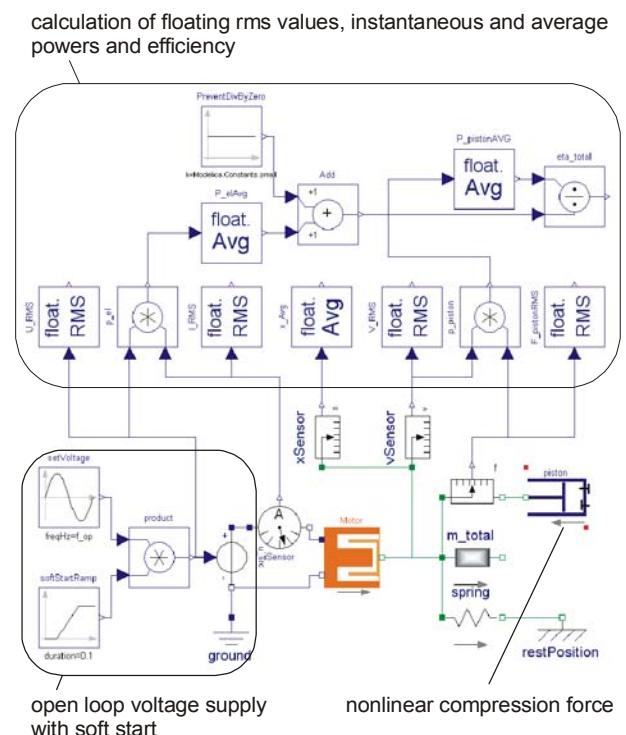


Fig. 2 Graphical representation of the Modelica system model that is used for analysis of fundamental system and motor behaviour

In this model, a sinusoidal voltage is fed to the motor model to be evaluated. This voltage will evoke a cur-

rent i through the motor that causes a motor force F_M to be developed. This force acts against the inertia of the armature/piston assembly's mass m , against the spring(s) used for resonance operation (spring stiffness c) and against the compression force that acts upon the piston F_P with the above mentioned four idealised phases per cycle according to the force equilibrium

$$0 = F_M(i) + m\ddot{x} + c(x - x_0) + F_P(x, x_0, p_s, p_d, n) \quad (1)$$

In the above equation, x denotes the piston position, x_0 its rest position in non-energised state (no pressure difference acting on piston, no supply voltage), p_s and p_d the suction and discharge pressure, respectively, and n the isentropic exponent used for modelling of idealised compression and re-expansion forces according to the isentropic condition $px^n = \text{const.}$

For initial analyses of system behaviour, a simple model of a coarsely dimensioned electrodynamic linear motor was used in the system model depicted in Fig. 2. This motor is of moving coil type. The electro-magneto-mechanical energy conversion process is based on *Lorentz* forces and is described by

$$F_M = Bli \quad (2a)$$

$$u_i = Bl\dot{x} \quad (2b)$$

with F_M being the motor force, u_i the induced counter electromotive force (back-emf), B the flux density imposed to the coil, l the total length of the coil wire and i and \dot{x} current and armature velocity, respectively. In the graphical Modelica representation of the above motor equations shown in Fig. 3, B and l are combined into the motor constant c_m . Besides its back-emf, the electrical subsystem of the motor is made up of its coil resistance R and its inductance L .

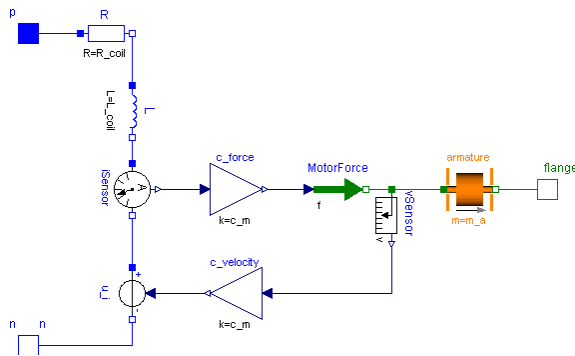


Fig. 3 Graphical Modelica representation of the simple model of an electrodynamic linear motor

It must be pointed out that the above initially used model of a moving coil type linear motor is not the optimum motor design with respect to efficiency, related power and other criteria to be expected. Neither is the used model a detailed one: Assumption of a constant motor constant c_m throughout the complete stroke range is a rather coarse approximation. Also, the motor's magnetic subsystem is not incorporated into the above model as must be done for design of a motor's magnetic components. However, the above simple model is well suited for initial analyses due to its simplicity and linearity. More detailed motor models for additional principle motor designs will be described in section 3.

Using *Kirchhoff's* voltage law, the differential equation of the electrodynamic motor's electrical subsystem can be written as

$$u_{src} = L\dot{i} + Ri + c_m\dot{x} \quad (3)$$

with u_{src} being the supply voltage. Equations 1 and 3 make up the system of nonlinear ordinary differential equations that describes the compressor's dynamic behaviour. Transient simulation of this system will reveal its behaviour with respect to time and enables for derivation of (mostly integral) quantities appropriate for performance evaluation of directly driven reciprocating compressors and their linear motors. For the herewith presented project, *Dymola* is used as simulation environment [11].

Fig. 4 shows exemplary results of a dynamic simulation for a particular motor design and compression load. These results have been obtained with the system model of Fig. 2. A slow initial increase of the supply voltage u_{src} prevents from strong overshoot of the piston oscillation at start-up and hence from piston collisions with the cylinder head and valve assemblies. The fact that the motor voltage u_{src} , the motor's internal back-emf u_i (induced at armature motion) and the voltage drop across the coil resistance $u_R = Ri$ are nearly in phase at steady-state oscillation indicates that the piston oscillates close to its resonance frequency for this particular compression load.

The diagram in the middle of Fig. 4 shows instantaneous and floating root mean square (rms) values of the compression force F_P (index P for piston) and the motor force F_M , respectively. From an energetic point of view it is interesting to note that the motor force $F_{M\text{rms}}$ is smaller than the piston force $F_{P\text{rms}}$ although both motor armature and piston move with the same velocity due to its direct coupling. A detailed investigation shows that this is due to the fact that at piston force maximum (discharge phase) the piston velocity becomes zero (upper dead point posi-

tion). Thinking of the system as being linear with harmonic oscillations would result in a phase angle between piston force and velocity close to 90° and hence in reactive components of the mechanical load. The analysis of both compression power and electrical input power with the presented nonlinear model though shows that the differences in the rms values of motor force and piston force are in accordance with proper physical behaviour (see efficiency calculation, equation 4).

From the bottom diagram of Fig. 4 can be seen that build-up of the piston oscillation from its rest position x_0 in presence of one-sided, i.e. unbalanced compression forces results in a shift of the piston's average position x_{avg} away from this rest position as discussed earlier. In the diagram, the resulting suction length l_{suc} for each cycle is shown, too.

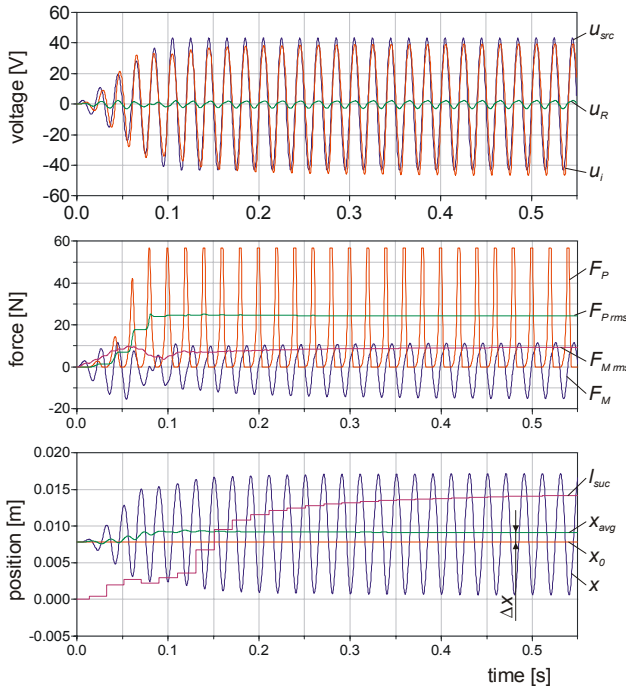


Fig. 4 Selected simulated voltages, forces and positions of an exemplary motor design and compression load

In Fig. 5, the piston force versus position $F_P(x)$ for the simulated compressor start-up shown in Fig. 4 is depicted. Note that the piston's upper and lower dead point positions and suction lengths are different for each cycle prior to achievement of steady-state oscillations.

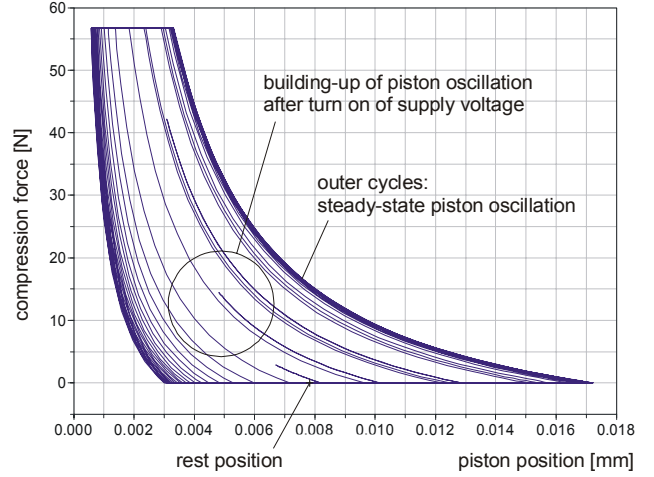


Fig. 5 Simulated piston force vs. position for the compressor start-up shown in Fig. 4

2.3 Performance evaluation

The most important aspects of system behaviour and motor performance that are analysed with the above system model are:

- effect of varying compression loads (suction and discharge pressure, stroke) on net resonance frequency and mechanical and electrical phase relations,
- influence of electrical subsystem on net resonance frequency,
- overall efficiency,
- electrical power factor,
- quality factor of motor (section 3.3).

The above mentioned dependency of the system's mechanical resonance frequency on the motor's electrical parameters coil inductance L and resistance R can be shown with an analysis of the corresponding, but simplified linear 3rd order system in the frequency domain, too. However, the important impact of nonlinear compression forces on system behaviour can not be considered with such an analysis.

The Overall efficiency η of the compressor can conveniently be calculated from floating average values per cycle of compression power and electrical input power:

$$\eta = \frac{P_{c\,avg}}{P_{el\,avg}} = \frac{\frac{1}{T} \int_{t-T}^t F_P(\tau) v(\tau) d\tau}{\frac{1}{T} \int_{t-T}^t i(\tau) u(\tau) d\tau} \quad (4)$$

with T being the period of one cycle, F_p the compression force acting on the piston, v piston velocity and i and u motor current and voltage, respectively. At present, only ohmic losses of the coil are considered for calculation of efficiencies. Hence, the efficiency values shown below are too high compared to reality. Additional losses such as mechanical friction, friction due to gas leakage and ferromagnetic losses in the motor are not yet considered in the above model. This was done intentionally in order to keep the model behaviour straightforward during initial analyses. However, model components for consideration of the above loss mechanisms are currently added to the developed system and motor models and their suitability for system and motor design is tested.

Characteristic values for evaluation of the principle motor designs currently under development are briefly discussed in section 3.3.

2.4 Control concepts

Modulation of the compressor capacity, i.e. adaption of the refrigerant mass flow rate to the requested cooling power of the refrigerator, is possible by one or more of the following means:

- variation of suction length and associated stroke,
- variation of operating frequency and
- intermittent operation with varying duty cycle (as for conventional compressors with crankshaft mechanism and constant speed).

Since continuous operation at constant frequency is projected for the chosen sample application, only suction length control is applicable. Suction length can not directly be measured without large effort, therefore measures are taken to estimate the suction length of each cycle from the piston stroke and to realise a stroke control instead.

By now, different control concepts and appropriate controllers have been developed, implemented in Modelica and successfully tested with dynamic simulations at system level. The following control concepts are currently under closer investigation:

- stroke control only without control of top dead center position (guarantees optimum motor efficiency since no DC bias voltage is superimposed on the controlled AC supply voltage),
- combined stroke control and control of top dead center position (guarantees minimum clearance volume and related thermo-dynamic losses but increases ohmic losses in the motor due to a DC

bias voltage to be superimposed on the controlled AC supply voltage),

- various additional control concepts that do not require measured or estimated information of the piston stroke, but information on top dead center position only (e.g. obtained from a proximity sensor).

The performance of the above control concepts with respect to overall and motor efficiency, electrical power factor and phase relations is currently investigated with the developed system models. The stroke controllers are mostly of proportional and integral type (PI). For the test of principle voltage supply concepts, different models have been developed and are currently tested (e.g. thyristor and triac control for simple and cost-effective mains operation).

2.5 Simulation-based design

The above mentioned controller models for stroke control are not only used for simulation and test of concepts for capacity modulation of the compressor, but also for simulation-based selection of fundamental design parameters. This is because stable piston oscillations with constant stroke can be guaranteed during dynamic simulation with closed loop stroke control, but not during dynamic simulation of large strokes with open loop voltage supply only due to the nonlinear compression forces.

A good example for the simulation-based selection of a design parameter is the stiffness of the spring(s) needed for resonant piston oscillation (Fig. 6).

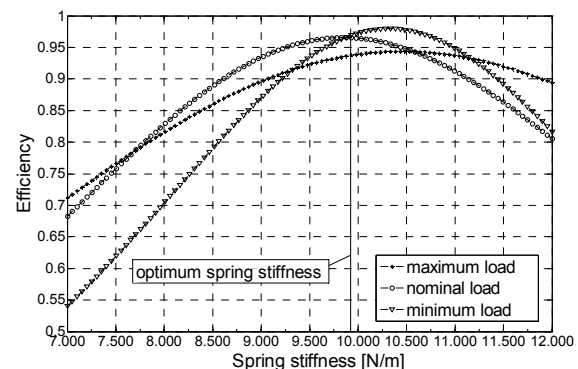


Fig. 6 Simulation-based selection of optimum spring stiffness obtained with parameter sweeps for this design parameter (only ohmic losses in the motor coil considered for efficiency calculation, see text)

The optimal required spring stiffness c_{spring} for a particular design can not be obtained otherwise with the same certainty. This is because of the properties of the compressed gas that resemble that of a strongly

nonlinear gas spring (section 2.2). Its stroke-, position-, and pressure-dependent stiffness c_{gas} adds to the mechanical spring stiffness and affects the net resonance frequency f_{res} of the mechanical subsystem with the moving mass m :

$$f_{res} = \frac{1}{2\pi} \sqrt{\frac{c_{spring} + c_{gas}}{m}} \quad (5)$$

Each efficiency value shown in Fig. 8 was obtained from a steady-state efficiency calculation at the end of respective dynamic simulations during parameter sweeps for the spring stiffness c_{spring} . Stroke control was implemented in the Modelica model during these simulations, so the desired stroke length for each particular compression load was assured.

Notice that the calculated efficiency values are too high compared to reality since only ohmic losses in the motor coil were considered during these simulations as described in section 2.3.

3 Magnetic converter design

3.1 Design Methodology

An appropriate approach for the motor design for directly driven reciprocating compressors can roughly be outlined as follows:

1. Specification of mechanical motor parameters (stroke, rms force, average power) from compression data (idealised indicator diagram, compression work per cycle) for specified compression load scenarios (nominal and maximum load),
2. Specification of additional motor parameters, e.g. max. ambient temperature within compressor capsule, volume, refrigerator supply voltage and frequency,
3. Selection of promising principle motor designs, coarse dimensioning of these principle solutions (geometry and material of magnetic components, coil and wire data) by means of lumped magnetic networks,
4. evaluation of the found principle solutions by means of characteristic factors (section 3.3) and dynamic simulation at system level (sections 2.2, 2.3), selection of the optimum solution,
5. fine dimensioning of the selected motor's magnetic design with Finite Element Analysis (FEA),
6. detailed design, manufacture and test of prototypes.

Whereas the use of FEA is valuable for optimisation of a particular motor's magnetic design, it is in most cases not suited for efficient coarse dimensioning of principle motor designs and for usage in extensive dynamic simulations due to the high time effort for model pre-processing and due to its high computational effort, respectively. Instead, lumped magnetic networks should be used for initial coarse designs. For the motor design within this project, the Modelica Magnetic library developed by the author is used for the implementation of magnetic network models of the motors [12]. Through its usage within this design project, the library is extended, e.g. with models for electrodynamic linear actuators, and its model components are enhanced. It is planned to submit a revised version of this library as soon as possible.

3.2 Principle motor designs

Different principle motor designs are already known for directly driven reciprocating compressors. They are a subset of the broad spectrum of different types and designs of electrodynamic and electromagnetic linear drives. In general, numerous classification criteria are popular for classification of electromagneto-mechanical converters. For example, based on the underlying physical principle one can distinguish between:

- electrodynamic linear drives based on *Lorentz* forces [13],
- electromagnetic or reluctance drives based on surface forces between ferromagnetic component(s) and adjacent air gap(s) [14] and
- linear drives that utilise both of the above working principles for force generation.

Depending on the moving component, classification between the following motor categories is popular:

- moving coil (mostly for electrodynamic drives),
- moving magnet (electrodynamic and/or reluctance forces) and
- moving iron (reluctance drives).

Location of stator and armature components is a classification criterion, too.

At present, known motor designs are analysed and additional principle solutions (a promising subset of motors from the above categories) are developed. All found solutions are evaluated. The required coarse dimensioning for the sample application is mostly done with the above mentioned lumped magnetic network models. Despite this efficient design approach, coarse dimensioning of motors based on different working principles is a rather time-consuming

process. Nevertheless, it is a prerequisite for optimum motor design for a particular application.

Although the above mentioned evaluation is not yet completed, it currently appears that the motor concept realised by Sunpower, Inc. [6] and already utilised in a commercial domestic refrigerator of LG Electronics, Inc. [4] [5] is an advantageous one. The principle structure of this motor is shown in Fig. 7 together with the magnetic field lines of the permanent magnetic flux in neutral position. Latter were obtained from FEA for the shown exemplary design. Operation of this motor is based on the superposition of permanent magnetic flux and electromagnetic coil flux in the respective air gap sections and results in a constant thrust over the complete stroke range. This thrust is proportional to the current. Hence motor operation can be described with equations 2a and 2b as for a purely electrodynamic linear motor. Based on that perception, the creators of this motor promote the treatment of the permanent magnet's magnetomotive force as equivalent currents at the two end planes of the permanent magnet's hollow cylinder [15]. Whereas this design approach enables for easy calculation of motor forces, it does not account for the permanent magnetic flux through the stator at off-center armature positions and for related saturation effects in the ferromagnetic stator components properly.

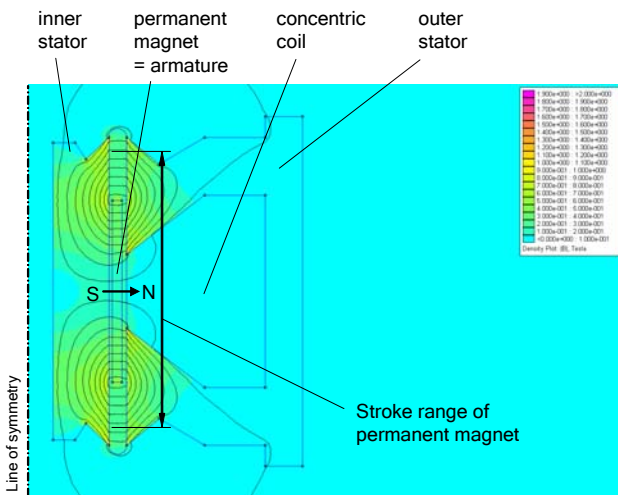


Fig. 7 Structure of a moving magnet motor based on [6] and permanent magnetic flux without stator current obtained by FEA

Unlike with the above modelling approach, the stator flux due to the permanent magnet at off-center armature positions is treated properly with the lumped magnetic network model shown in Fig. 8. It was created with model components from the Modelica Magnetic library [12]. The depicted reluctance elements

represent the respective air gap and permanent magnet regions at either of the two pole regions. The motor force F_M is developed according to

$$F_M = -\frac{1}{2} \sum_{i=1}^{n_{linear}} \Phi_i^2 \frac{dR_{mi}}{dx} \quad (6)$$

with n_{linear} denoting the number of linear reluctance elements, Φ_i the magnetic flux through each respective reluctance and R_{mi}/dx the derivative of each reluctance with respect to armature position x [14].

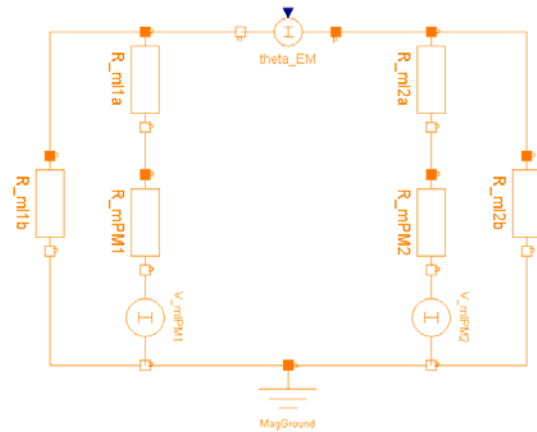


Fig. 8 Simple lumped magnetic network model of the moving magnet motor created with elements from the Magnetic library [12]

In the linear model of Fig. 8, nonlinear ferromagnetic reluctance elements of inner and outer stator components are omitted for reasons of initial simplicity. Also, the depicted model is only a stationary one. It was used for initial analyses of magnetic flux distributions at different armature positions and motor currents. The electro-magnetic energy conversion that is important for dynamic operation is not considered in this model. However, both nonlinear ferromagnetic reluctance elements and a dynamic model of the stator coil that describes the coupling between the motor's electrical and its magnetic domain can easily be added to the motor model from standard components of the Modelica Magnetic library (see examples in this library). Latter enhancement simply requires replacement of the coil's stationary magnetomotive force θ_{EM} of Fig. 8 with a dynamic model of a coil. The resulting dynamic motor model can then directly be used for dynamic system simulation, e.g. with the model shown in Fig. 2. This has already been done for magnetic network models of different moving coil motor designs and will be done for the additional motor designs currently under development, too.

3.3 Motor evaluation

For comparison of the found principle motor designs, the performance to be expected is currently evaluated by means of the developed motor models. Quantitative factors for this motor evaluation are:

- motor efficiency for specified minimum/ nominal/ maximum compression load,
- quality factor $E = F_M^2 / P_{el}$ as important measure for the sensitivity of electro-magneto-mechanical converters (F_M motor force, P_{el} electrical input power) [16],
- mass of selected components (permanent magnet, copper, iron) as measure for the material cost to be expected,
- related power, i.e. power per volume and
- motor constant c_m for calculation of thrust from motor current.

Additional qualitative criteria like the presence of parasitic radial magnetic forces or durability of the design (e.g. important for flexible wires to moving coils) shall be considered in this evaluation, too. This evaluation will be based on guidelines provided in [17].

4 Summary and outlook

The proper design of directly driven reciprocating compressors and their respective electrodynamic and electromagnetic motors is challenging due to nonlinear compression forces, strong interactions between the different subsystems and complex phase relations and resonance phenomena at varying compression loads. Modelica is excellently suited for consistent modelling of those heterogenous systems and the developed models can advantageously be used for simulation-based system and component design.

Development of compressor operation and control principles as well as of appropriate motor designs is currently done for a chosen sample application, namely for a refrigerant compressor of a domestic refrigerator. The elaborated design approach based on dynamic system simulation and analysis of integral quantities at steady-state piston oscillations proved to be well-suited for the rather complex system design. Although the time effort for initial model development is relatively high, the results obtained from dynamic simulations are extremely valuable for the search of appropriate system and component configurations and for understanding of the complex dynamic behaviour of directly driven reciprocating compressors. With the developed models, the ongoing simulation-based evaluation of different prin-

ciple motor designs for the chosen sample application will be completed prior to the intended test of first experimental motor prototypes.

Evaluation of the suitability of different motor principles for the chosen sample application requires the coarse dimensioning of respective motors. The usage of lumped magnetic network models built with the Modelica Magnetic library is an efficient means for this rather laborious design task.

It is worth to note that the developed model-based design approach for directly driven reciprocating compressors is not restricted to refrigerant compressors but can be adapted to different fields of applications, too.

References

- [1] Frenkel, M.I.: Kolbenverdichter: Theorie, Konstruktion und Projektierung. Berlin: Verlag Technik 1969
- [2] Kudrauskas, S.; Didziokas, R.; Simanyniene, L.: Innovative Free-piston Compressor Based on Generalized Conception of Compressors Driven by an Oscillating Electrical Motor. Proc. of International Compressor Engineering Conference at Purdue, West Lafayette, USA, July 17-20, 2006
- [3] Sawafuji Electronic Co., Ltd.: Home page Engel Swing Compressor. http://www.sawafuji.co.jp/engel/inde_e.htm, (18.05.2006)
- [4] Park, K.; Hong, E.; Lee, H.-K.: Linear Motor for Linear Compressor. Proc. of International Compressor Engineering Conference at Purdue, West Lafayette, USA, 16.-19.07.2002, pp. 337-342
- [5] LG Electronics, Inc.: Product home page refrigerator. http://www.lge.com/products/category/list/home%20appliances_refrigerator.jhtml, (18.05.2006)
- [6] Redlich, R.W. (Sunpower, Inc.): Electromechanical transducer particularly suitable for a linear alternator driven by a free-piston Stirling engine. US 4,602,174, 1986
- [7] CoolPack (ver. 1.46): Technical University of Denmark, Department of Mechanical Engineering, 2001, www.et.du.dk/CoolPack
- [8] Modelica Association: Modelica home page. <http://www.modelica.org>, (24.07.2006)

- [9] Modelon AB: AirConditioning Library.
<http://www.modelon.se/prod.htm>,
(13.07.2006)
- [10] Kudarauskas, S.; Simanyniene, L.; Guseinoviene, E.: Analysis of Operation of Piston Compressor Driven by Oscillating Electrical Motor. Proc. of International Compressor Engineering Conference at Purdue, West Lafayette, USA, July 16-19, 2002, pp. 869-876
- [11] Dymola (ver. 6): Dynasim AB, 2006,
<http://www.dynasim.se>
- [12] Bödrich, T.; Roschke, T.: A Magnetic Library for Modelica. Proc. of 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005, pp. 559-565
- [13] Stölting, H.-D.; Kallenbach, E. (Ed.): Handbuch Elektrische Kleinantriebe. 2nd ed., München, Wien: Hanser 2002
- [14] Kallenbach, E.; Eick, R.; Quendt, P. et. al.: Elektromagnete: Grundlagen, Berechnung, Entwurf und Anwendung. 2nd ed., Wiesbaden: B.G. Teubner 2003
- [15] Redlich, R.; Unger, R.; Van der Walt, N.: Linear Compressors: Motor Configurations, Modulation and Systems. Proc. of International Compressor Engineering Conference at Purdue, West Lafayette, USA, July 23-26, 1996
- [16] Lenk, A.; Pfeifer, G.; Werthschützky, R.: Elektromechanische Systeme - Mechanische und akustische Netzwerke, deren Wechselwirkungen und Anwendungen. Berlin, Heidelberg: Springer 2001
- [17] VDI 2225 Blatt 3 (1998): Konstruktionsmethodik - Technisch-wirtschaftliches Konstruieren - Technisch-wirtschaftliche Bewertung. Düsseldorf: Beuth-Verlag

Multizone Airflow Model in Modelica

Michael Wetter
 United Technologies Research Center
 411 Silver Lane
 East Hartford, CT 06108

Abstract

We present the implementation of a library of multizone airflow models in Modelica and a comparative model validation with CONTAM. Our models have a similar level of detail as the models in CONTAM and COMIS. The multizone airflow models allow modeling the flow between rooms through doors, staircases or construction cracks. The flow can be caused by buoyancy effects, such as stack effects in high rise buildings or air temperature imbalance between adjoining rooms, by flow imbalance of a ventilation system, or by wind pressure on the building envelope.

The here presented library can be used with a Modelica library for thermal building and HVAC system simulation to compute interzonal air flow rates. The combined use facilitates the integrated design of building systems, which is typically required for analyzing the interaction of room control loops in variable air volume flow systems through open doors, the flow in naturally ventilated buildings and the pressure in elevator shafts caused by stack effects.

Keywords: Multizone Airflow, Contaminant Transport, Stack Effect

1 Introduction

We present the implementation of a library of multizone airflow models in Modelica and a comparative model validation in which we computed the buoyancy driven air flow rates in CONTAM and in Modelica for a two storey building. The implemented models have a similar level of detail as the models in the state-of-the-art multizone airflow programs CONTAM [4] and COMIS [6]. Multizone airflow models consist of nodes that are connected by flow elements. The nodes may represent room air volumes, the exterior environment or connections in a duct system and contain state variables, typically pressure, temperature and concentrations such as water vapor, CO₂, smoke or pollutants.

The flow elements are airflow paths such as open doors and windows, construction cracks, stair cases, elevator shafts, ducts and fans.

Multizone airflow models are typically used for time domain simulation of convective energy and contaminant transport between thermal zones of a building and to quantify stack effects in high rise buildings. For thermal building simulation, the closed door and user-estimated airflows that are common practice in most multizone simulations are a poor representation of reality [8, 18]. Assessing the convective energy and species transport can also be important for controls analysis, since the convective transport through open doors can couple one local control loop with another. This situation may occur, for example, if two rooms have their own local VAV damper control loop that causes pressure or temperature imbalance if one damper closes. To model such phenomena, the here presented airflow models can be used to define flow paths that connect different thermal zones of a building that may be presented by the model described in [23]. Multizone airflow models represent the room volumes assuming uniform distribution of temperature, pressure and species concentration throughout the room as opposed to the spatially more refined resolution of zonal models [10] or computational fluid dynamics models. The reduced number of state variables of multizone airflow models makes it computationally feasible to perform annual simulations of heat and species transport in buildings.

In [7], Feustel and Dieris present an extensive literature review and questionnaire survey of more than 50 multizone airflow models and list the main equations used in those models. Today, two multizone airflow models are well established: CONTAM, developed by the National Institute of Standards and Technology (NIST) and COMIS, developed in 1988-89 in an international context within IEA Annex 23 at the Lawrence Berkeley National Laboratory (LBNL). COMIS has been interfaced with the thermal build-

ing energy simulation program EnergyPlus [9]. EnergyPlus uses airflow rates that were computed by COMIS in the previous time step. This coupling with time lagging, however, can cause instability if the time step is too large [18]. COMIS and CONTAM have both been implemented in TRNSYS in the form of a TRNSYS TYPE (i.e., a component model) that can be interfaced with TRNSYS' multizone thermal building model TYPE 56 [15]. In subsequent work, COMIS has also been integrated directly into TYPE 56 due to convergence problems and to improve the user interface [22]. The thermal building simulation programs IDA [19] and ESP-r [3, 8] have models for interzonal air exchange for bi-directional flow through open doors. The basic flow resistances used in COMIS, CONTAM, IDA and ESP-r are powerlaw equations that compute the steady-state flow as a function of the static pressure difference.

Lorenzetti [12] reports that some flow models in COMIS and CONTAM can give non-repeatable results due to memory in the calculation routines. For example, the crack model in COMIS finds the density of the air in the crack using the flow calculated in the last evaluation of the crack in question. This can cause the solver to fail converging to a solution. Another concern is that COMIS does not solve the mechanical energy balance and the friction models simultaneously which can yield to errors of around 30% for flows through passive vertical shafts for stairways, elevators or natural ventilation. Lorenzetti concludes that due to the decoupling of the mechanical energy balance and the friction models, COMIS cannot implement any meaningful model for bi-directional flow between floors of a building.

2 Physical Model Description

We use a powerlaw relationship to model the volume flow rate through an aperture as a function of the static pressure difference ΔP between two rooms. Let $\Delta P_{0,\varepsilon} \ll 1$ be a user specified parameter. The governing equations are

$$\dot{V} = \begin{cases} k\Delta P^m, & \text{for } \Delta P \geq 3/2\Delta P_{0,\varepsilon}, \\ -k(-\Delta P)^m, & \text{for } \Delta P \leq -3/2\Delta P_{0,\varepsilon}, \end{cases} \quad (1a)$$

where \dot{V} is the volume flow rate, k is a flow characteristics, ΔP is the static pressure difference over the aperture and $m \in [0.5, 1]$ is a flow exponent, with $m = 0.5$ for turbulent flow and $m = 1$ for laminar flow. For $m < 1$, the derivative $d\dot{V}/d\Delta P$ tends to infinity,

as $\Delta P \rightarrow 0$, which can cause problems for numerical solvers. Thus, we use for small pressure differences the linear function

$$\dot{V} = k\Delta P_{0,\varepsilon}^{m-1} \Delta P, \quad \Delta P \in [-\Delta P_{0,\varepsilon}/2, \Delta P_{0,\varepsilon}/2]. \quad (1b)$$

The transition between equation (1a) and (1b) is done in the intervals $\Delta P \in (-3/2\Delta P_{0,\varepsilon}, -1/2\Delta P_{0,\varepsilon})$ and $\Delta P \in (1/2\Delta P_{0,\varepsilon}, 3/2\Delta P_{0,\varepsilon})$ using the built-in Dymola function `spliceFunction`, which defines $\dot{V}(\cdot)$ as a function that is once continuously differentiable in ΔP . Linearization around zero is also used in [4, 2].

We observed good numerical experiments by setting $\Delta P_{0,\varepsilon} = 0.1$ Pa. Using $\Delta P_{0,\varepsilon} = 0.001$ Pa caused the time integration solver in some examples to jam at state events caused by a buoyancy driven flow reversal.

We will now discuss the implemented models, which all compute the flow based on (1).

2.1 Orifice

The flow through an orifice with cross section area A can be deduced from the Bernoulli equation and is given by

$$\dot{V} = C_d A \sqrt{2/\rho} \Delta P^m, \quad (2)$$

where $C_d \in (0, 1)$ is a dimensionless discharge coefficient and ρ is the density of the medium that flows through the orifice. Large openings are characterized by m very close to 0.5, while values near 0.65 have been found for small crack-like openings [21, 4].

In [20], van der Mass et al. present a brief literature review about discharge coefficients and conducted own experiments on a 1:10 and a full scale model. They concluded that C_d can confidently be chosen in the range of 0.6 to 0.75, provided that the temperature stratification in the rooms is taken into account (and hence \dot{V} is obtained using the integration $\dot{V} = \int (d\dot{V}/dh) dh$ over the aperture height.) They report that $C_d = 0.61$ is widely used, and that Mahajan [14, 13] found values as low as 0.33 for isothermal rooms.¹ Van der Mass et al. also report that Riffat [17] found for the transport through both a door and a staircase a value of $C_d = 0.6$ for an interzonal air temperature difference of $\Delta T = 1$ K, decreasing to $C_d = 0.25$ for $\Delta T = 10$ K, which is in contrast to Kiel and Wilson [11], which obtained in full scale experiments $C_d = 0.40 + 0.0045 \Delta T$.

¹Two rooms are said to be isothermal if the temperature difference between the rooms is much larger than the vertical temperature difference between floor and ceiling in each room.

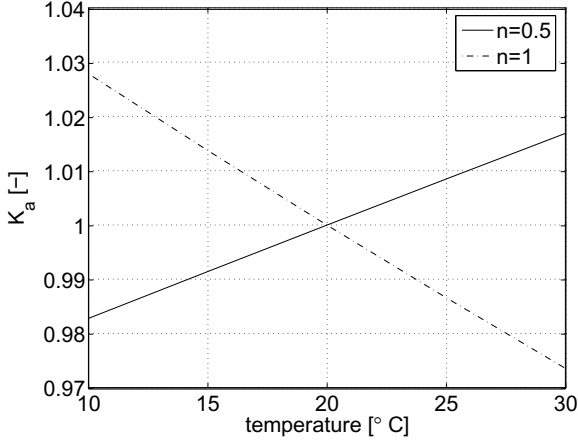


Figure 1: Correction factor used in CONTAM for change of volume flow rate as a function of temperature.

To account for the temperature dependency of the air density ρ and the kinematic viscosity ν , CONTAM multiplies the right hand side of the orifice equation (2) by the correction factor

$$K_a(P, T) = \left(\frac{\rho_0}{\rho(P, T)} \right)^n \left(\frac{\nu_0}{\nu(P, T)} \right)^{2n-1}, \quad (3)$$

with $\rho_0 = 1.2041 \text{ kg/m}^3$, $\nu_0 = 1.5083 \cdot 10^{-5} \text{ m}^2/\text{s}$, $\rho(P, T) = P/(287.055 T)$, $\mu(T) = 3.7143 \cdot 10^{-6} + 4.9286 \cdot 10^{-8} T$ and $\nu(P, T) = \mu(T)/\rho(P, T)$. Fig. 1 shows $K_a(P, T)$ for the limiting cases $n=0.5$ and $n=1$ with $P = 101325 \text{ Pa}$. Since K_a is close to one for the expected temperature range, we will use $K_a = 1$ for all temperatures. Furthermore, the air density in (2) should be evaluated at the temperature and pressure of the medium that flows through the orifice. However, if the air temperature is different on either side of the orifice, then the derivative $d\dot{V}/d\Delta P$ of (1b) is discontinuous at $\Delta P = 0$. We prevent this discontinuity by setting $\tilde{\rho} = 1.2 \text{ kg/m}^3$ in the orifice equation (2), independent of the temperature and pressure. Thus, we implemented (2) in Modelica using the powerlaw (1) with

$$k = C_{d,r} A \sqrt{2/\tilde{\rho}}. \quad (4)$$

2.2 Effective Air Leakage Area

ASHRAE Fundamentals [1, p. 25.18] and Clarke [3] list effective air leakage areas for various building construction elements, such as doors, windows and vents. The effective air leakage area tabulated in [1] are based on a reference pressure difference $\Delta P_r = 4 \text{ Pa}$ and a discharge coefficient $C_{d,r} = 1$, but other data sets use $\Delta P_r = 10 \text{ Pa}$ and $C_{d,r} = 0.6$ [4].

We convert the effective air leakage area to the orifice equation (2) as follows. Let L denote the effective air leakage area with units m^2 . The area A in the orifice equation (2) is obtained by equating (2) with

$$\dot{V} = C_{d,r} L \sqrt{2\Delta P_r/\rho}, \quad (5)$$

from which follows that

$$A = \frac{C_d}{C_{d,r}} L \Delta P_r^{0.5-m}. \quad (6)$$

If no value of m is reported with the test results, then Dols and Walton [4] recommend $m = 0.6$ to $m = 0.7$.

2.3 Large Vertical Open Aperture

We will now present a model for bi-directional airflow at steady state conditions through a large aperture. The model can be used to compute the bi-directional airflow through open doors or windows. We assume in this section that the aperture is always open. In Sec. 2.4, we extend the model in such a way that the aperture can also be closed.

2.3.1 Governing Equations

To deduce the model, we consider an aperture with height h and width w that separates two air volumes (see also Fig. 2). Let z denote the height coordinate, defined so that $z = 0$ at the bottom of the aperture. Let \bar{P}_A and \bar{P}_B denote the static pressures, let \bar{T}_A and \bar{T}_B denote the temperature and let $\bar{\rho}_A$ and $\bar{\rho}_B$ denote the air density in the two air volumes at height z_A and z_B . Let $T'_A \triangleq dT_A(z)/dz$ and $T'_B \triangleq dT_B(z)/dz$ denote the vertical temperature gradient in each volume which is assumed to be independent of z . We approximate the local air density in volume A at a height z above the bottom of the aperture as

$$\rho_A(z) = \bar{\rho}_A \left(1 - \frac{(z - z_A) T'_A}{\bar{T}_A} \right). \quad (7)$$

The static pressure at height z in volume A is

$$P_A(z) = \bar{P}_A + g \int_0^{z_A} \rho_A(s) ds - g \int_0^z \rho_A(s) ds, \quad (8)$$

where $g = 9.81 \text{ m/s}^2$ is the earth acceleration. Substituting (7) in (8) and evaluating the integrals yields

$$P_A(z) = \bar{P}_A + g \bar{\rho}_A (z_A - z) + g \bar{\rho}_A \frac{T'_A}{2 \bar{T}_A} (z^2 - 2 z z_A + z_A^2). \quad (9)$$

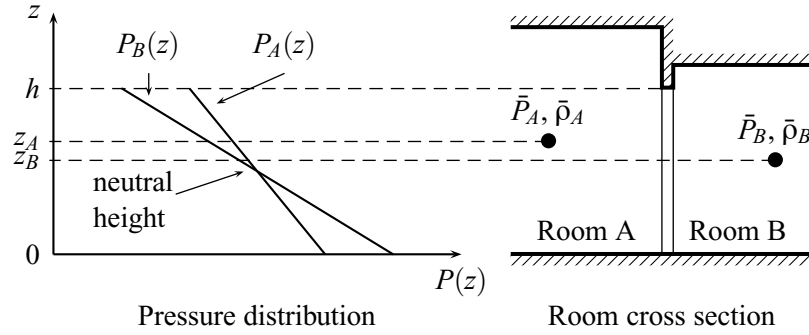


Figure 2: Schematics of two rooms with different heights that are connected by a large vertical opening of height h . The left-hand graphics shows schematically the static pressures $P_A(\cdot)$ and $P_B(\cdot)$ as a function of the height z for the situation where $\bar{P}_A < \bar{P}_B$, $\bar{\rho}_A < \bar{\rho}_B$ and without vertical temperature gradient in the rooms. The neutral height is at the point where $P_A(z) = P_B(z)$.

An identical deviation can be done for the static pressure $P_B(z)$. Thus, the static pressure difference at height z is

$$\begin{aligned} \Delta P_{AB}(z) &\triangleq P_A(z) - P_B(z) \\ &= \bar{P}_A - \bar{P}_B + g(\bar{\rho}_A(z_A - z) - \bar{\rho}_B(z_B - z)) \\ &\quad + g\bar{\rho}_A \frac{T'_A}{2\bar{T}_A} (z^2 - 2zz_A + z_A^2) \\ &\quad - g\bar{\rho}_B \frac{T'_B}{2\bar{T}_B} (z^2 - 2zz_B + z_B^2). \end{aligned} \quad (10)$$

Equation 10 is a quadratic equation in z . Its solutions $z^* \in [0, h]$ that satisfy $\Delta P_{AB}(z^*) = 0$, if they exist, are called the neutral heights. At the neutral heights, there is no flow through the aperture.

The volume flow rate from A to B through the height element dz is for $P_{AB}^m(z) > 0$, $\dot{V}_{AB}(z)/dz = C_d \sqrt{2/\rho_A(z)} \Delta P_{AB}^m(z) w dz$ or else $\dot{V}_{AB}(z)/dz = -C_d \sqrt{2/\rho_B(z)} (-\Delta P_{AB}(z))^m w dz$.

The net flow from A to B is

$$\dot{V}_{AB} = \int_0^h \max(0, d\dot{V}_{AB}(z)/dz) dz \quad (11)$$

and similarly

$$\dot{V}_{BA} = \int_0^h \min(0, d\dot{V}_{AB}(z)/dz) dz. \quad (12)$$

The exact evaluation of the integrals in (11) and in (12) requires the knowledge of the neutral heights. In the absence of vertical temperature gradients, i.e., if $T'_A = T'_B = 0$, the neutral height is

$$z^* = \frac{\bar{P}_A - \bar{P}_B + g(\bar{\rho}_A z_A - \bar{\rho}_B z_B)}{g(\bar{\rho}_A - \bar{\rho}_B)}. \quad (13)$$

CONTAM and IDA neglect vertical temperature gradients and evaluate symbolically the integrals (11)

and (12). Furthermore, in computing the pressure difference across the opening, CONTAM assumes the vertical temperature profile to be constant [4, p. 152], which facilitates in view of the denominator of (13) the numerical solution if $\bar{\rho}_A$ and $\bar{\rho}_B$ are close to each other. In COMIS [6], however, the opening is discretized along the height coordinate to eliminate the need for solving the pressure difference equation (10) for the neutral height. The total flow is then obtained by summation of the flows for the whole opening.

2.3.2 Model Discretization

As in COMIS, we also select a discretized model to implement an approximate solution of (11) and (12) since we expect the discretized model to be numerically more robust than an implementation of the analytical solution of (11) and (12) which requires solving (10) for the neutral height which may have zero, one or two solutions in $[0, h]$.

For a fixed $n \in \mathbb{N}$, we discretize the height h of the door into n compartments of equal height $\Delta h = h/n$. For $i \in \{1, \dots, n\}$, we define $z_n^i \triangleq h(i - 1/2)/n$. We use the orifice equation (2) to compute the flow in each compartment. For sufficiently large positive pressure differences, the orifice equation in each compartment is

$$\dot{V}_{AB,n}^i = C_d w \frac{h}{n} \sqrt{\frac{2}{\rho_A(z_n^i)}} \Delta P_{AB}^m(z_n^i), \quad (14)$$

where $\Delta P_{AB}(z_n^i)$ is defined in (10). We implement the orifice equation using the powerlaw relation (1) with

$$k = C_d w \frac{h}{n} \sqrt{\frac{2}{\bar{\rho}}}, \quad (15)$$

where $\hat{\rho} \triangleq (\bar{\rho}_A + \bar{\rho}_B)/2$. In (15), we used the average

of the density at the two reference points A and B to approximate the density in dz . As mentioned earlier, the density in dz depends on the flow direction, but using the density of the inflowing medium would cause the volume flow rate to be non-differentiable with respect to the pressure difference at $\Delta P_{AB}^m(z) = 0$. In this model, we know the density on both sides of the aperture and hence use their average value rather than a constant density. In the current implementation, we neglect the vertical temperature gradient in volume A and B and set $T'_A = T'_B = 0$. The model could be extended with little effort to take into account a temperature gradient by adding the quadratic terms in the implementation of the pressure difference equation (10). Extending the model may be important if a room has a displacement ventilation, since vertical temperature gradients can have a major influence on the heat exchange by interzonal airflows [8].

In order to formulate a once continuously differentiable approximation to the $\max(\cdot, \cdot)$ and $\min(\cdot, \cdot)$ functions in (11) and in (12), we compute the average velocity in each compartment as

$$v_n^i \triangleq \frac{\dot{V}_{AB,n}^i}{wh/n}. \quad (16)$$

The net flow from A to B and from B to A through each element is computed as

$$\vec{V}_{AB,n}^i = \dot{V}_{AB,n}^i \tilde{H}(v_n^i; v_\epsilon), \quad (17a)$$

$$\vec{V}_{BA,n}^i = -\dot{V}_{AB,n}^i + \vec{V}_{AB,n}^i, \quad (17b)$$

where $\tilde{H}(\cdot; v_\epsilon)$ is a once continuously differentiable approximation to the Heaviside function, implemented using the Dymola built-in function `spliceFunction`, parametrized by a user specified smoothing parameter v_ϵ . Good numerical performance has been obtained with $v_\epsilon = 0.001$ m/s.

2.4 Large Vertical Operable Aperture

We will now combine the model for the effective air leakage area, presented in Sec. 2.2, and the model for large vertical open apertures, presented in Sec. 2.3, to construct a model for a large aperture that can be open or closed, depending on an input signal. As in the model for the open aperture, we discretize the opening in n horizontal segments, and compute the pressure difference $\Delta P_{AB}(z)$ in each compartment using (10). Let $y \in [0, 1]$ be an input signal, defined such that the aperture is closed if $y = 0$ and open if $y = 1$.

We combine the two models by using a linear combination of the flow constants k and the flow exponents

m of each model. In particular, in view of (6) and (4), we set for the closed aperture

$$k_{clo} = \frac{1}{n} C_d \frac{C_d}{C_{d,r}} L \Delta P_r^{0.5-m} \sqrt{\frac{2}{\bar{\rho}}} \quad (18)$$

where n is the number of compartments and $\bar{\rho} \triangleq (\bar{\rho}_A + \bar{\rho}_B)/2$, and we set for the open aperture

$$k_{ope} = \frac{1}{n} C_d wh \sqrt{\frac{2}{\bar{\rho}}}. \quad (19)$$

In both models, the flow exponent m is a user-specified parameter which we denote by m_{clo} and m_{ope} . The effective flow constant and flow exponent is computed as

$$k = y k_{ope} + (1 - y) k_{clo}, \quad (20)$$

$$m = y m_{ope} + (1 - y) m_{clo}. \quad (21)$$

We do not claim that this model is able to describe accurately the flow through a half opened door by setting $y = 1/2$. However, for numerical reasons, to change the aperture from open to closed, it may be better to vary the input signal y continuously in time as opposed to using a step function.

2.5 Pressure of a Medium Column

This model can be used to model the static pressure of a medium column. The model computes the difference between the static pressure at the bottom P_b and the top P_t of a vertical medium column of height h as

$$P_b = P_t + h \rho g, \quad (22)$$

where the density ρ is a user input and $g = 9.81$ m/s² is the earth acceleration.

3 Implementation in Modelica

We implemented the models using Modelica 2.2 [16] and Modelica_Fluid [5]. All models use one or several instances of a model that implements the power law (1) to define the relationship between flow and pressure difference. Fig. 3 and Fig. 4 show the class hierarchy of the models with one- and two-directional flow, and how the different models relate to the physical description described in the previous sections.

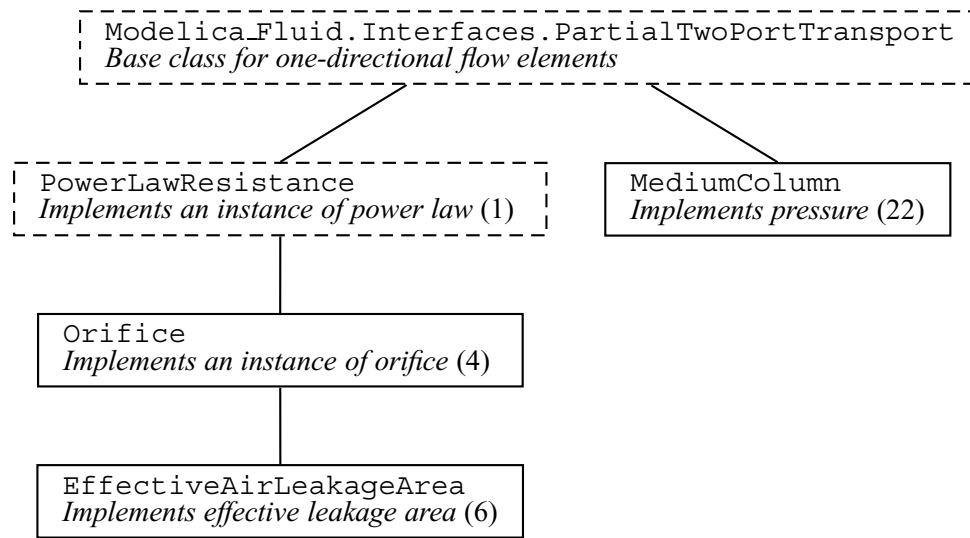


Figure 3: Class hierarchy of models with one-directional flow. The dashed models are partial models.

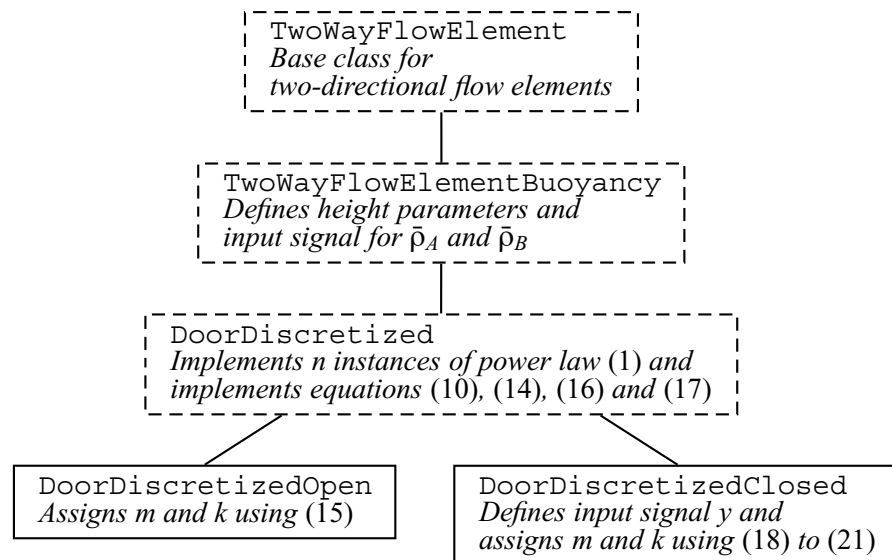


Figure 4: Class hierarchy of models with two-directional flow. The dashed models are partial models.

Table 1: Parameters of the flow elements.

	Orifice	Door
Discharge coefficient C_D	0.65	0.78
Flow coefficient m	0.5	0.78
Transition to linear pressure law $\Delta P_{0,\varepsilon}$ in Pa	0.1	0.1
Area A in m^2	0.01	2.2
Width w in m	-	1
Height h in m	-	2.2
Number of compartments n_{com}	-	10
Minimum velocity v_ε in m/s	-	0.001

Table 2: Mass flow rates in kg/s computed by CONTAM and by Modelica.

Flow element	CONTAM	Modelica
door West to East	0.4103	0.4184
door East to West	0.4146	0.4228
opening WT	0.0044	0.0044
opening ET	0.0044	0.0044
opening OB	0.0080	0.0080
opening OT	0.0080	0.0079

4 Comparative Model Validation

We will now compare the buoyancy driven air flow rates through orifices and open doors that were computed in Modelica using Dymola 5.3b with the ones computed in CONTAMW 2.1. The computations were done on a Windows 2000 computer.

Fig. 5 shows a side view of the three rooms used in the validation, Fig. 6 shows the implementation in Dymola and Tab. 1 shows the parameters used to describe the flow paths. In the Modelica computations, the parameter $\Delta P_{0,\varepsilon}$ used in the smoothing described in (1) was set to 0.1 Pa and we used $\tilde{\rho} = 1.2 \text{ kg/m}^3$ in the orifice equation (2).

Tab. 2 lists the mass flow rates computed by CONTAMW and by Modelica. The door model has the biggest difference in mass flow rate. However, the difference is only 2%, and hence the CONTAM and Modelica model agree within the accuracy expected by the model simplification, such as the assumption of constant density in the flow path and the discretization of the door height.

5 Conclusions

Using an object-oriented equation-based modeling language allowed a rapid implementation of a library with multizone air flow models that can be connected to thermal building models. A comparative model validation of the buoyancy driven steady-state mass flow rates in a two storey building between our models and CONTAM showed agreement of the mass flow rates within 2%. This deviation is significantly smaller than the uncertainty associated with selecting model parameters. Large uncertainty in model parameters exist in the selection of the discharge coefficient, the flow exponent and the equivalent leakage area of cracks in the building envelope.

From a computational point of view, smoothing the equations to convert them into once continuously differentiable equations was critical. Without smoothing, the solver failed in some examples to solve the system of equations.

6 Acknowledgments

This research was supported by the U.S. Department of Commerce, National Institute of Standards and Technology, Advanced Technology Program under the agreement number 70NANB4H3024.

7 Nomenclature

7.1 Conventions

1. Vectors elements are denoted by superscripts.
2. $f(\cdot)$ denotes a function where (\cdot) stands for the undesignated variables. $f(x)$ denotes the value of $f(\cdot)$ for the argument x .

7.2 Symbols

$a \in A$	a is an element of A
\mathbb{N}	$\{1, 2, 3, \dots\}$
\triangleq	equal by definition

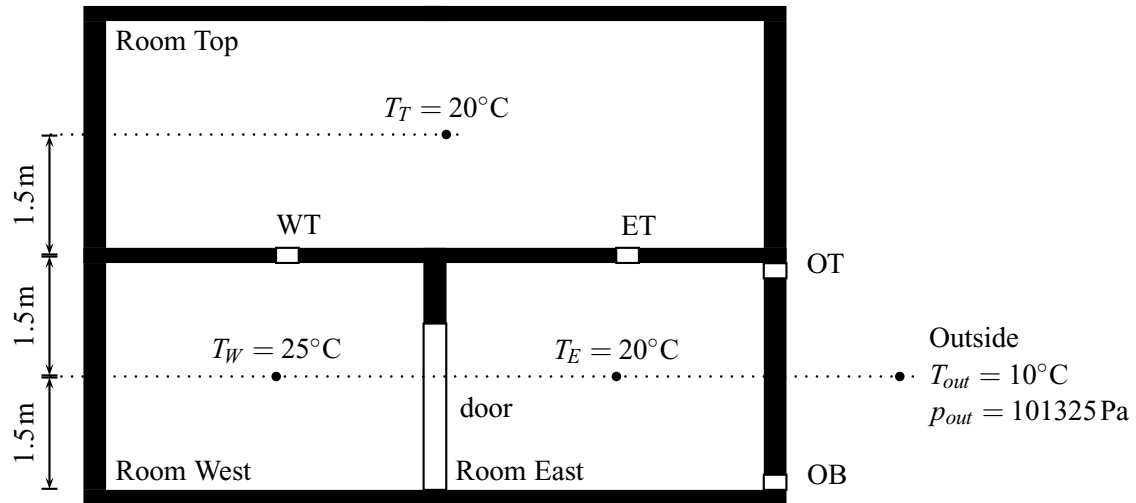


Figure 5: Side view of the three rooms used in the validation study.

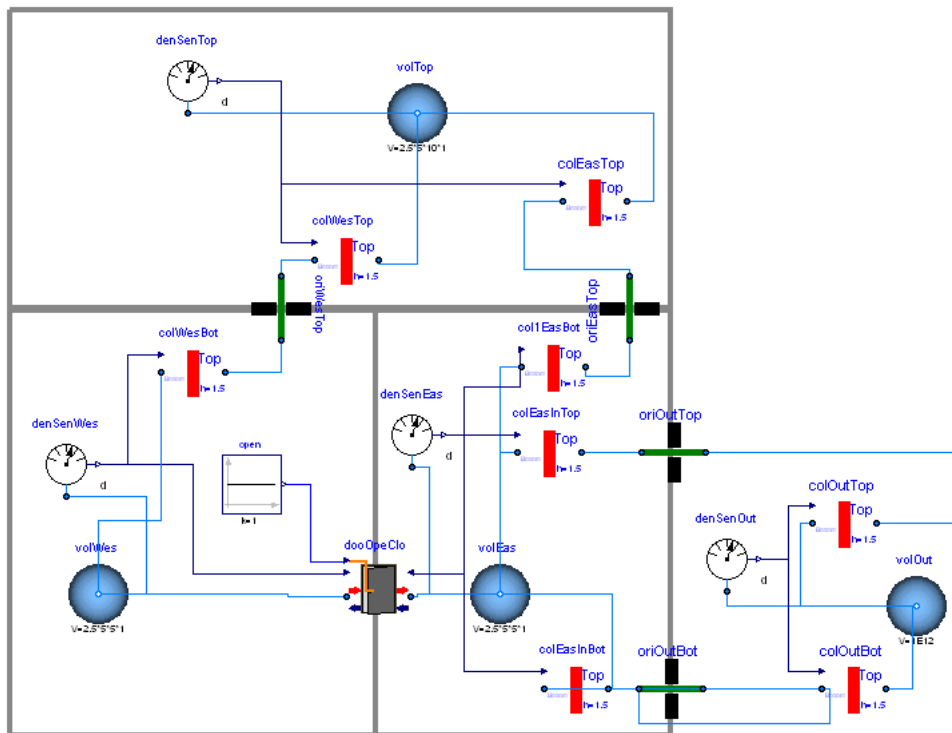


Figure 6: Implementation of the three room problem that was used for the comparative program validation. The green flow paths are orifice models that implement the model of Section 2.1, the red bars are medium column models that implement the model of Section 2.5, and the gray rectangle is the door model of Section 2.3 and 2.4.

References

- [1] ASHRAE. *Fundamentals*. American Society of Heating, Refrigeration and Air-Conditioning Engineers, 1997.
- [2] Axel Bring and Per Sahlin. Modelling air flows and buildings with NMF and IDA. In *Proc. of the 3-rd IBPSA Conference*, pages 463–469, Adelaide, Australia, August 1993.
- [3] Joe A. Clarke. *Energy Simulation in Building Design*. Butterworth-Heinemann, Oxford, UK, 2nd edition, 2001.
- [4] W. Stuart Dols and George N. Walton. CONTAM 2.0 user manual, multizone airflow and contaminant transport analysis software. Technical Report NISTIR 6921, National Institute of Standards and Technology, November 2002.
- [5] Hilding Elmqvist, Hubertus Tummescheit, and Martin Otter. Object-oriented modeling of thermo-fluid systems. In Peter Fritzson, editor, *Proceedings of the 3rd Modelica conference*, pages 269–286, Linköping, Sweden, November 2003. Modelica Association and Institutionen för datavetenskap, Linköpings universitet.
- [6] Helmut E. Feustel. COMIS - an international multizone air-flow and contaminant transport model. Technical Report LBNL-42182, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, August 1998.
- [7] Helmut E. Feustel and Juergen Dieris. A survey of airflow models for multizone structures. *Energy and Buildings*, 18:79–100, 1992.
- [8] J. L. M. Hensen, J. van der Maas, and A. Roos. Air and heat flow through large vertical openings. In *Proc. of the 3-rd IBPSA Conference*, pages 479–485, Adelaide, Australia, August 1993.
- [9] Joe Huang, Frederick C. Winkelmann, Frederick F. Buhl, Curtis O. Pedersen, Daniel E. Fisher, Richard J. Liesen, Russell D. Taylor, Richard K. Strand, Drury B. Crawley, and Linda K. Lawrie. Linking the COMIS multi-zone airflow model with the EnergyPlus building energy simulation program. In *Proc. of the 6-th IBPSA Conference*, volume II, pages 1065–1070, Kyoto, Japan, September 1999.
- [10] Christian Inard, Hassan Bouia, and Pascal Dali-cieux. Prediction of air temperature distribution in buildings with a zonal model. *Energy and Buildings*, 24(2):125–132, July 1996.
- [11] D. E. Kiel and D. J. Wilson. Gravity driven flows through open doors. In *7th AIVC Conference*, Stratford on Avon, UK, September 1986. Air Infiltration and Ventilation Centre.
- [12] David M. Lorenzetti. Assessing multizone air-flow software. Technical Report LBNL-47653, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, December 2001.
- [13] B. M. Mahajan. Measurement of interzonal heat and mass transfer by natural convection. *Solar Energy*, 38:437–446, 1987.
- [14] B. M. Mahajan and D. D. Hill. Interzonal natural convection for various aperture configurations. In *ASME Winter Annual Meeting*, Anaheim, CA, 1986. American Society of Mechanical Engineers.
- [15] Timothy P. McDowell, Steven Emmerich, Jeff W. Thornton, and George N. Walton. Integration of airflow and energy simulation using CONTAM and TRNSYS. *ASHRAE Transaction*, 109(1):1–14, 2003.
- [16] Modelica Association. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 2.2*, February 2005.
- [17] S. B. Riffat. A study of heat and mass transfer through a doorway in a conventional house. Technical Report AIVC 2454, Research in Building Group, Polytechnic Central London, January 1988.
- [18] Per Sahlin. On the effects of decoupling airflow and heat balance in building simulation models. *ASHRAE Transaction*, 109(2):788–800, 2003.
- [19] Per Sahlin and Axel Bring. IDA solver – A tool for building and energy systems simulation. In J. A. Clarke, J. W. Mitchell, and R. C. Van de Perre, editors, *Proc. of the IBPSA Conference*, Nice, France, August 1991.
- [20] J. van der Mass, C. A. Roulet, and J. A. Hertig. Some aspects of gravity driven air flow through large apertures in buildings. *ASHRAE Transactions*, 95(2):573–583, 1989.

- [21] George N. Walton. Airflow network models for element based building airflow modeling. *ASHRAE Transactions*, 95(2):611–620, 1989.
- [22] Andreas Weber, Markus Koschenz, Viktor Dorer, Marion Hiller, and Stefan Holst. TRNFLOW, a new tool for the modeling of heat, air and pollutant transport in buildings within TRNSYS. In R. Lamberts, C. O. R. Negrão, and J. Hensen, editors, *Proc. of the 7-th IBPSA Conference*, volume III, pages 1363–1368, Rio de Janeiro, Brazil, August 2001.
- [23] Michael Wetter. Multizone building model for thermal building simulation in modelica. In *Submitted to: Modelica conference 2006*, Vienna, Austria, September 2006. Modelica Association.

Modelling of a Solar Thermal Reactor for Hydrogen Generation

Jürgen Dersch Andreas Mathijssen Martin Roeb Christian Sattler

Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)

Institute of Technical Thermodynamics, Solar Research

51170 Köln, Germany

juergen.dersch@dlr.de andreas@mathijssen.de martin.roeb@dlr.de christian.sattler@dlr.de

Abstract

The transient thermal behavior of a solar thermal reactor for hydrogen generation has been modeled using Modelica/Dymola. The model is used to predict temperatures and reaction rates inside the reactor which is operated alternating between two temperature levels. The purpose of this reactor is the production of hydrogen as a future carbon free fuel using exclusively regenerative energy resources.

First results are promising and the model may be used to explain experimental observations from the operation as well as for theoretical studies.

Keywords: solar, hydrogen, chemical reaction, heat conduction

1 Introduction

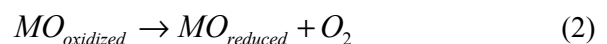
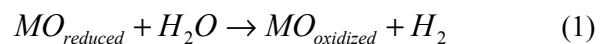
Hydrogen generated from renewable energy sources has the potential to become an important energy carrier for the future, particularly for mobile applications. Beside the well known path of using electrical energy from renewables for electrolysis, the solar heat may be used directly for water splitting without the additional step of electricity generation. Whereas direct thermal water splitting needs temperatures above 2500 K, several multi-step thermo-chemical processes are known, enabling the hydrogen generation at temperatures which are controllable by today's technical equipment. One of these thermo-chemical processes is currently under investigation in the HYDROSOL-2 project funded by the EC [1]. A two-step water splitting process has been developed using mixed iron-oxides on a porous ceramic structure in a reactor heated directly by solar radiation. The reactor is operated with alternating reaction conditions, in particular with two alternating temperature levels of about 1073 K and 1350 K respectively. Experimental results from a small scale lab reactor are available which prove the general feasi-

bility of the process and the "cycleability" of the redox materials. A scale-up of the technology aiming at the demonstration in a 100kW-scale is in progress.

From this short description and the fact that solar radiation varies with daytime and cloud coverage, it is evident that the operation of this reactor is highly dynamical. Therefore a mathematical tool is necessary in order to model the reactor for theoretical studies and scale-up.

2 Description of the process and the experimental setup

The experimental and theoretical work is focused on a thermo-chemical cycle using materials that can act as effective water splitters at moderate temperatures in a two-step reaction scheme. The overall reaction may be described by the following steps which are conducted successively at different temperature levels:



The metal oxides (MO) used in this particular reaction are ion oxides doped with other divalent metals (Zn, Ni, Mn). The basic idea is to employ an innovative monolithic solar reactor for the production of hydrogen from the splitting of steam using solar energy, by combining a refractory ceramic thin-wall, multi-channeled (honeycomb) support structure optimized to absorb solar radiation and develop sufficiently high temperatures, with a redox pair system suitable for the performance of water dissociation and at the same time suitable for regeneration at these temperatures. With this concept complete operation of the whole process (water splitting and redox material regeneration) can be achieved by a single solar energy converter.

Figure 1 shows a cross section of the experimental reactor used for the first experiments in the solar fur-

nance at DLR, Cologne. The outer diameter of the housing is 0.43 m and the diameter of the porous silicon carbide structure carrying the ion oxides is 0.144 m. The front side is covered by a quartz glass window for the concentrated sunlight admission. The concentrated sunlight is provided by a mirror system capable for concentrations up to 5000 suns.

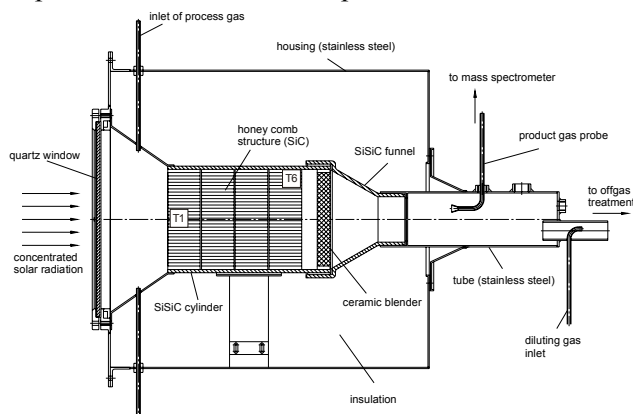


Figure 1: Cross section of the laboratory scale reactor for solar thermal hydrogen production

Typical input power for this experimental setup was between 2000 and 3000W. The input gases (N_2 and H_2O) are fed via four small tubes radially distributed over the reactor circumference. The entrance for these gases is on the right hand side of Figure 1 and the flow direction is from right to left. Behind the SiC structure, a ceramic blender is used, to achieve an optimal mixing and then the gases are leaving the reactor to the product gas analysis and off-gas treatment.



Figure 2: Photo of the laboratory scale reactor for solar thermal hydrogen production

The typical operation of this reactor may be described by the following steps:

1. Preheating of the cold reactor with concentrated solar radiation up to 1073 K.

2. Steam admission and production of hydrogen at 1073 K.
3. Increasing the solar thermal input in order to reach the higher temperature level of 1350 K.
4. Regeneration period at this temperature level. During this period the reactor is flushed with nitrogen.
5. Reduction of the solar thermal input to come down to the low temperature level of 1073 K

Steps 2-5 are repeated consecutively until the end of the experiment. Experimental setup with two reactors of this type, each one in a different step, has also been tested in order to get a continuous hydrogen production.

3 Model approach

An overall model has been set up for this reactor in Dymola using partial models from the standard library Modelica 2.2 in combination with the Modelica_Fluid library and additional adapted or newly developed models. The main purpose of this first model approach is the simulation of the thermal behavior of the reactor. Reaction kinetics are considered by simple preliminary approaches since quantitative experimental data on reaction velocities are not yet available.

Figure 3 shows a screenshot of the overall model. The model is 2-dimensional due to the rotational symmetry of the reactor. From this figure, it becomes obvious, that most of the icons used here are from the Modelica 2.2 library. This is also valid for most of the associated models, but some of them have been modified in order to meet the special requirements. The only model used without any modifications is the Modelica.Blocks.Sources.CombiTimeTable for input data of ambient temperature and solar power input. These values are available from measurements in the solar furnace and they are used as input data for the simulations.

The input model MassFlowSource is a descendant of CombiTimeTable too, but with some additional code to allow for input flows with different units (like kg/s or l/h) and multiple output flows in SI units. PrescribedTemperature, Convection, FeedPump, IdealMixer and Sink are made from standard Modelica models with the extensions of distributed connectors and multi-component mass flows. The reactor, the insulation and the solar heat input distribution are new models and further details are given below.

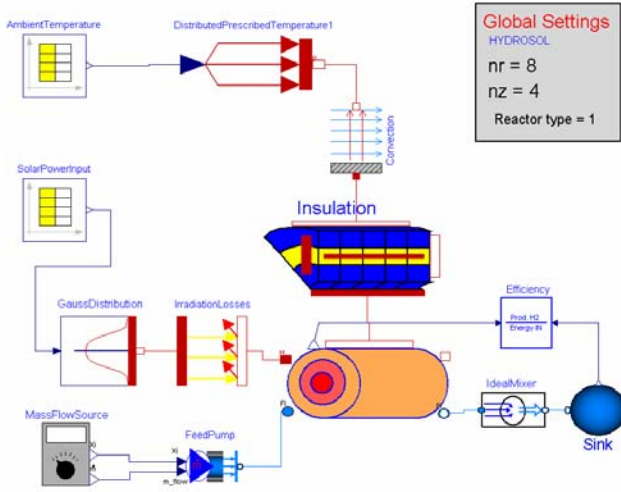


Figure 3: Screenshot of the overall model used for the simulation of the laboratory scale reactor for solar hydrogen generation

The spatial distribution of solar radiation input is not constant throughout the aperture, it shows a Gaussian-like shape with a peak in the centre and considerably lower values towards the rim. Measurements of the flux distribution at the reactor entrance are available as well as the integral value of the solar power input during the experiments. The object called “GaussDistribution” in Figure 3 is used to distribute the total input power among the individual rings of the reactor front surface. The shape of the power distribution is approximated by the equation:

$$P(r) = P_{total} \left[y_0 + \frac{a_0}{w\sqrt{0.5\pi}} \exp\left(-2\left(\frac{r}{w}\right)^2\right) \right] \quad (3)$$

Measured values of the input flux distribution at nominal power are shown in Figure 4 in comparison to this approximation. Although the measured distribution is not exactly symmetrical, this assumption was made for the model in order to use a 2-dimensional model and restrict the number of equations.

The object IrradiationLosses in Figure 3 is used to model the thermal losses due to absorption and re-radiation through the quartz glass window at the front side of the reactor. The result is a net power input to the reactor depending on the actual surface temperature.

The ceramic monolith inside the reactor as well as the insulation and the housing are modeled as concentric circular rings build up from several slices in axial direction (Figure 5). Mass and energy balances

are set up using the finite volume method as described in [2].

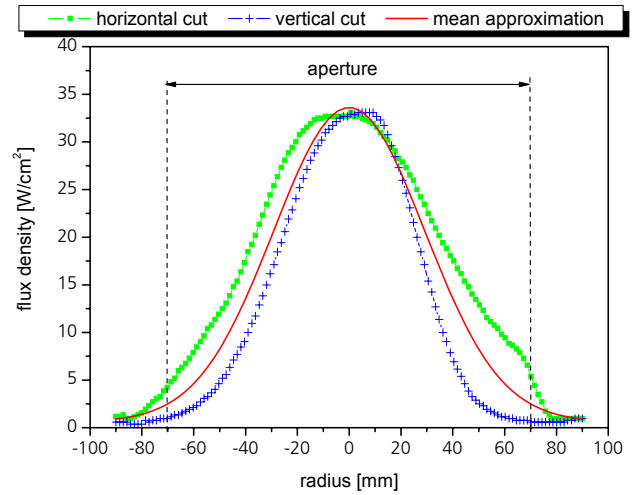


Figure 4: Measured and approximated flux density at the reactor entrance.

Pressure gradients are neglected in this model. They are low enough to neglect an influence on the chemical reactions but the authors are well aware that there may occur instabilities influencing the local gas flow and thereby the temperature distribution within the ceramic monolith [3]. Therefore these local pressure drops may be incorporated into a more advanced future model.

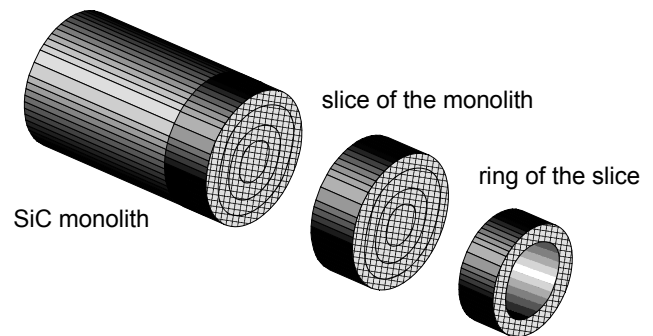


Figure 5: Balance volumes used for the model of the solid SiC monolith

The cylindrical ceramic monolith as the main component of the solar hydrogen reactor is modeled by three major objects:

1. the “solid”-object representing the SiC matrix with the task of absorbing the radiation at the front surface and providing an even temperature distribution by heat conduction,
2. the “fluid”-object representing the gas channels with square cross sections to allow a gas flow through the reactor,

3. and the “coating”-object for the reactive layer, which is located at the inner surface of the SiC matrix.

The latter object may be considered as intermediate layer between the solid and the fluid object. Its mass balance accounts for accumulation or release of oxygen depending on the operating conditions. The fluid object does not consider any mass accumulation and concentrations of the gaseous species are assumed to be constant within each control volume. Transport resistance between fluid and coating is neglected as well. Since the coating itself has a very small mass and heat capacity, the thermal inertia is represented solely by the SiC matrix.

The solid object is based on the differential equation for 2-dimensional heat transfer in cylinder coordinates:

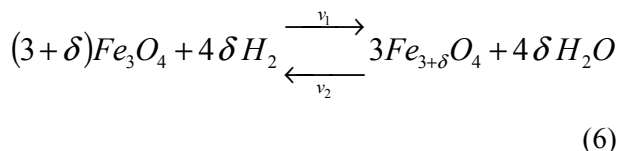
$$\rho c \frac{\partial T}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left(r \lambda \frac{\partial T}{\partial r} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + \phi \quad (4)$$

A central difference scheme was applied to approximate the spatial partial derivatives. The effective thermal conductivity of the SiC honeycomb structure is not the same for the radial and the longitudinal direction since the solid walls show different thickness. The effective values are calculated according to cross section fraction covered by the solid material. For the radial direction the equation reads:

$$\lambda_r = \frac{A_{r,solid}}{A_{r,total}} \lambda \quad (5)$$

The fluid object contains an instance of Modelica.Media.IdealGases.Common.MixtureGasNasa representing a mixture of N₂, H₂O, H₂ and O₂. This Modelica package provides all fluid properties needed for the simulation.

At present, the knowledge about reaction kinetics is only marginal and therefore the reaction rate is calculated by a preliminary equation based on investigations of Tsuji et al. [4]. The experimental investigation of reaction rates of Tsuji et al. have been carried out in the temperature range between 250 and 350°C, which is much lower than the operating temperature range of the solar reactor.



According to [4] the overall reaction rate for the water splitting is:

$$v_s = v_2 - v_1 = k_2 \delta^{0.95} p_{H_2O}^{0.45} - k_1 p_{H_2}^{1.1} \quad (7)$$

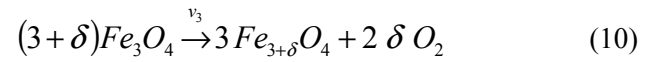
with

$$k_1 = 5.345e-12 \exp\left(-\frac{63.8}{RT}\right) \quad (8)$$

and

$$k_2 = 0.03634 \exp\left(-\frac{25.5}{RT}\right) \quad (9)$$

For the reverse regeneration reaction



The velocity is calculated from

$$v_3 = k_3 \frac{m_{O_2,stored}}{V_{react}} \quad (11)$$

$$k_3 = 2.1e8 \exp\left(-\frac{225}{RT}\right) \quad (12)$$

Activation energy and frequency factor in Eq. (10) are estimated from the experimental observations that a considerable reaction rate does not take place beyond 1200 K and the time period needed for a complete regeneration is about 20 minutes. It should be emphasized, that these reaction rates are only first estimates and further investigations are necessary in order to get more reliable data about these reactions. The solar reactor is not suitable for this kind of experiments because the number of uncertain parameters is quite high. They should be carried out under well defined and almost constant conditions, e.g. in a differential reactor.

The temperature distribution inside the reactor, the total product mass flow rate, and the efficiency of the reactor are important results from this model.

4 Simulation results

The first experiments with the solar reactor were performed without any hydrogen production in order to investigate the thermal behavior of the plant. Figure 6 shows simulation results in comparison to measured data from one of these first experiments. Inside the ceramic monolith only one thermocouple is mounted for temperature measurement. It is located at the center near the front side where the feed gases

and the solar radiation are entering the monolith (see Figure 1).

The matching of simulated and measured temperatures is fairly good, a conclusion which is also valid for other experiments. The remaining differences are due to model simplifications and also due to uncertainties in the available physical properties, e.g. the thermal conductivity of SiC at elevated temperatures, which is not exactly known since it depends on the material and also on the processing of the monolith.

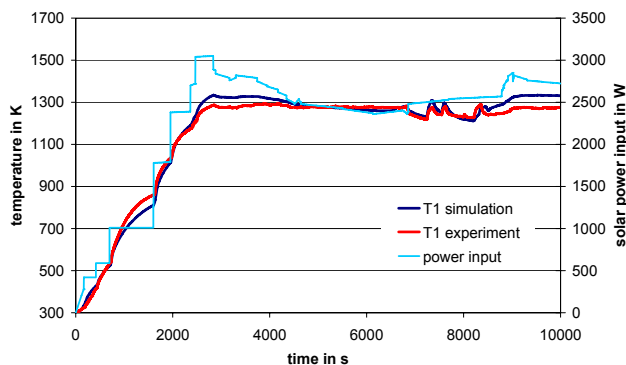


Figure 6: Comparison of measured and calculated temperature in the center of the reactor for a start-up.

The model was also used to investigate the sensitivity with regard to the reactor length. Figure 7 shows the results of this study. Here two different lengths of the ceramic reactor element have been simulated: 5 cm (the actual length used in the laboratory reactor) and 10 cm. Beside the length and the number of axial balance volumes all input parameters are identical. In Figure 7 the temperature along the center axis is drawn for steady state conditions. The temperature curve for the 10 cm monolith is not affected by the number of discrete elements (4 or 8). The shorter monolith shows a lower temperature towards the gas exit (right hand side). This result was unexpected because the heat input is at $l=0$ and with increasing distance the temperature should decrease from a first feeling. Heat transfer and temperature compensation between solid and gas are the main reason for the temperature decrease along the axis. The gas feed is cold compared to the temperature of the SiC and the temperature balancing needs about 4 cm independently of the reactor length.

The remaining temperature decrease of gas and solid is due to losses caused by radial heat conduction. Due to the extended heat transfer area between gas and solid for the longer monolith, the radial losses are smaller than for the shorter type. This behavior is supported by the different effective solid heat conductivity in axial and in radial direction. Due to the macroscopic structure the conductivity in axial direc-

tion is 1.72 times of the conductivity in radial direction.

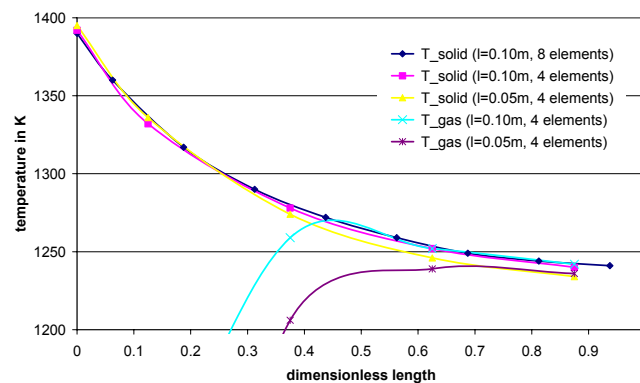


Figure 7: Study on the dependency of reactor length and number of axial discrete volume elements (steady state conditions)

Since the temperature difference at the exit for both lengths is not very large (about 8 K) its impact on the reaction rate may be neglected compared to other model uncertainties. From Figure 7 it is also obvious, that the gas and solid temperatures are equalized at the exit of the monolith for both lengths. Nevertheless, a doubling of the monolith length means also a doubling of the reactive surface, which could be an advantage for the chemical reaction because the available reactive coating is twice as for the short monolith.

The solar reactor shown in Figure 1 has been redesigned and currently a dual reactor concept is used for continuous hydrogen production. This type has two reaction chambers; one can be operated in production mode while the other one is in regeneration mode (Figure 8). After a certain time period the power input and the gas feed are shifted and the operation mode is inverted. The shape and the dimensions of the SiC monolith are the same as for the original reactor. Experimental data with hydrogen production are available from this setup.

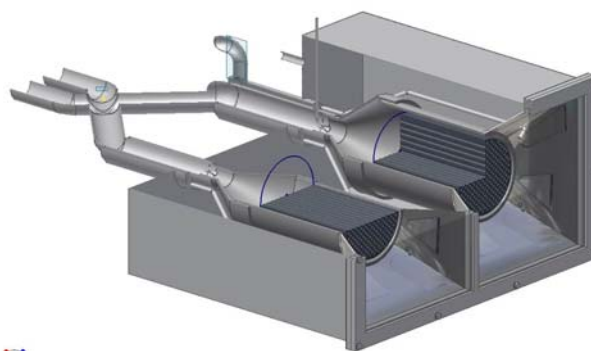


Figure 8: Drawing of the dual reactor for solar hydrogen production

The main difference between both reactors from the model point of view is the rectangular shape of the insulation and the outer shell. This would break the rotational symmetry and account for a 3-dimensional model. Nevertheless the 2-dimensional approach has been used for the simulation in order to limit the calculation time. Since the rectangular geometry is only used for the outer shell, this approximation seems to be still acceptable. Furthermore the current knowledge about the chemical reaction allows only a qualitative comparison of simulation and experiment concerning the hydrogen production rate. Consequently the main purpose of the model is not the prediction of hydrogen production rate but the dynamical thermal analysis of the reactor since the spatial and temporal temperature distribution inside the ceramic monolith is of great importance for the reactor performance.

For this qualitative comparison the input data was simplified and smoothened because particularly the solar power shows many fluctuations and the experimental dataset has a temporal resolution of 5 seconds. According to the Modelica concept, this would lead to many events and thus to unacceptable simulation times. Figure 9 shows the simplified power input and two calculated temperatures. T1 is the temperature in the center of the monolith near the front side and T6 at the outer rim of the monolith near the back side.

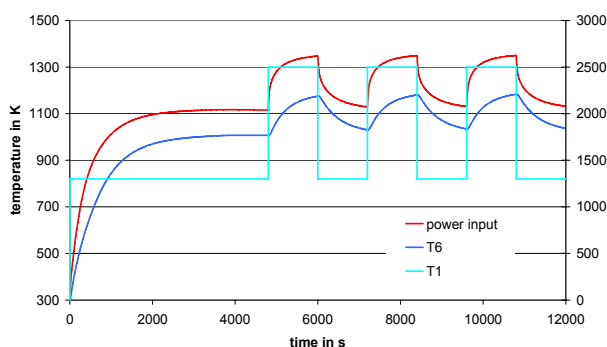


Figure 9: Solar power input and calculated temperatures of a reactor startup and 3 cycles of hydrogen production and regeneration. (see Figure 1 for the position of T1 and T6)

The first hour was used to heat up the whole installation by solar radiation and the steam admission starts at $t=3600$ s. At the same time the hydrogen production starts with a peak mass flow rate, which is decreasing during the next 1200s of the experiment (Figure 10). At $t=4800$ s the steam input is stopped, the solar power input is increased immediately to heat up the reactor and the regeneration period starts. At $t=6000$ s the hydrogen production starts again, but

with a lower initial rate than the first time. This periodic operation is repeated two more times. From Figure 9 it is obvious, that the temperature difference between T1 and T6 is about 100 – 170K, which concurrently represents the largest temperature difference inside the monolith.

An acceptable matching between experiment and simulation was reached by adjusting the reaction rate parameters k_1 and k_2 , although the reaction rates are highly temperature dependent and more reliable kinetic data is urgently desired.

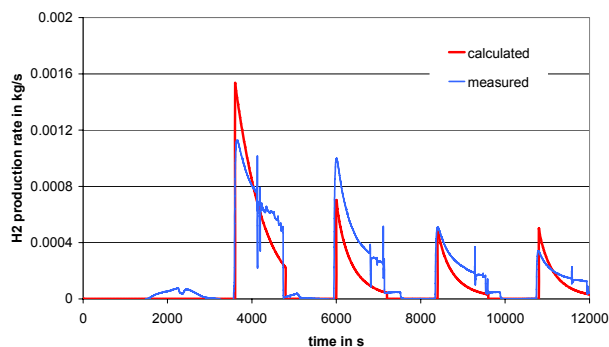


Figure 10: Comparison of calculated and measured hydrogen production rates

The temperature difference between T1 and T6 in Figure 9 may be used to explain the decreasing initial hydrogen production rate, which is presumably caused by large temperature gradients inside the ceramic structure. These temperature gradients are preventing a complete regeneration of the reactive layer since a certain temperature level is necessary for this step.

Figure 11 shows the absolute mass of available ferrite, which is the figure determining the potential of hydrogen production since the oxygen is captured by these ferrite molecules. Three curves are shown, the total amount of ferrite, the amount in the center and at the outer rim of the monolith.

The amount of available ferrite for the new monolith starts at a high initial level, decreases during the first hydrogen production period (3600 – 4800s) and increases again during the regeneration period (4800 – 6000s). The regeneration is incomplete and the next cycle starts at a lower initial peak ferrite concentration compared to the first cycle. The simulation results are demonstrating that the regeneration reaches an almost constant peak level at the second regeneration cycle (8400s) because the same value (0.06 mol) is achieved for the following cycle. From Figure 11 it is also obvious, that the incomplete regeneration is caused mainly by the outer areas of the monolith, which are not hot enough for regeneration. This lower temperature is caused by the lower solar power

input towards the rim and the radial heat losses at this area.

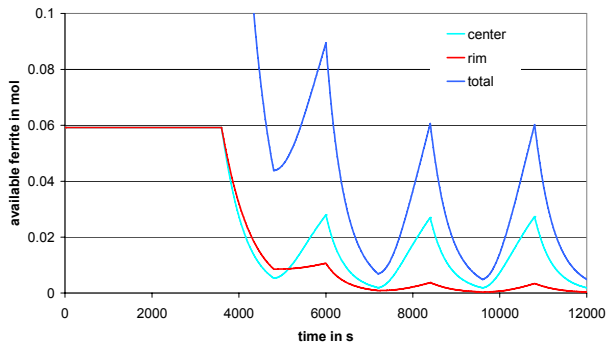


Figure 11: Calculated values of available ferrite for the center part, the outer part and the whole monolith during the periodic operation

5 Conclusions

The Modelica language has been used to set up a model for a solar heated hydrogen generation reactor and to simulate the dynamic behavior of the reactor. First results are promising and the model may be used to explain the decreasing reaction rates, which have been observed during the experiments. The main purpose of the model is the prediction of the thermal dynamics of this reactor rather than the prediction of the hydrogen production rate. Nevertheless heat production and heat consumption are combined with the reactions and therefore the proper calculation of the reaction rates is important for the heat balances.

Model developments as well as experimental investigations are continued, in order to get a deeper insight into this interesting path of solar energy utilization. Next steps are model refinements as well as the set up of a structured library and simulations of an up-scaled plant.

This project was the first experience of the authors using Modelica/Dymola though they have been using other modeling languages (Smile [5]) in the past. The initial orientation period in Dymola/Modelica was rather short and successful working was possible after an acceptable time. Due to the complex matter, general knowledge about dynamical models and the continuous working with the software is an advantage for the person performing the work.

Acknowledgement

The authors would like to thank the European Commission for partial funding of this work within the Project HYDROSOL-2 “Solar Hydrogen via Water Splitting in Advanced Monolithic Reactors for Future Solar Power Plants” (SES6-CT-2005-020030), under the Sixth Framework Programme of the European Community (2002-2006).

Symbols

A	area	m ²
a ₀	coefficient of the solar power input distribution	-
c	specific heat capacity	J/(kg K)
k ₁	reaction rate constant for water recombination	mol/(dm ³ s Pa ^{1.1})
k ₂	reaction rate constant for water splitting	mol/(dm ³ s Pa ^{0.45})
k ₃	reaction rate constant for regeneration	mol/(kg s)
P	power input	W
p	partial pressure	Pa
R	gas constant	kJ/(mol K)
r	radius, coordinate in radial direction	m
T	temperature	K
t	time	s
v	reaction velocity	mol/(dm ³ s)
w	standard deviation of the power distribution	-
y ₀	coefficient of the solar power input distribution	-
z	coordinate in axial direction	m
δ	amount of excess cations	-
φ	internal heat source	W/m ³
λ	thermal conductivity	W/(m K)
ρ	density	kg/m ³

References

- [1] Roeb, M., Monnerie, N., Schmitz, M., Sattler, C., Konstandopoulos, A., Agrafiotis, C., Zaspalis, V.T., Nalbandian, L., Steele, A., Stobbe, P.: Thermo-chemical production of Hydrogen from Water by Metal Oxides Fixed on Ceramic Substrates, Proceedings of the 16th World Hydrogen Energy Conference, Lyon, France, June 13-16, 2006.
- [2] Elmqvist H., Tummescheit H., Otter M.: Object-Oriented Modeling of Thermo-Fluid Systems, Modelica 2003, Sweden, November 3-4, 2003.
- [3] Buck R., Massenstrom-Instabilitäten bei volumetrischen Receiver-Reaktoren, VDI-Verlag, Düsseldorf, 2000.
- [4] Tsuji M., Togawa T., Wada Y., Sano T., Tamaura, Y., Kinetic Study of the Formation of Cation-excess Magnetite, J. Chem. Soc. Faraday Trans. 1995 , 91(10), 1533-1538.
- [5] Jochum P. , Kloas M. The Dynamic Simulation Environment Smile. In Tsatsaronis Ed., Second Biennial European Conference on System Design & Analysis (1994), pp. 53-56. ASME

Object Oriented Modelling of DISS Solar Thermal Power Plant

L.J. Yebra^{†,*} M. Berenguel^{*} E. Zarza[†] S. Dormido[‡]

[†]PSA-CIEMAT. Ctra. de Senés s/n. Tabernas.
E04200 Almería. Spain. E-mail: {luis.yebra, eduardo.zarza}@psa.es

^{*}Universidad de Almería. Dpto. de Lenguajes y Computación. Ctra. de Sacramento s/n.
La Cañada. E04120 Almería. Spain. E-mail: beren@ual.es

[‡]U.N.E.D. Escuela Técnica Superior de Ingeniería Informática. Dpto. Informática y Automática.
C/ Juan del Rosal, 16. E28040 Madrid. Spain. E-mail: sdormido@dia.uned.es

Abstract

The design of advanced control systems to optimize the overall performance of Parabolic Trough Collectors solar plants with Direct Steam Generation is currently a priority research line at PSA-CIEMAT. The development of dynamic models for use in simulation and control of this type of solar power plant is presented in this article, focused on the DISS experimental solar plant. The developed model is based in the thermohydraulic modelling framework ThermoFluid, within the Modelica modelling language. The DISS facility is presented and main modelled components are presented as well as the respective modelling assumptions. A complete model of the facility is presented, developed with the exposed modelling assumptions and component models interconnected. A simulation of this complete model is performed with the boundary conditions defined by a real experiment, and the predicted model output variables are compared with their respective experimental measurements.

Keywords: *object oriented modeling; solar energy; simulation; evaporator; thermofluid; water-steam; two-phase*

1 Introduction

This paper presents the current status of the research performed within the framework of modelling and simulation of Parabolic Trough Collectors (PTC) in the scope of Solar Power Plants. The work is mainly oriented to the development of dynamic models of so-

lar energy plants to be used in the design of automatic control systems aimed at optimizing overall performance.



Figure 1: DISS facility at Plataforma Solar de Almería, in Almería (Spain).

The system used as test-bed plant is the DISS facility, see figure 1. Actually, it is a row arrangement formed by eleven PTCs with a combined length of 500 m, working as a 1 MW_t solar power plant belonging to CIEMAT (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas - *Research Centre for Energy, Environment and Technology*), Research Centre owned by the Spanish Ministry of Science and Education. This solar plant is located at the Plataforma Solar de Almería (PSA), Southeastern Spain. A joint project between CIEMAT-PSA, the University of Almería (UAL), the National University of Distance Education (UNED) and the University of Seville (US) is being carried out in order to develop models and control systems to automatically control

these type of plant. The model presented in this paper will be used in the design of hybrid model predictive control and intelligent control schemes to optimize plant performance, even under start-up and shutdown situations and in spite of highly variable load disturbances due to the daily cycle of solar radiation and passing clouds.

2 The Parabolic Trough Collectors Facility DISS

In this section, an overview of the basic components and operating procedures for the DISS plant is presented. A schematic diagram is shown in figure 2, in which the most important components are depicted.

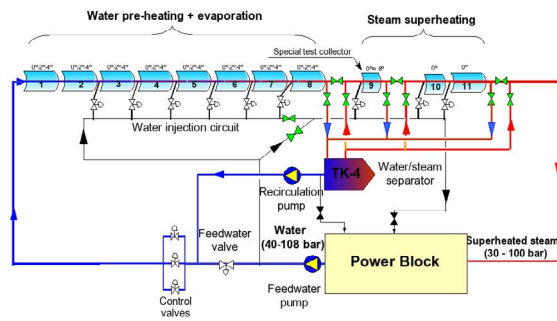


Figure 2: Schematic diagram of DISS facility at PSA

There are three main different operational configurations of DISS facility. In all of them, the operation is based on the concentration of incoming direct solar radiation onto the absorber tube located in the geometrical focal line of a cylindric-parabolic mirror. As the sun position changes during the day, each PTC of the facility has to change its orientation as the solar radiation vector does. The absorber tube in each PTC acts as an energy exchanger, receiving solar energy and transferring it to a thermo-hydraulic circuit with a heat transfer fluid (HTF) as the medium. Traditionally in PTCs the HTF used has been thermal oil, which presents major drawbacks with respect to the water-steam medium used in the DISS facility, as explained in [16]. In addition of the PTCs, the facility is composed by the following components:

- Water-steam separator. Only used in *recirculation* operational mode.
- Pumps: feedwater and recirculation. The former pumps subcooled water into the row and the latter drains saturated liquid water from water-steam generator.

- Injectors. Are actuators to control temperature by injection of subcooled water from the injection line.
- Valves. Let the system be configured in any of the three main operational modes and for control purposes.
- Power Block. It is a component representing any possible load process consuming the regulated outlet of thermal power from the plant. In this case, for water conservation during the experiments, the current implementation returns the thermodynamical state of the outlet superheated steam to subcooled liquid to be pumped by feed-water pump.

Figure 3 shows the three main operational modes, with their pros and cons:

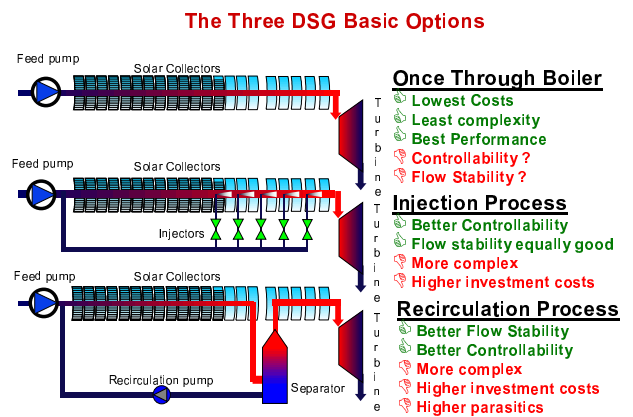


Figure 3: DISS plant operational modes

- *Once-Through*. Lowest investment costs and engineering complexity, although great controllability problems and instabilities during operation. The system works as a distributed once-through boiler fed with subcooled water and superheated steam at outlet, at nominal operating conditions.
- *Injection*. Highest investment costs, although better controllability and less instabilities. The system works as a once-through evaporator in which, at certain points in space, an injectors arrangement help to control spatial temperature distribution along the whole row.
- *Recirculation*. Second highest investment costs, with a good controllability. In this mode, the water-steam separator placed in the row decouples two effective once-through evaporators: the

first one is a flooded evaporator and the second is a dry-expansion one.

In all cases, the flow temperature at the outlet superheated steam must be controlled. Control of the facility has been one of the main efforts that CIEMAT has developed, see ([12]), although the lack of robust dynamic models for control is a major obstacle in the control system development.

CIEMAT is currently interested in these two configurations:

1. *Recirculation*. It offers the trade-off between investment costs and plant controllability. This configuration will be preferred for experimental applications.
2. *Once-through*. Spurred by the lowest investment costs, theoretical work must solve controllability and instabilities through the development of integral control systems for this configuration.

The modelling work presented in this paper will focus on *once-through* configuration for two reasons. First, the process is not observable by means of the actual sensors in the facility; it is not possible to measure/estimate specific enthalpy or mass fractions in transient experiments in the two-phase sections, which in some operational points achieve 50% of the whole length. Second, it is supposed the greatest disagreement between the model and experimental data should appear in this configuration. So any other configuration modelled with components validated in *once-through*, should show a closer agreement with experimental measurements.

In *once-through* operation mode the DISS acts as a *once-through* evaporator of 500 m length with subcooled water at the inlet and superheated steam at the outlet, in nominal operating conditions. A cascade local control loop for the feedwater pump and an outlet controlled valve define boundary conditions for the system. The final objective of the model is to predict the transient behavior of the thermodynamic variables associated with the thermo-hydraulic output power of the evaporator (temperature, pressure, specific enthalpy, etc.), when the external disturbances (mainly concentrated solar radiation, ambient temperature, inlet subcooled water temperature and inlet subcooled injector water temperature) and controllable inputs (inlet subcooled mass flow rate, inlet final injector mass flow rate and outlet superheated steam pressure) change.

3 Object Oriented Modelling of DISS

In this paper we will concentrate in the modelling of the thermo-hydraulic part of the system, skipping the rest the remaining subsystems (pneumatic, mechatronic, etc.) needed to maintain the proper instantaneous orientation of the PTC group, and assuming a known input radiation power in the absorber pipe, as a consequence of the radiation reflected in the cylindrical-parabolic mirrors. For a detailed explanation of this subsystems read [16] and [15].

Due to the fact that the main phenomena of interest is the thermofluid dynamics, the object oriented Modelica language ([1]) has been used to develop these models with the Dymola tool ([5]). Within this modelling language the ThermoFluid library ([11],[6]) is a framework over which develop own libraries and final component models ready to be instantiated as components for simulations. The authors believe this library is an important reference in the framework of object oriented modelling of thermofluid systems with Modelica and its existence makes it unnecessary, in most cases, to develop thermo-hydraulic models from scratch. Instead, the models can be designed by inheritance and aggregation from base classes in the ThermoFluid framework.

The work analyzes each of the components of the thermo-hydraulic water-steam circuit and explains the modelling assumptions, trying to justify each one as they are oriented to get - by means of the symbolic manipulations that Dymola tool performs - a not high index DAE system for the complete model, in which the number of nonlinear algebraic loops is minimized. For this purpose, all the components are classified, following the modelling methodology derived from the Finite Volume Method (FVM) [9], in Control Volumes (CV in ThermoFluid nomenclature) and Flow Models (FM in ThermoFluid nomenclature).

In some cases information about the future control system architecture to be implemented is introduced in the modelling phase. This methodology, from a strict point view, breaks the sequencing work of first model and then design the control system based in this model. But it helps to simplify the design of the models and enhances the numerical behavior of the whole modelled system in the simulation execution phase, without a significant loss of accuracy.

Due to the existence of components whose internal implementation may vary depending on the modelling hypotheses, the polymorphism and the Modelica language constructs for classes and components parameterization has been extensively used and specifically

applied in PTCs models.

Figure 4 shows the developed Modelica model of the DISS facility working in *once-through*.

3.1 ThermoFluid usage

The dynamic behavior expected to be predicted by the models is mainly the thermal one, so the steady state formulation for the momentum balance is selected for the utilized ThermoFluid base classes. The selected time scales for the thermal dynamics are for control design and simulation purposes.

The thermo-hydraulic interface for all the models is formed by connectors from the *Interface* package, for single component media and steady-state momentum balance statement.

The modelling methodology adopted from the beginning for the design of the classes was: *if there exists any class in ThermoFluid that implements the physical phenomenon to model, use it with the corresponding parameterization. If not, design the classes using inheritance from the high level partial classes from ThermoFluid; in other cases then use proper ThermoFluid interfaces and base classes and develop the class with the lacking behavior expressed in differential and algebraic equations from first principles.*

3.2 Designed Classes

In the next subsections the most important components models will be detailed and the modelling hypotheses will be explained and justified.

3.2.1 Pumps and Injectors

In this kind of active FM [11], the authors decided to make a simplifying assumption based on the gained experience in control of Parabolic Trough Fields with thermal oil as medium, case of Acurex field of CIEMAT-PSA [3], [4], and water-steam as medium in DISS facility [16], [12]. This assumption consists that the water pumps are controlled in a cascade scheme [2] with a local control loop whose dynamics is much faster than the rest of the thermo-hydraulic system. This assumption has been experimentally validated in blowers and water pumps, and helps simplify these components models until the possibility of modelling them as steady-state quasi-ideal mass flow rate generators. This approximation avoids the time-consuming work of fitting the nonlinear multivariate curves of the pumps and injectors. So, the algebraic equation for these components is $\dot{m} = \dot{m}_{ref}$,

where \dot{m}_{ref} is the setpoint of the local pump/injector control loop and it is assigned in a connector to the model, as can be seen in figure 4. From an energy point of view are supposed to behave as isenthalpic model, i.e., $h_{inlet} = h_{outlet}$.

3.2.2 Parabolic Trough Collectors

The PTC is the most important component in the facility. Its aim is to carry most of the part of the solar direct incident irradiance in the mirror to the absorber tube. To achieve this aim, the manufacturing process uses advanced material sciences and technologies to minimize the power loss in direction to absorber tube. In [7] and [16], an analysis of the energy flows into the absorber from the sun is developed. The figure 5 shows a front view of a DISS PTC.



Figure 5: Front view of one collector from DISS facility.

The figure 6 shows the schematic diagram of the PTC's Modelica model. The main components of a PTC are shown:

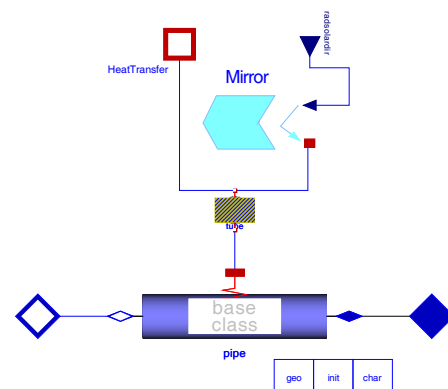


Figure 6: PTC Modelica model.

- Cylindric-Parabolic mirror surface. It reflects the

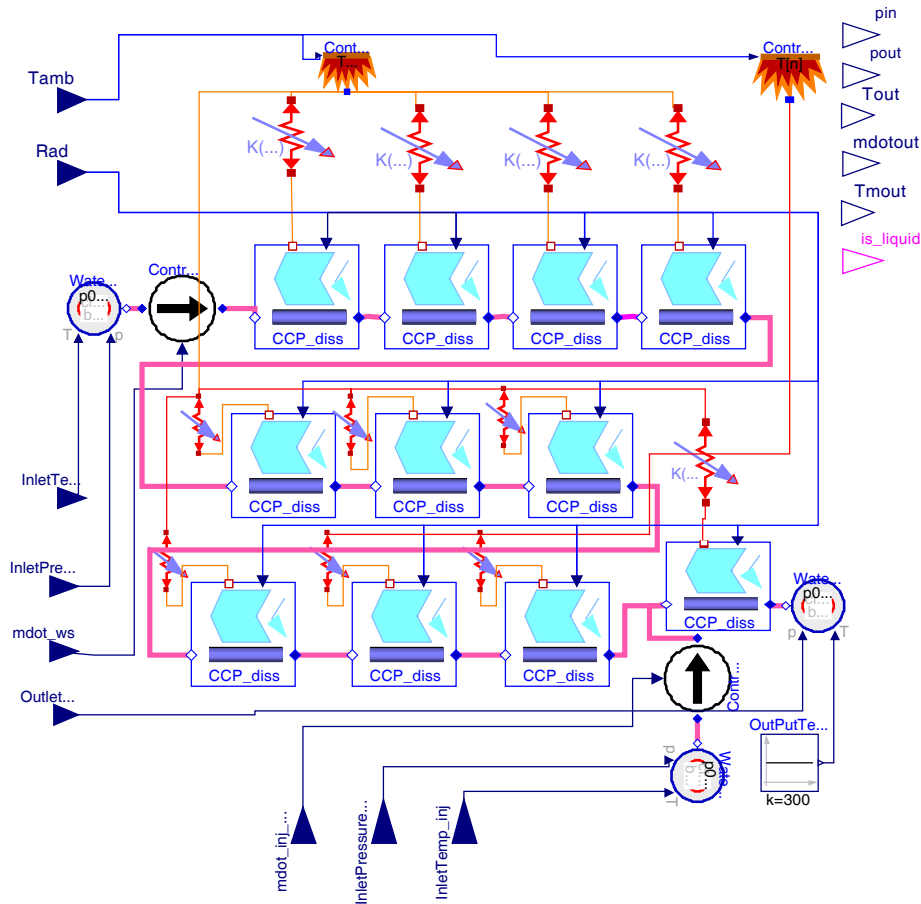


Figure 4: Modelica model of DISS plant in *once-trough* configuration.

incoming direct solar radiation to the focal line of the mirror.

- Absorber metal pipe. It absorbs the major part of the energy reflected by the mirror.
- Energy loss to the environment by conduction-convection and radiation.
- Medium model, that is, the HTF. In the case of this work, water-steam is the selected medium.
- Distributed CV, with a discretization level of n in which mass, energy and momentum is conserved.

For modelling effects, this component could be considered as a heat exchanger composed of one pipe with water and/or steam as media fluid, and a circular wall allowing thermal interaction with the fluid. This hex is fed by solar energy through the external perimeter of the circular wall and, at the same time, some energy flow is leaving through this external perimeter by conduction-convection and radiation processes.

The length of the water/steam pipe is 50 m. and under normal operating conditions the inlet/outlet flow

could be in any of the three states: liquid water, two-phase mix of saturated liquid and vapor, or superheated steam. This depends on the position of the PTC in the row, in addition to the incident solar radiation in the row.

Thus the dynamic behavior of each PTC will vary along the DISS row depending on the thermodynamic and transport state of the water/steam in each PTC. With the configuration shown in figure 6, the PTC is fully discretized in n CVs in the major length direction, in which mass, energy, and momentum balances are stated. Momentum conservation is stated in staggered CVs with respect to those ones in which mass and energy balance is stated; see [9], [13] and [11]. The number of CVs, n , is a trade off between accuracy and computing cost, so the final choice is the minimum n that models dominant dynamics for control purposes. Currently we are working with values in the interval [2,5] per PTC. The wall is discretized with the same discretization level.

To solve the PDE system stated from balance equations, ThermoFluid provides partial classes [11] in which the discretization

with the Finite Volume Method (FVM) ([9]) is applied. One of these classes is *ThermoFluid.PartialComponents...Volume2PortDS_pT*, which implements this mass, energy and static momentum conservation equations in a staggered grid formed by n subvolumes. For the solid media, there exists final use classes that implement energy conservation in distributed solids, *ThermoFluid.Components.HeatFlow.Walls*.

To close the system of equations, it is mandatory to introduce the heat transfer coefficient between the water-steam flow and the solid media. This coefficient depends of heat transfer correlations using adimensional fluid numbers (Reynold, Prandtl, Pecklet,...), geometry of the contact surface and thermodynamic and transport properties of the fluid (i.e. water-steam). Some of the correlation parameters strongly depend on the experimentation and parameter adjusting phase of the modelling work. See [10].

In the development of experimental correlations classes for the heat transfer coefficients *sliding modes* have appeared with some frequency around the phase boundaries of water/steam CVs. Those phenomena are manifested with more frequency when CVs pass from subcooled (region 1 in IAPWS-IF97 standard for water/steam properties, [14]) to saturated (region 4 in IAPWS-IF97), due to two reasons: first, is the existence of discontinuities in the heat transfer coefficients in the limit boundary between water and walls; second, is the opposite gradients in the state velocity vectors present around the phase change boundaries. To avoid those cases in which *chattering* causes troubles in the simulation, another polymorphic evaporator model has been developed, in which the subcooled and saturated regions of water/steam pipe are replaced by an equivalent Moving Boundary Model (MBM) [8]. Figure 7 shows this mixed discretized and MBM model for the complete DISS plant, where the MBM component has been designed with ThermoFluid interfaces to be connected with the rest of components.

Although the mixed model lowers the likelihood of finding *chattering* in the integration process, it is theoretically less accurate; and experimentally it is harder to find consistent DAE initial conditions and the validity range of the model is more limited than that of the fully discretized one.

With the help of *replaceable/redeclare* constructs and the *choices annotations* ([1], [5]), switching between fully discretized and mixed MBM-discretized models at instantiation time simplifies the modelling work.

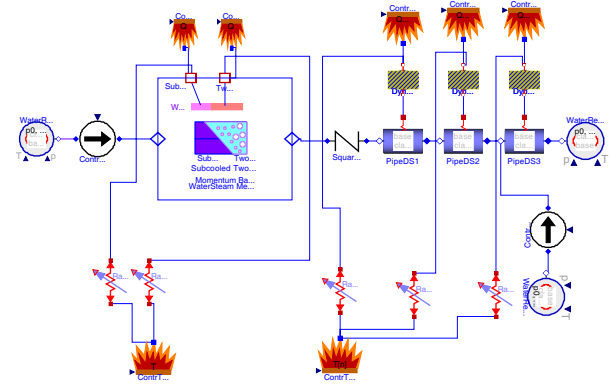


Figure 7: Mixed Moving Boundary and Discretized model of the DISS facility.

3.2.3 Thermo-Hydraulic boundary conditions

When the FVM is applied in thermo-hydraulic modelling, components to define boundary conditions (b.c.) to the original PDE formulated are necessary to close the system of equations. This boundary conditions are implemented in ThermoFluid library by means of reservoirs components, in package *ThermoFluid.PartialComponents.Reservoirs*.

The boundary conditions are defined by reservoirs components that represent pressure, specific enthalpy and temperature b.c. In cases of one-phase fluid (pressure, temperature) pair is selected, and in two-phase fluid the (pressure, specific enthalpy) pair is selected.

4 Simulation and Experimental Validation

In this section the results of a simulation are shown and compared with the corresponding experimental values for some of the variables measured in the actual plant. The experiment was conducted on 1 of April of 2001, starting 9:00 AM and lasting 6 hours and 40 minutes (24000 s.). At which point the DISS plant was forced to the next boundary conditions shown in the top two graphs, shown in figure 8.

In the top graph, the boundary conditions for inlet mass flow rates for the row (\dot{m}_{in}) and for the injector for the PTC number 11 ($\dot{m}_{inj-C11}$) are represented. In the next graph, the boundary condition direct solar irradiance (Solar.Radiation) is shown. From the third to fifth graphs, the measured (MeasuredToutCXX) and simulated (SimulatedToutCXX) outlet temperatures from PTCs nine, ten and eleven are shown. The measured variables are depicted in dashed lines as the simulated in continuous lines.

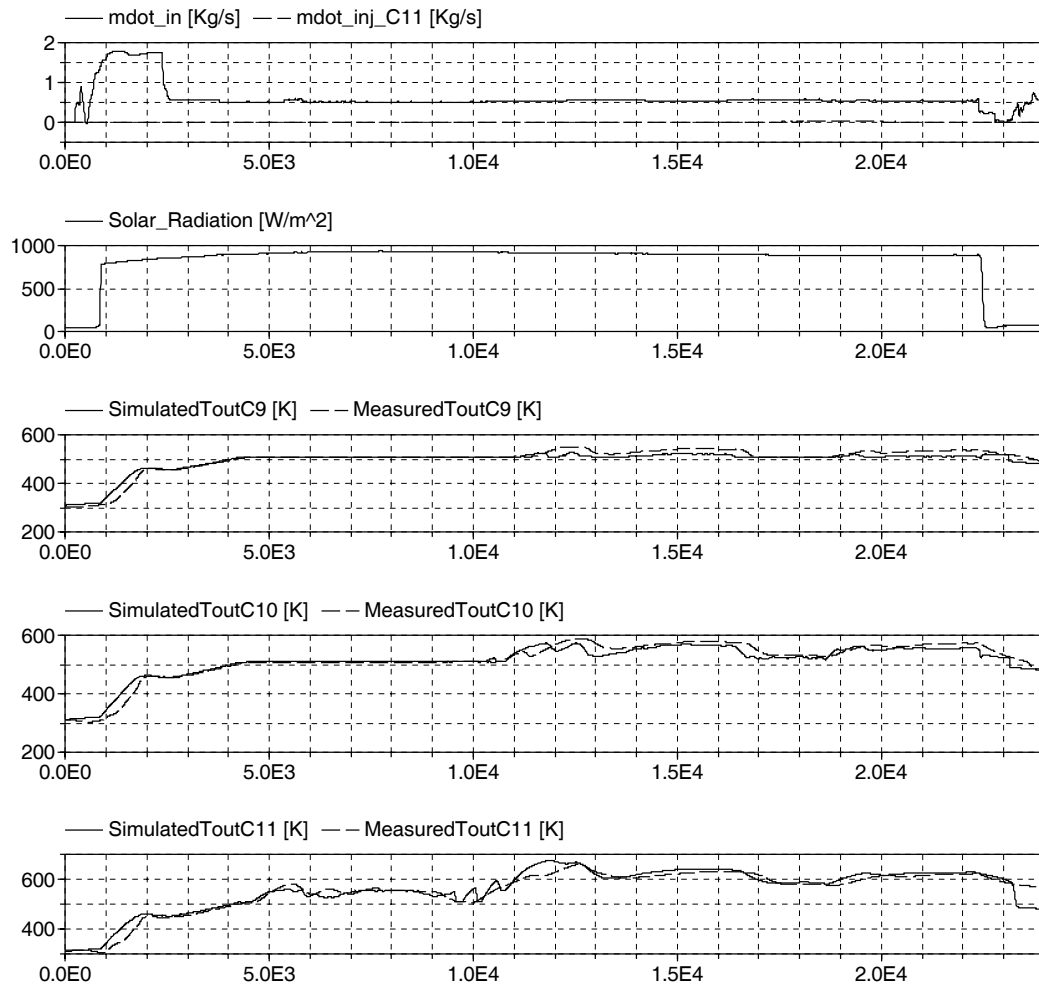


Figure 8: Simulation results for the experiment realized the day 01 of April of 2001. The model is forced to the boundary conditions measured from the experiments.

It can be observed that there is a visible difference in the PTC 9 outlet between the experimental and simulated data when superheating appears. This disagreement is augmented due to the fact that a minimum energy difference for the fluid in the twophase-superheating boundary is reflected in a major temperature differences.

In the case of outlet temperature in PTC 10 and 11, a better agreement between experimental and simulated values can be observed. Although in the time intervals in which superheating appears at the outlet of both PTCs a major difference between simulation and experiment can be observed, the dynamic response is similar in both cases. Then, for control purposes the models can be regarded as valid, although a better calibration would be wellcome. For simulation purposes, the calibration work must be refined in the future with the help of the new features introduced in Dymola 6.0,

like Model Calibration package, which beta release has helped in the calibration work presented in this article.

5 Concluding remarks and Ongoing work

This article shows the development of a dynamic model for the CIEMAT's DISS facility using object oriented modelling methodology. The major part of the components are based in the ThermoFluid framework for thermo-hydraulic modelling. The DISS components and main operation principles have been described. For the main components, the modelling hypotheses and the composition Modelica diagrams developed with the Dymola tool have been presented. References to the underlying physical phenomena have been made in these composition diagrams, with-

out entering into detail of quantitatively describing them through differential and algebraic equations; instead, the basic bibliography and the ThermoFluid classes that implements them have been referenced. Finally, a simulation of the system is executed with the boundary conditions of a experiment. Some temperature results of the model simulation are represented with the corresponding experimental measured values. The graphs show that the differences between the experiments results and predicted values from the model are relatively small.

The ongoing work to develop consists in refining the main models parameter calibration based on the experimental results of the actual plant. In this work, the validation of empirical correlations for heat transfer and pressure loss will be an important issue.

The final aim is to develop control and automatic operation systems that would help this type of plants operate in the most autonomous way and in spite of large disturbances. Automatic start-up and shutdowns of the plants is one of the main objectives in this direction.

Acknowledgements

This work has been financed by CICYT-FEDER funds (projects DPI2002-04375-C03, DPI2004-07444-C04-04 and DPI2004-01804) and by the Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía. This work has been also performed within the scope of the specific collaboration agreement between the Plataforma Solar de Almería and the Automatic Control, Electronics and Robotics (TEP197) research group of the Universidad de Almería titled "Development of Control Systems and Tools for Thermosolar Plants".

Authors would like to acknowledge Dynasim SE for providing the beta release of Model Calibration package to CIEMAT for testing and calibration purposes. In special, to Dr. Sven Erik Mattsson and Dr. Hilding Elmqvist for the support provided in the calibration process.

References

- [1] Modelica Association. Modelica, a unified object oriented language for physical systems modeling. language specification 2.2. Technical report, Modelica Association, <http://www.modelica.org>, 2005.
- [2] K.J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 1997.
- [3] M. Berenguel. *Contribuciones al control de colectores solares distribuidos*. PhD thesis, Escuela Superior de Ingenieros Industriales de Sevilla, España, Marzo 1996.
- [4] E.F. Camacho, M. Berenguel, and F.R. Rubio. *Advanced Control of Solar Plants*. Springer, 1997.
- [5] A.B. Dynasim. *Dymola 5.3 User Manual*. Dynasim, A.B., <http://www.dynasim.se>, 2004.
- [6] J. Eborn. *On Model Libraries for Thermo-hydraulic Applications*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, March 2001.
- [7] L. González, E. Zarza, and L. J. Yebra. Determinación del modificador por Ángulo de incidencia de un colector solar ls-3, incluyendo las pérdidas geométricas por final de colector. Technical Report DISS-SC-SF-29, CIEMAT-PSA. Almería. España, Agosto 2001.
- [8] J.M. Jensen and H. Tummescheit. Moving boundary models for dynamic simulations of two-phase flows. In Martin Otter, editor, *Proc. of the 2nd Int. Modelica Conference*, pages 235–244, Oberpfaffenhofen. Germany, March 2002.
- [9] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow. Series in Computational and Physical Processes in Mechanics and Thermal Sciences*. Taylor & Francis, 1980.
- [10] W.M. Rohsenow, J.P. Hartnett, and Y.I. Cho. *Handbook of Heat Transfer*. McGraw-Hill, 1998.
- [11] H. Tummescheit. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, August 2002.
- [12] L. Valenzuela, E. Zarza, M. Berenguel, and E.F. Camacho. Direct steam generation in solar boilers. *IEEE Control Systems Magazine*, 24:15–29, April 2004.
- [13] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Addison Wesley Longman Limited, 1995.
- [14] W. Wagner and A. Kruse. *Properties of water and steam*. Springer-Verlag, Berlin, 1998.
- [15] E. Zarza. Aplicaciones industriales de los colectores cilindro-parabólicos. Technical Report R09/01EZ, CIEMAT-PSA, Crta. Senés s/n. 04200 Tabernas. Almería. España, 2001.
- [16] E. Zarza. *La Generación Directa de Vapor con Colectores Cilindro-Parabólicos. El Proyecto DISS*. PhD thesis, ESII Sevilla, España, Noviembre 2003.

Session 5a

Language, Tools and Algorithms 4

OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging

Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, David Akhvlediani
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, S-581 83 Linköping, Sweden
{adrpo, petfr}@ida.liu.se

Abstract

The OpenModelica (MDT) Eclipse Plugin integrates the OpenModelica compiler and debugger with the Eclipse Integrated Development Environment Framework. MDT, together with the OpenModelica compiler and debugger, provides an environment for Modelica development projects. This includes browsing, code completion through menus or popups, automatic indentation even of syntactically incorrect models, and model debugging. Simulation and plotting is also possible from a special command window. To our knowledge, this is the first Eclipse plugin for an equation-based language.

1 Introduction

The goal of our work with the Eclipse framework integration in the OpenModelica modeling and development environment is to achieve a more comprehensive and more powerful environment. It can be useful to first take a general look at this area including some background.

1.1 Integrated Interactive Programming Environments

An integrated interactive modeling and simulation environment is a special case of programming environments with applications in modeling and simulation. Thus, it should fulfill the requirements both from general integrated environments and from the application area of modeling and simulation mentioned in the previous section.

The main idea of an integrated programming environment in general is that a number of programming support functions should be available within the same tool in a well-integrated way. This means that the functions should operate on the same data and program representations, exchange information when necessary, resulting in an environment that is both powerful and easy to use. An environment is interactive and incre-

mental if it gives quick feedback, e.g. without recomputing everything from scratch, and maintains a dialogue with the user, including preserving the state of previous interactions with the user. Interactive environments are typically both more productive and more fun to use.

There are many things that one wants a programming environment to do for the programmer, particularly if it is interactive. What functionality should be included? Comprehensive software development environments are expected to provide support for the major development phases, such as:

- Requirements analysis.
- Design.
- Implementation.
- Maintenance.

A programming environment can be somewhat more restrictive and need not necessarily support early phases such as requirements analysis, but it is an advantage if such facilities are also included. The main point is to provide as much computer support as possible for different aspects of software development, to free the developer from mundane tasks, so that more time and effort can be spent on the essential issues. The following is a partial list of integrated programming environment facilities, some of which are already mentioned in (Sandewall 1978 [11]), that should be provided for the programmer:

- Administration and configuration management of program modules and classes, and different versions of these.
- Administration and maintenance of test examples and their correct results.
- Administration and maintenance of formal or informal documentation of program parts, and automatic generation of documentation from programs.
- Support for a given programming methodology, e.g. top-down or bottom-up. For example, if a top-down approach should be encouraged, it is natural for the interactive environment to maintain successive

composition steps and mutual references between those.

- Support for the interactive session. For example, previous interactions should be saved in an appropriate way so that the user can refer to previous commands or results, go back and edit those, and possibly re-execute.
- Enhanced editing support, performed by an editor that knows about the syntactic structure of the language. It is an advantage if the system allows editing of the program in different views. For example, editing of the overall system structure can be done in the graphical view, whereas editing of detailed properties can be done in the textual view.
- Cross-referencing and query facilities, to help the user understand interdependences between parts of large systems.
- Flexibility and extensibility, e.g. mechanisms to extend the syntax and semantics of the programming language representation and the functionality built into the environment.
- Accessible internal representation of programs. This is often a prerequisite to the extensibility requirement. An accessible internal representation means that there is a well-defined representation of programs that are represented in data structures of the programming language itself, so that user-written programs may inspect the structure and generate new programs. This property is also known as the principle of program-data equivalence.

Early work in interactive integrated programming environments supporting a specific language was done in the InterLisp system for the Lisp language: (Teitelman 1974 [12]), common principles and experience of early interactive Lisp environments are described in (Sandewall 1978 [11]), interactive and incremental Pascal with the DICE system: (Fritzson 1983 [3]), the integrated Mjölner environment, (Lindskov, Knudsen, Lehrmann-Madsen, and Magnusson 1993 [9]).

1.2 The Eclipse Framework

Eclipse [1] is an open source framework for creating extensible integrated development environments (IDEs). (For the history of Eclipse, see Section 6). One of the goals of the Eclipse platform is to avoid duplicating common code that is needed to implement a powerful integrated environment for development of software. By allowing third parties to easily extend the platform via the plugin concept, the amount of new code that needs to be written is decreased.

1.3 Eclipse Platform Architecture

By itself, Eclipse does not provide much end-user functionality. The important contributions to Eclipse are based on its plugins. The smallest architectural unit of the Eclipse platform is the plugin.

At the core of Eclipse is the Eclipse Platform Runtime. The Runtime in itself mostly provides the loading of external plugins. The Java Development Tooling (JDT) is for example a collection of plugins that are loaded into Eclipse when they are requested. The fact that Eclipse is in itself written in Java and comes with the Java Development Tooling as default often leads newcomers to believe that Eclipse is a Java IDE with plugin capabilities. It is in fact the other way around, with Eclipse being just a base for plugins, and the Java Development Tooling plugging into this base.

To extend Eclipse, a set of new plugins must be created. A plugin is created by extending a certain extension point in Eclipse. There are several predefined extension points in Eclipse, and plugins can provide their own extension points. This means that you can plug in plugins into other plugins.

An extension point can have several plugins attached, and the plugin that will be used is determined by a property file. For example, the Modelica Editor is loaded at the same time as the Java Editor is loaded. When a user opens a Java file, the Java Editor will be used, based on a property in the Java Editor extension. In this case, it is the file name extension that selects what editor should be used.

As the number of plugins in Eclipse can be very large, a plugin is not actually loaded into memory before its contribution is directly requested by the user. This design makes the memory impact reasonably low while running Eclipse.

A user-friendly aspect of Eclipse is the Eclipse Update Manager which allows the user to install new plugins just by pointing Eclipse to a certain website. This website is provided by the developers of the plugin that the user may wish to install. An update site at the OpenModelica [13] web site is for example provided for easy installation of the latest version of MDT.

1.4 OpenModelica MDT Plugin into Eclipse

The MDT Eclipse plugin provides file and class hierarchy browsing and text editing capabilities. Syntax highlighting facilities and a compilation manager are also included in MDT, as well as integration of the debugger for the algorithmic Modelica code.

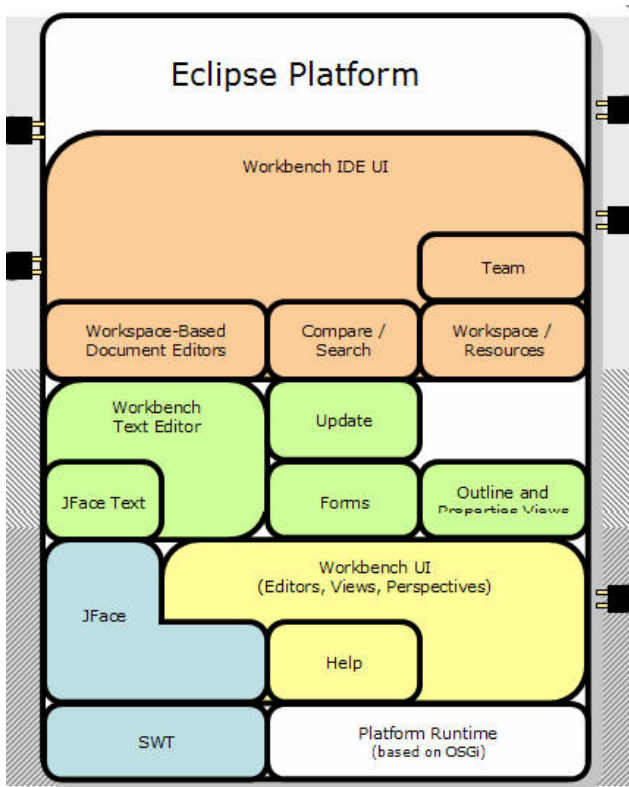


Figure 1. The architecture of Eclipse, with possible plugin positions marked.

The Eclipse framework (Figure 1) has the advantage of making it easy to add future extensions.

2 OpenModelica Environment Architecture

The MDT Eclipse plugin is integrated in the OpenModelica environment which consists of several interconnected subsystems, as depicted in Figure 2 below.

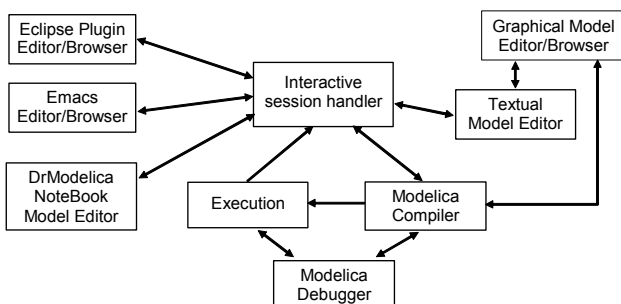


Figure 2. The architecture of the OpenModelica environment.

Arrows denote data and control flow. Several subsystems provide different forms of browsing and textual editing of the Modelica code.

OpenModelica is structured as several communicating processes in client-server architecture, primarily exchanging information through a Corba interface, see Figure 3. The OpenModelica compiler/interpreter (OMC) is the server, communicating with clients. The Eclipse MDT plugin is one of the clients.

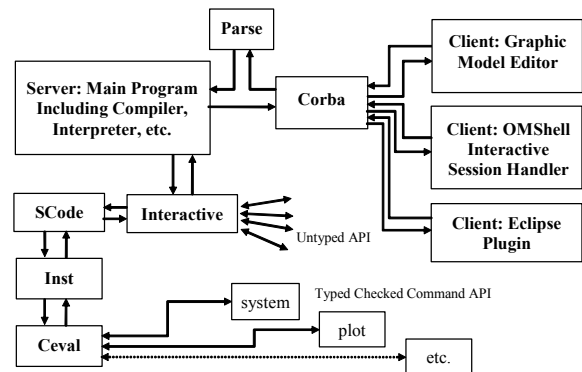


Figure 3. The client-server architecture of the OpenModelica environment.

Messages from the Corba interface are of two kinds. The first group consists of expressions or user commands which are evaluated by the Ceval module. The second group includes declarations of classes, variables, etc., assignments, and client-server API calls that are handled via the Interactive module, which also stores information about interactively declared/assigned items at the top-level in an environment structure

3 Modelica Development Tooling (MDT) Eclipse Plugin

As mentioned, the Modelica Development Tooling (MDT) Eclipse Plugin provides an environment for working with Modelica development projects.

The following features are available:

- Browsing support for Modelica projects, packages, and classes.
- Wizards for creating Modelica projects, packages, and classes.
- Syntax color highlighting.
- Syntax checking.
- Code completion when writing code to reference a class.
- Code completion/signature information when writing function calls.
- Browsing of the Modelica Standard Library and other Modelica package hierarchies.
- Support for MetaModelica extensions to standard Modelica.

3.1 Using the Modelica Perspective

The most convenient way to work with Modelica projects is to use the Modelica perspective. To switch to the Modelica perspective, choose the Window menu item, select Open Perspective followed by Other... Select the Modelica option from the dialog presented and click OK.

3.2 Creating a Project

To start a new project, use the New Modelica Project Wizard. It is accessible through File->New->Modelica Project or by right-clicking in the Modelica Projects view and selecting New->Modelica Project.

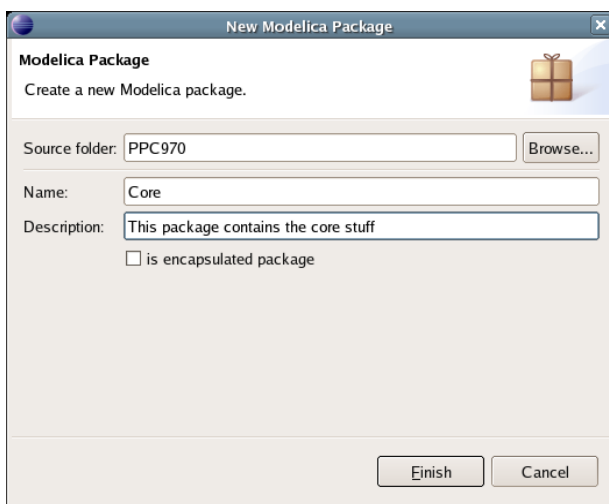


Figure 4. Creating a new package.

3.3 Creating a Package

To create a new package inside a Modelica project, select File->New->Modelica Package. Enter the desired name of the package and a description of what it contains.

3.4 Creating a Class

To create a new Modelica class, select where in the hierarchy that you want to add your new class and select File->New->Modelica Class. When creating a Modelica class you can add different restrictions on what the class can contain. These can for example be model, connector, block, record, or function.

When you have selected your desired class type, you can select modifiers that add code blocks to the generated code. 'Include initial code block' will for example add the line 'initial equation' to the class.

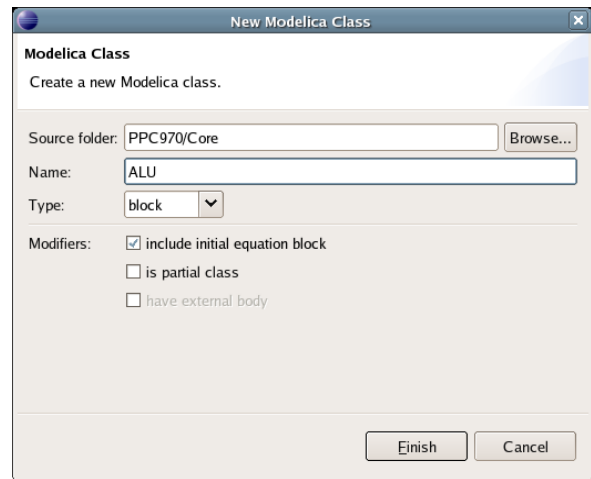


Figure 5. Creating a new class.

3.5 Syntax Checking

Whenever a Modelica (.mo) file is saved by the Modelica Editor, it is checked for syntactical errors. Any errors found are added to the Problems view and also marked in the source code editor.

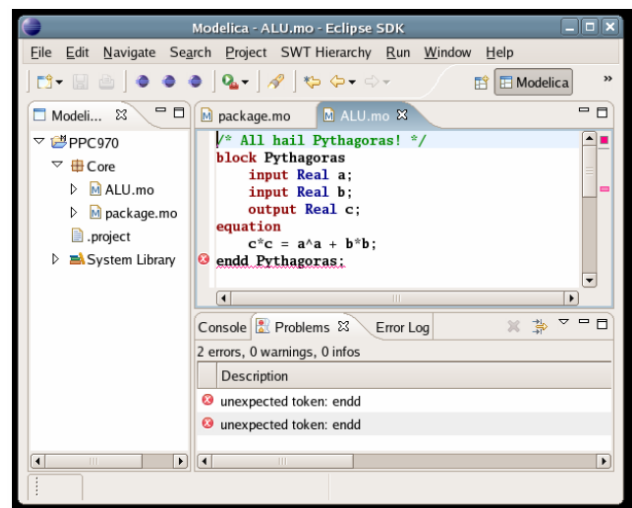


Figure 6. Syntax checking.

Errors are marked in the editor as a red circle with a white cross, a squiggly red line under the problematic construct, and as a red marker in the right-hand side of the editor. To reach the problem, one can either click the item in the Problems view or select the red box in the right-hand side of the editor.

3.6 Code Completion

MDT supports Code Completion in two variants. The first variant, code completion when typing a dot after a class (package) name, shows alternatives in a menu:

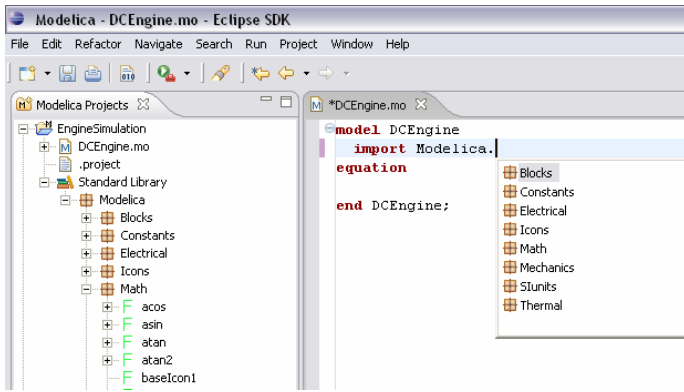


Figure 7. Code completion using a popup menu after a dot

The second variant is useful when typing a call to a function. It shows the function signature (formal parameter names and types) in a popup when typing the parenthesis after the function name, here the signature `Real sin(SI.Angle u)` of the `sin` function:

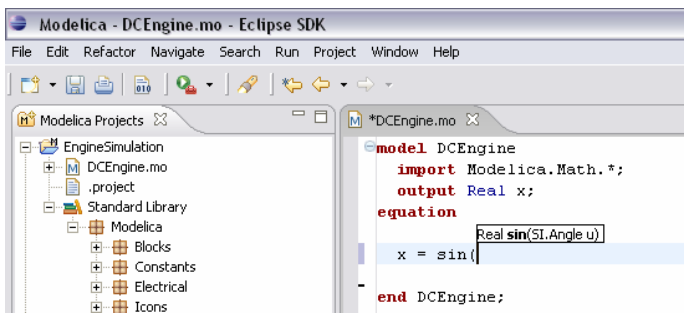


Figure 8. Code completion showing a popup function signature after typing a left parenthesis.

3.7 Automatic Indentation

MDT has recently obtained support for automatic indentation. When typing the Return (Enter) key, the next line is indented correctly. You can also correct indentation of the current line or a range selection using CTRL+I or “Correct Indentation” action on the toolbar or in the Edit menu.

Indentation can be applied to incomplete code as a heuristic Modelica scanner is used and the indentation is based only on the tokens generated by this scanner. The indenter indents one line at a time. For example, consider that line four (4) in Figure 10 should be indented. The indenter asks the heuristic scanner to give tokens from the starting token in backwards direction to the start of the file until a scope introducer is recognized, which for this particular file is `model MoonAndEarth`. The reference position of the start of the scope introducer is computed and line four (4) is indented from this reference position one indent unit. The indentation result is presented in Figure 10.

Indenting Modelica code is far from trivial when incomplete (possibly incorrect) code should be indented correctly. Most of the difficulty comes from Modelica

scopes which are hard to recognize using just a scanner and some logic behind it. In languages like C/C++ and Java finding enclosing scopes is very easy as one character tokens are used for the scope opening and closing: “{” and “}”. In Modelica you need at least two tokens and much more case analysis to find where a scope starts and ends. Complications also arise when mixing if-statements with if-expressions (which was further complicated by the introduction of conditional declarations in the Modelica language). In this particular case we implemented a parser emulator that recognizes these constructs based on scanner tokens delivered backwards.

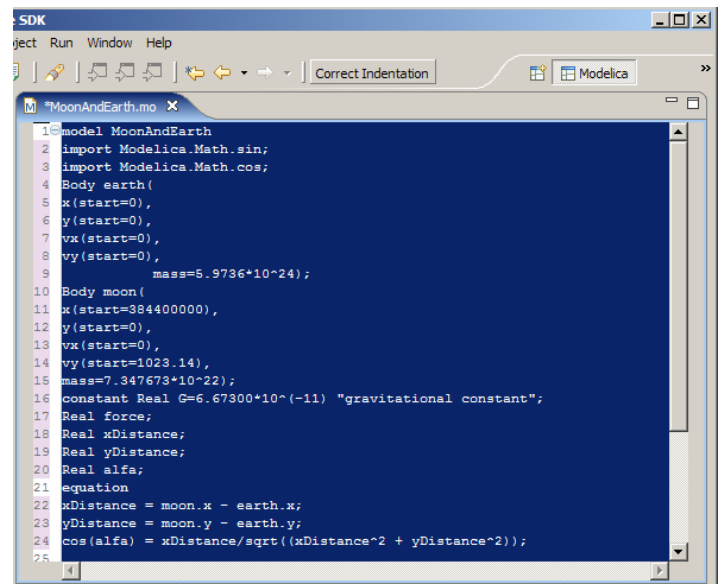


Figure 9. Example of code before indentation.

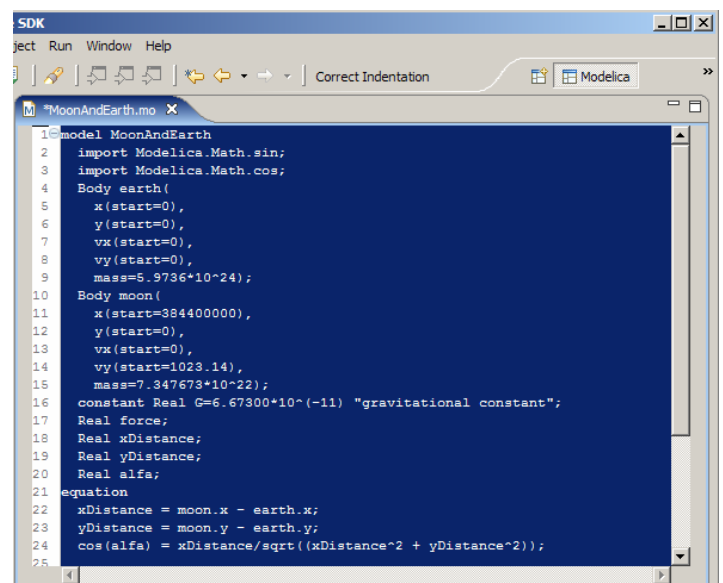


Figure 10. Example of code after automatic indentation.

The indenter works in almost all cases, but there are cases in which is impossible to find the correct indentation. For example when the indentation of a line con-

sisting of "end Name;" is requested and the scope introducer for Name is not found (that is identifier Name followed backwards by class, model, package, block, record, connector etc.) then the indenter fails and returns the indentation of the previous line.

4 The OpenModelica Debugger Integrated in Eclipse

We have integrated our debugger for algorithmic Mod-
elica code (Adrian Pop and Peter Fritzson, 2005 [10])
within the Eclipse debugging framework.

The communication protocol between MDT and the
debugger (which is included in the compiled executable
built for simulation) is based on a client-server archi-
tecture and is implemented via sockets. The debugger
is the server and MDT is the client.

When the debugged model is simulated, the debug-
ger receives from MDT all the breakpoints set within
the algorithmic code. Then the debugger resumes the
application program. When a breakpoint condition be-
comes true the debugger stops the program and listens
to commands it may receive from MDT. The com-
mands accepted by the MDT client are classic: variable
value printing, stack trace printing, stepping, running,
etc. MDT sends appropriate commands to the debug-
ger, parses the information received and displays it
within the MDT debugging views to be inspected by
the programmer.

Because algorithmic code can be executed millions
of times within a simulation, it is very important to be
able to specify breakpoints based on variable values
and/or the number of times a function executes. These
types of breakpoints were recently added to the debug-
ging framework previously described in (Adrian Pop
and Peter Fritzson, 2005 [10]) and are now available,
also in MDT within Eclipse.

5 Simulation and Plotting from MDT

Simulation and plotting is possible from a special
command window, where commands are sent to omc.
For example:

Simulation:

```
>> simulate(Influenza,startTime=0.0,
stopTime=3.0)
record
    resultFile = "Influenza_res.plt"
end record
```

The simulated population is plotted (Figure 11).

```
>> plot({Infected_Popul.p})
true
```

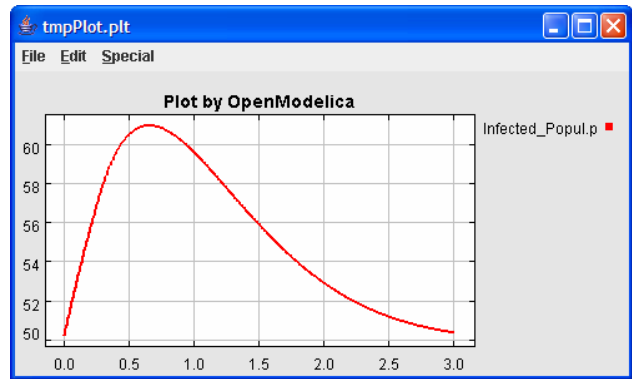


Figure 11. Plot of the Influenza model.

6 Eclipse History

In the mid 1990s software developments tools were
primarily dominated by systems built around two tech-
nologies. Many of the tools were focused on a runtime
environment developed and controlled by the Microsoft
Corporation. The other was built around the Java plat-
form. The Java platform is less dominated by a single
company and more open to industry and community
input. IBM felt it was important to contribute to the
growth of the more open Java platform to become in-
dependent of Microsoft.

By creating a common platform for development
tools built on top of the Java platform, IBM hoped to
attract more developers from competing environments.
In late 1998, the software division at the IBM Corpora-
tion began working on the software project that is today
known as Eclipse. The original work was based on re-
sources developed by Object Technology International
labs. In the beginning, the work on a new Java IDE was
performed at the IBM laboratories. At the same time
additional teams were setup by IBM to build other
products on top of the platform.

In order to increase the rate of adaptation of the
platform and to instill confidence in the Eclipse plat-
form, IBM decided to release the code base under an
open source license, and to build a community around
the project.

In 2001, IBM together with eight other organiza-
tions created the Eclipse consortium. A website at
eclipse.org was started in order to create and coordi-
nate a community around Eclipse. The goal was that
source code would be controlled and developed by the
open source community and the consortium would
handle the marketing and business side of the project.

At that point, IBM was the largest contributor to
both the open source community and the consortium.
Two years later the first major public release of the
Eclipse platform was made. The release got a lot of
attention from developers and was well received. How-

ever, industry analysts suggested that many still perceived Eclipse as an IBM-controlled technology. Many key players in the industry did not want to make commitments to a project controlled by the International Business Machines Corporation.

After discussions within the consortium it was decided that a new organization was needed to make the status of Eclipse as an open and community driven project clear. At the EclipseCon 2004 gathering an announcement was made that the Eclipse Foundation was formed. The foundation is an independent not-for-profit organization. It has its own full time paid professional staff, supported by foundation members.

The new organization has proven itself a success. At this point the foundation have released version 3.0 and 3.1 of Eclipse since its birth. These releases have gained more adoption and recognition than any earlier versions. Today (2005) the foundation has more than 90 full-time developers on the pay roll and receives more than \$2 millions in funding each year.

Currently there are more than eighty member companies in the foundation of which at least sixty-nine are providing add-on products to Eclipse. Today there exists hundreds of proprietary and an even greater number of free plugin products. Eclipse has gained a strong foothold in the industry and is one of the major open source software development platforms [2].

7 Conclusion

The OpenModelica integrated development environment for Modelica has been augmented with a plugin to the Eclipse framework. The plugin, called MDT (Modelica Development Tooling) [13], an earlier version is described in [14], is primarily aimed at development of large models. It has support for browsing, editing, code completion, automatic indentation, building executables, and debugging. It also allows simulation and plotting from a special command window. To summarize, it provides a rather complete integrated development environment, and it is also the first available Eclipse plugin for an equation-based language.

8 Acknowledgements

This work was supported by Vinnova in the SWEB-Prod project, by the CUGS graduate school, and by MathCore Engineering AB.

References

- [1] Eclipse website. <http://www.eclipse.org>.
- [2] Eclipse history. A brief history of eclipse. <http://www-128.ibm.com/developerworks/rational/library/nov05/cernosek/>
- [3] Peter Fritzson. Symbolic Debugging through Incremental Compilation in an Integrated Environment, *Journal of Systems and Software*, 3, pp. 285–294, 1983.
- [4] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, David Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005)*, Trondheim, Norway, October 13-14, 2005.
<http://www.ida.liu.se/projects/OpenModelica>
- [5] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.
- [6] The Modelica Association. The Modelica Language Specification Version 2.2, March 2005. <http://www.modelica.org>.
- [7] Peter Fritzson et al. The OpenModelica Users Guide, version 0.7, May 2006.
<http://www.ida.liu.se/projects/OpenModelica>.
- [8] Peter Fritzson et al. The OpenModelica System Documentation, version 0.7, May 2006.
<http://www.ida.liu.se/projects/OpenModelica>.
- [9] J. Lindskov, M. Knudsen, O. Löfgren, Ole Lehmann-Madsen, and Boris Magnusson (Eds.). *Object-Oriented Environments - The Mjølner Approach*. Prentice Hall, 1993.
- [10] Adrian Pop and Peter Fritzson. A Portable Debugger for Algorithmic Modelica Code, the 4th International Modelica Conference (Modelica2005), March 7-9, 2005, Hamburg, Germany.
- [11] Erik Sandewall. Programming in an Interactive Environment: The “LISP” Experience, *Computing Surveys*, 10:1, Mar. 1978.
- [12] Warren Teitelman. *INTERLISP Reference Manual*. Xerox Palo Alto Research Center, Palo Alto, CA, 1974.
- [13] PELAB, *Modelica Development Tooling (MDT)*.
<http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/MDT>
- [14] Andreas Remar and Elmir Jagudin. Modelica Development Tooling for Eclipse. Master Thesis LITH-IDA-EX-06/024-SE, April 10, 2006.

A Modelica-based Format for Flexible Modelica Code Generation and Causal Model Transformations

Jonas Larsson⁺Peter Fritzson^{*}⁺ Fluid- and Mechanical Engineering Systems, Dept. Mechanical Engineering^{*} PELAB – Programming Environments Lab, Dept. Computer Science

Linköping University, S-581 83 Linköping, Sweden

⁺ jonla@ikp.liu.se ^{*} petfr@ida.liu.se

Abstract

The equation-based modeling language Modelica offers the possibility to extract a differential algebraic equation system (DAE). The DAE in turn can be used in numerical simulation, optimization, sensitivity analysis, diagnosis and more. For each of these categories, there exist software tools that all require slightly different input data, sometimes in symbolic form. Through examples, this paper briefly outlines a Modelica based format which can be used as input to the final code generation. This could yield a modular Modelica compiler in which the creation of diverse code generators is encouraged, which in turn widens the application area for Modelica. However, no formal definition of the format is presented here..

The paper also briefly addresses the issue of possible model causal adaptation needed at the equation level to make the model fit into target application with a causal usage context and how to automate the inclusion of these changes for seamless translation.

Keywords: Modelica, DAE, numerical solver, code generation, DSblock

1 Introduction

Modelica [1][8] is one of several equation-based modeling languages for describing continuous dynamic behavior and discrete events in the context of technical systems. A Modelica model can be flattened from its object-oriented form to a hybrid DAE, i.e., a DAE containing both continuous and discrete variables. One example of a model resulting in a hybrid DAE is that of a chemical process in a plant including a software-based controller and actuated on/off valves. The hybrid DAE is normally optimized such that numerical time-domain solutions will converge rapidly at a low computational cost.

Finally, code generation takes place where the actual programming code is created for the application at hand, for example a simulation, optimization, sensitivity analysis, or diagnosis.

This compiling process is complex and among other things includes an extensive flattening procedure, efficient symbolic manipulations and sorting algorithms for a large number of equations. The creation demands a deep knowledge in the Modelica semantics as well as in the area of solving DAEs. As an example, the OpenModelica compiler [2], programmed in a Modelica-like syntax [3] contains some 100 000 lines of code.

In order to facilitate and encourage a widened cooperation in Modelica compiler construction, this paper discusses a Modelica based format for the information available between the optimization and code generation steps, called MOO from now on. The information is essentially several equation sets for initialization, numerical integration, re-initialization at discrete events, and finally output. Today there already exist a Modelica format between flattening and optimization that is a subset of Modelica since the object-oriented syntax is not needed at that stage.

The equation-based MOO format exemplified in this paper enables a discussion between compiler designers and the simulation tool developers at the other end on the interface information needed. This can lead to widened support for Modelica in simulation tools and to new solvers, etc. due to a more modular development of code generators.

The second but smaller subject of the paper is on adapting Modelica models in terms of interface variables and the equation set to fit a target application with a specific causality context, such as a model library with conventions for the exchanged variables among models in terms of computational causality, signs, and names. Through the algorithms described,

a model can be fitted with new interface variables and equations if needed.

2 Related Work

The file based approach MOO has its disadvantages, such as low performance in comparison to a monolithic compiler since the internal data structure needs to be converted to and from the Modelica syntax. On the other hand, a programmer does not need to get acquainted with the rest of the compiler code and could save time this way. Moreover, flattened Modelica of the OpenModelica compiler is already used today as input to third party compilers.

Another alternative would be to keep the current situation where the generated C code is utilized through function calls to get access to derivatives, output values etc, as is performed in DSblock [4] and SimStruct [5]. However, this C code can be generated from MOO, so the MOO file is an inclusive option. Also, the advantage with introducing MOO is that the equation sets produced are available on symbolic form, which is important in order to create new types of applications where the equations need to be processed further through symbolic mathematics and other operations. One example is optimization, where the Hessian (matrix of second-order partial derivatives) is of interest. Many mathematical operations can be performed numerically, such as creating the Hessian, but often a mixed symbolic-numeric approach results in higher accuracy and better performance.

The paper [6] describes the syntax and semantics of an interchange format for hybrid models in general and is thus at a higher, more abstract level than MOO. MOO is however specified with [6] in mind not at least since compatibility between the two means that the code generators implemented for Modelica can be applied for other hybrid modeling languages as well with minor effort.

3 Limitations

Algorithms, arrays, and function calls of the Modelica language are not the main concern of this paper but are at times commented in terms of implementation.

4 Hybrid Modeling in Modelica

In this chapter the basics of hybrid modelling in Modelica is described as well as the example model used throughout the text. In the following two sections, the resulting typical information from the flattening and optimization steps is described as well as the information needed for code generation with different applications in mind.

A bouncing ball is a good example of a hybrid system. The motion of the ball is characterized by the height h and the vertical velocity v . The ball moves continuously between bounces, whereas discrete changes occur at bounce times, as depicted in Figure 1 below. When the ball bounces against the ground its velocity is reversed. An ideal ball would have an elasticity coefficient e of 1 and would not lose any energy at a bounce. A more realistic ball, as the one modelled below, has an elasticity coefficient of 0.9, making it keep 90 percent of its speed after the bounce.

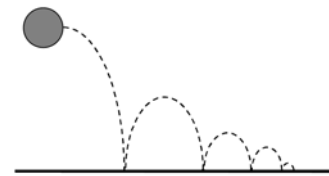


Figure 1. A bouncing ball

The bouncing ball model contains the two basic equations of motion relating height and velocity as well as the acceleration caused by the gravitational force. At the bounce instant where the boolean `impact` variable switches value, the velocity is suddenly reversed and slightly decreased, i.e., $v(\text{after bounce}) = -e \cdot v(\text{before bounce})$, which is accomplished by the special syntactic form of instantaneous equation: `reinit(v, -e*pre(v))`. The `pre` operator returns the value of the variable argument just before the latest event, in this case the velocity just before impact. The edge operator used in the example is simply defined as: `var` and `not pre(var)` and thus indicates a change in the discrete variable `var`.

```
model BouncingBall
  parameter Real e=0.7 "bounce coeff";
  parameter Real g=9.81 "gravity acc.";
  Real h(start=1) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start=true);
  Boolean impact;
  Real v_new;
equation
  impact = h <= 0.0;
  der(v) = if flying then -g else 0;
  der(h) = v;
```

```

when {h <= 0.0 and v <= 0.0, impact} then
  v_new = if edge(impact) then
    -e*pre(v)
  else
    0;
  flying = v_new > 0;
  reinit(v, v_new);
end when;
end BouncingBall;

```

The equations within the `when`-equation are executed whenever any of the given boolean expressions changes values. Thus, if the ball is on the ground and has a negative velocity but the impact variable has not changed the velocity will be reset to zero and the boolean variable `flying` is set to false. The derivative of the velocity in the equation section is then also set to zero due to the `if`-expression.

5 Model Translation to Solvable Mathematical Representation

A Modelica model is first flattened from its object oriented form to a set of hybrid differential and algebraic equations on the hybrid DAE implicit form [8]

$$F(\dot{x}(t), x(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) = 0 \quad (1)$$

The vector $q(t_e)$ is discrete-time variables and the corresponding predecessor variable vector $q_{pre}(t_e)$ is denoted `pre(q)` in Modelica. The time t_e used instead of t for these variables indicate that such variables may only change value at event time points t_e . The time t and constant vector p of parameters and constants are made explicit in the equation. The vector $c(t_e)$ is the conditional expressions from for example `if` and `when` constructs, evaluated at the most recent event.

Equation (1) can contain explicit or implicit algebraic relations among the states x which leads to difficulties in DAE solvers since numerical differentiation is needed. Index reduction [7][8] is therefore applied, which means that certain equations of the DAE are differentiated symbolically wrt time. The result can be brought to the form

$$f(\dot{x}(t), x(t), u(t), y_f(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) = 0 \quad (2a)$$

$$g(x(t), u(t), y_f(t), y_g(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) = 0 \quad (2b)$$

where $\partial f / \partial \dot{x}$, $\partial f / \partial y_f$ and $\partial g / \partial y_g$ are nonsingular but most often nonlinear. The derivatives \dot{x} and algebraic variables y_f are found by solving Eq. 2a after which 2b is used for computing y_g , which are output variables. Equation 2 is supplied to an ODE or DAE solver in order to compute x and y in between events.

Equation 2 is the presumed resulting equation set from the optimization phase of the model compiling process.

Apart from the equations in (2), there exist assignments of discrete variables in normal equations as well as in the instantaneous `when` equations which are active only at events. These assignments can be written

$$q(t_e) := f_q \left(\dot{x}(t_e), x(t_e), u(t_e), y_f(t_e), y_g(t_e), t_e, q(t_e), q_{pre}(t_e), p, c(t_e) \right) \quad (3)$$

The boolean conditions vector is stated [8] as follows:

$$c(t_e) := f_c \left(q^B(t_e), q_{pre}^B(t_e), p^B(t_e), rel(v(t_e)) \right) \quad (4)$$

The function f_c takes as arguments the subset of discrete variables and parameters of boolean type and the boolean vector rel contains all elementary relations of the model. Two possible relations are $\{a > b, c < a\}$ which yields a possible $c(t_e)$ as $\{a > b$ and $\text{not}(c < a), a > b$ or $c < a\}$.

6 Modelica Representation for Input to Code Generation

A simulation comprises initialization in order to reach consistent start values for dependent variables, numerical integration between events, event detection during integration by detecting changes in the boolean relations rel , solving both continuous and discrete variables at events, and termination. The MOO file contains information needed for all these operations and is described in this section. The information in the MOO file is based on how the present OpenModelica compiler is implemented and on [5] and [8] - [11].

Since Modelica algorithms demand that all variables on the right hand side are defined, a Modelica model is more suitable than a function since the model can contain equations as well. The model below is filled with some of the Modelica statements discussed from now on.

```

model PreCodeInfo
  // Parameter, constant and
  // variable declarations
equation
  // Equations
end PreCodeInfo;

```

6.1 Constants and Parameters

The parameter and constant declarations from the flattened Modelica model are replicated. Attributes such as minimum and maximum values and units are important information since they can be used for error control in the generated code. Parameters such as `notset` below with no fixed values are computed in the initialization described later. This is also true for bound parameters, i.e., computed from other parameters.

```
parameter Real e=0.7;
parameter Real g=9.81;

parameter Real other=e*g(min=...);
parameter Real notset(fixed=false);
```

6.2 Variable Declarations

Variables have start values and at times a prefix stating whether the variable is discrete, input, or output. States are found by searching equations for the `der` operator applied to the variable in question in some equation. The variable declarations for the bouncing ball model are:

```
discrete Real v_new;
discrete Boolean impact;
discrete Boolean flying(start=true);
Real v;
Real h;
```

6.3 Initialization

Before simulation, the initial values for all variables, including states and nonfixed parameters, need to be computed. A similar procedure takes place after an event has occurred. Initial equations (from the `initial` equation construct in Modelica) are part of the total initial system of equations to be solved as well as when-equations with the `initial()` condition and equations formed from initial values of variables with the `fixed` attribute set to `true`.

An equation list is formed containing all model equations, excluding those within when-clauses without an `initial()` condition. Also, for continuous variables `v` with `start=startExp` and `fixed=true`, equations on the form `v=startExp` are added. For discrete variables under the same conditions, equations are added in which `pre(v) = startExp`. If `fixed=false` then `startExp` is only used as start value for the initialization, as is the case here. The when equation `reinit(v,v_new)` is treated as `v=v_new` in an initial equation if the when-condition contains `initial()`. For all when-equations with no `initial()` in the when-conditions, `v=pre(v)` is added for all `v := expr`.

It may very well be that the initialization is not a well posed problem, i.e., that the number of equations differs from the number of dependent variables. In the bouncing ball model case, the start value of `h` needs to be made fixed and `v` has no start value altogether. The discrete variables `v_new` and `flying` are initialized as well using default value 0 and start value `true`.

The resulting set of equations is shown below where both start values of discrete and continuous variables are computed. Two algorithms for finding a zero free diagonal [12] and then transforming into a block lower triangular form [13] have been applied. This results in a sequence of equation systems that can be solved more efficiently than the complete implicit equation system. The first index in the equations below is the block and the other the equation number. Where possible, the variables have been solved for and are given as the first element in the list, followed by the equation right hand side and then a string commenting on whether the second argument is the right hand side of an assignment or the residual.

```
iEq[1,1] = {pre(v_new), 0, "ass"};
iEq[2,2] = {v_new, pre(v_new), "ass"};
iEq[3,3] = {pre(flying), true, "ass"};
iEq[4,4] = {flying, pre(flying), "ass"};
iEq[5,5] = {h, 1, "ass"};
iEq[6,6] = {v, 0, "ass"};
iEq[7,7] = {der(h), v, "ass"};
iEq[8,8] = {impact, h<=0.0, "ass"};
iEq[9,9] = {der(v), if flying then -g else 0, "ass"};
```

For equations that cannot be solved explicitly, such as `nonlinear(v)=expression`, the equation is simply written in the residual form, shown below. Any other residual is formed by subtracting the second argument of the equation expressions above from the solved variable, such as `pre(flying) - true` in the third equation.

```
iEq[1] = {v, nonlinear(v)-expression, "res"};
```

In the bouncing ball model case, the variables can be computed sequentially. For each block however, if containing several equations, the mixed set of equations containing both discrete (Eq. (3)) and continuous variables (Eq. (2)) needs to be solved either through fixed point iteration over the discrete variables followed by normal solving of remaining variables or through other means such as optimization of the complete equation set in the block.

Jacobians

Jacobians can be defined only for the continuous equation subsets of each block since the discrete variables are discontinuous. Newton-Raphson iteration or other techniques can then be utilized for both linear and nonlinear equation blocks in order to provide a consistent and simple code generation. The Jacobian elements can in the case of a linear block alternatively be made input to a linear solver such as those contained in LAPACK [14]. Whether a block is linear or nonlinear is easily found by analyzing and searching the Jacobian for non-constant elements.

The Jacobians usually increase performance but demand symbolic manipulations and are therefore valuable to relieve the code generation software from. Consider the nonlinear equation system below:

```
iEq[1,1] = {a, sin(a)-userf(b), "res"};
iEq[1,2] = {b, cos(a)-abs(b), "res"};
```

The equations form a block of two equations and the Jacobian becomes :

```
iJac[1] = {{1, cos(a)}, {2, -der userf(b)}};
iJac[2] = {{1, -sin(a)}, {2, -(if b<= 0 then -1 else 1)}};
```

where only non-zero elements are included. In this case, the partial derivative function `der_userf` has been defined by the modeler as below:

```
function userf
  input Real x;
  output Real y;
algorithm
  // . . .
end userf;
der_userf = der(userf,x);
```

When such a partial derivative definition exists, automatic differentiation as in [15] can be utilized in many cases, at least if the derivative function is not defined as external. For external functions, only numerical differentiation remains in the general case.

6.4 Numerical Integration

After the initialization, numerical integration of the derivatives can take place in simulation in order to compute the states. For this purpose the ODE of Eq. (2) is supplied on the form previously described where the equations after the last differential equation are separated (2b) and computed in the output section together with simple equations that have been removed during optimization. In the ball model

case the derivative equations can be extracted from the complete set of equations below

```
Eq[1,1] = {der(h), v, "ass"};
Eq[2,2] = {der(v), if flying then -g else 0, "ass"};
```

```
Jac[1] = {{1, 1}};
Jac[2] = {{1, 1}};
```

6.5 Reinitialization

When an event has occurred, certain when conditions will change and thus a certain new set of when equations will be active. The equation system will therefore change between events. This can be exploited by preparing efficient code for each combination of events. The number of combinations easily becomes large however and that is why the Modelica compiler Dymola [9] performs an offline simulation in order to see which combinations are likely to occur. A general solution works as well but with lower performance and is the one sketched here.

The equation set is created for the case where all when-equations are active. After an event, the effective equation system is gathered from this list using information on which when-equations are active. The states are considered known unless they are part of any active reinit equations in which case the reinit is treated as state assignments. In the example below, the complete equation set is shown together with the Jacobian. Here, the complete set is the same as the one formed when one of the when-conditions becomes true.

```
riEq[1,1] = {impact, h<=0.0, "ass"};
riEq[2,2] = {v_new, if edge(impact) then -e*pre(v) else 0, "inst"};
riEq[3,3] = {v, v_new, "inst"};
riEq[4,4] = {flying, v_new>0, "inst"};
riEq[5,5] = {der(h), v, "ass"};
riEq[6,6] = {der(v), if flying then -g else 0, "ass"};
```

```
riJac[5] = {{5, 1}};
riJac[6] = {{6, 1}};
```

The equation system above has been sorted and partitioned, which is too demanding during simulation. The Jacobian can however be used at all times since it only is defined for the continuous equations and is not affected by causality.

6.6 Event Detection and Location

Any discontinuity in the ODE equation set during simulation can be captured through zero crossing functions that switch sign at the discontinuity. One function is created for each unique relation in *rel*.

These functions are provided to the numerical integrator in order for the integrator to identify the time point of the discontinuity and restart the simulation at that point with new values for the states. The zero crossing functions below are given together with which equations they either are part of (can be both in Eq. (2) and (3)) or can make active by being part of when conditions.

```
z = {0.0-h,{1,2,3,4}}; 0.0-v,{2,3,4}};
```

Zero crossing functions reflect changes in relations that affect boolean variables that are part of when-conditions. To determine the current active set of when-equations it is necessary to know which when-equations a boolean equation triggers. In this case, the `impact` equation triggers the second when-condition and therefore could make equations 2, 3 and 4 active.

```
c = {1,{2,3,4}};
```

6.7 Code Generation

Given the complete Modelica model and naming conventions described above code can be generated for usage in a simulation environment, for example. Since it is Modelica syntax, the Modelica parser can be reused as well as parts of existing Modelica compilers for generating C code for individual Modelica expressions.

By translating the Modelica code to ModelicaXML [16], an XML implementation, a standard XML parser can be used instead together with widely used XML tools for traversing the XML structure and extracting and transforming information.

7 Model Adaptation for Causal External Usage

Imagine the scenario where a Modelica model is to be used within several causal models, for example within the simulation environments EASY5 and Simulink. One example could be a Modelica model of an engine, or even a complete standard Modelica library of driveline components that modelers of different enterprises have the interest of making part of their complete vehicle simulations performed in domain specific simulation tools with no support for equation-based models. A recent case is the CAPSim simulation centre [17] for hybrid vehicle simulation that has a model library open for usage and where the intention is to make the models usable for many simulation environments.

A flexible compiler supports straightforward code generation suited for these different environments, but the models themselves need to be adapted at the equation level as well, as is described in this section.

Consider a mechanical inertia model with outside connectors containing position and force. The connector class is specified in Modelica as shown below. The `flow` keyword tells the compiler to sum corresponding variables in a connection with other connectors. The non-flow variables are set equal. The mentioned inertia model connectors are part of the external interface and it is through those the communication of connector variables to and from other models takes place.

```
connector flange_pos
  Real pos(unit="m") "Pos into";
  flow Real force(unit="N") "Pos out";
end flange_pos;
```

Now consider a model where the outside connectors contain speed and force instead. Connecting the two models could be performed through a model that contains an equation for the relation between speed and position. This could even be automated by identifying differential variables and corresponding derivatives through unit checking.

```
model intermediate
  flange_speed fs;
  flange_pos fp;
equation
  fs.speed=der(fp.pos);
  fs.force=fp.force;
end intermediate;

connector flange_speed
  Real speed(unit="m/s");
  flow Real force(unit="N");
end flange_speed;
```

But in a causal target environment, the connectors are causal. The next example demonstrates a wrapper that assigns causalities to an acausal force source model to make it suitable for insertion into a specific causal target model.

```
model force_source
  flange_pos flange_p;
equation
  flange_p.force = 0.2*cos(time);
end force_source;

model wrapper
  flange_pos flange_p;
  input Real speed;
  output Real force;
equation
  speed=der(flange_p.pos);
  force=flange_p.force;
end wrapper;
```

The causal model is created as below by connecting the wrapper to the force source. A Modelica com-

piler is able to create code for this model and saves information about the connectors in the comments of the input and output variables.

```

model causal
  wrapper w;
  force_source fs;
  input Real speed "($Conn:flange_p)";
  output Real force "($Conn:flange_p)";
equation
  connect (w.flange_p, fs.flange_p);
  w.speed=speed;
  force=w.force;
end causal;

```

7.1 Automatic Adaptation

The content of the wrapper model depends on both source and target model connector variables. New variables may need to be created, as the speed in this case, unit conversions may need to be applied as well as sign changes. All in all, a large number of combinations exist for which wrappers need to be created to support the usage of the model in arbitrary causal models. But the standards for exchanging variables among causal models in a certain simulation environment can be declared in template connectors, where also sign conventions are declared. In the force source example, the connector template could look like

```

model template_connector
  input Real speed(unit="m/s") "Pos out";
  output Real force(unit="N") "Pos into";
end template_connector;

```

From the template connector, a compiler can create the wrapper and causal model automatically as is described from now on. By comparing the connector class `flange_pos` with `template_connector`, it can be seen that two variables have the same units; the forces. The variables `pos` and `speed` have different units, but here it can be seen that speed is the derivative of `pos` by analyzing the units. The wrapper model is now complete, the forces are set equal and the derivative relation is established. If the speed of `template_connector` would have had the unit "mm/s" instead, a factor 1000 would have been needed in the wrapper equation.

There is no sign convention information in the Modelica models apart from comments. This information can be used as a starting point, looking for keywords such as "out of" and "into" and similar. But of course more explicit information would be valuable. In this case the sign conventions differ and the wrapper equations will both contain negations.

Here follows yet another template connector that could be used instead of the one defined earlier. The resulting causal model is compatible with the use of so called bilateral delay lines [18], which provide

means for robust coupled simulation in which every component model solves its own equations and communicates at discrete time points with surrounding models. The variable `C` is in this case a force wave that passes back and forth in the delay line, as is the case in metal rods, for example. The end result is an overall numerical solution that mimics the time delay existing in physical interactions in nature. The variable `Z` is a kind of impedance. Without going further into the details of delay lines, the example shows how new equations can be automatically added to existing models in a mathematically sound way in order to adjust the models towards various boundary conditions.

```

model template_connector
  input Real C(unit="N");
  input Real Z(unit="N.s/m");
  output Real speed(unit="m/s");
  output Real force(unit="N");
equation
  force = C + Z*speed;
end template_connector;

```

7.2 Causality Combinations

It could be that the force source model cannot be solved with the specified input-output causality in the target connector. Since in causal environments, every second model has inverted causalities for compatibility in variable exchange, this can be tested for the target connector to see if that works better.

The force source model can be solved with speed as input and force as output, but not the other way around. The test can be performed through maximum matching algorithms [12] present in standard Modelica compilers.

What has not been mentioned so far is how to find a matching target connector for a certain outside connector of the Modelica model. For a match to exist there must be a one to one match between an outside connector variable and a causal target connector variable in terms of units. The units may be of differentiated/integrated form in the target connector. The target connector may have more variables if they are output variables and are part of supplied equations.

If the model has a number of connectors of different classes, such as both mechanical, electrical and other types, the testing becomes tedious and the testing can be handed over to the compiler to either generate all solvable combinations of inverted and non-inverted target connector causalities or generate the "best" solution. One measure for how good a solution is to see how high index the adapted model has. One example is feeding a mass with a position rather than

force and thus removing its inertia effect. The user interface could be implemented such that there is a choice to enforce the default target boundary conditions or to give priority to low index.

The model adaptation method where low index is prioritized described in this section has been implemented and tested in a Modelica compiler [19] supporting only continuous equations without arrays. The model adaptation takes place after flattening where the outside connectors of the flattened model have not been flattened so that they can still be identified. The algorithms have been proven useful since the compiler automatically produces numerically sound code for a given set of target connector templates and a source Modelica model.

8 Conclusions

A partial implementation of the Modelica intermediate code format has been done in the OpenModelica compiler to be used as input to external code generators. The intent is that this information can be produced and is complete for generating simulation code from hybrid Modelica models.

The model adaptation functionality for using Modelica models in an external causal context needs to be implemented in a compiler supporting the complete Modelica language, e.g. soon OpenModelica. This can pose problems, not at least due to the increasingly complex definition of connectors in Modelica. Nonetheless the model adaptation can be made less automatic if needed, for example by defining explicit transformations among different connector definitions.

References

- [1] Modelica Association (2005): **Modelica – A Unified Object-Oriented Language for Physical Systems Modeling – Language Specification**. Version 2.2
- [2] P Fritzson, et al (2002): **The Open Source Modelica Project**. In Proceedings of The 2nd International Modelica Conference, 18-19 March, 2002, Munich, Germany. See www.ida.liu.se/projects/OpenModelica
- [3] P. Fritzson, A. Pop and P. Aronsson (2005): **Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica**. Proceedings of the 4th International Modelica Conference, Hamburg, pp. 519-525
- [4] M. Otter and H. Elmqvist (1995): **The DSblock model interface for exchanging model components**. In Proceedings of the 1995 EUROSIM Conference, Elsevier Science Publishers, pp. 505-510
- [5] P. J. Mosterman and J. E. Ciolfi (2002): **Embedded Code Generation for Efficient Reinitialization**. Proceedings of 2002 IFAC 15th Triennial World Congress, Barcelona, Spain.
- [6] A. Pinto, L. P. Carloni, R. Passerone and A. Sangiovanni-Vincentelli (2006): **Interchange Semantics for Hybrid System Models**. Proceedings 5th MATHMOD Vienna
- [7] C. Pantelides (1988). **The Consistent Initialization of Differential Algebraic Systems**. SIAM Journal on Scientific and Statistical Computing, Vol. 9
- [8] P. Fritzson (2004): **Principles of Object-Oriented Modeling and Simulation with Modelica 2.1**. Wiley-IEEE Press, ISBN 0-471-47163-1
- [9] S.E. Mattsson, M. Otter, M., and H. Elmqvist (1999): **Modelica Hybrid Modeling and Efficient Simulation**, the 38th IEEE Conference on Decision and Control, CDC'99, Phoenix, Arizona, USA
- [10] T. Park, P.I. Barton (1996): **State Event Location in Differential-Algebraic Models**. ACM Transactions on Modeling and Computer Simulation, Vol. 6, No. 2, pp. 137-165
- [11] P. I. Barton and C. K. Lee (2002): **Modeling, Simulation, Sensitivity Analysis and Optimization of Hybrid Systems**. ACM Transactions on Modeling and Computer Simulation, Vol. 12, No. 4, pp. 256-289
- [12] I. S. Duff and J. K. Reid (1981): **Algorithm 575 Permutations for a zero-free diagonal**. ACM Transactions on Mathematical Software, Vol. 7, pp. 387-390
- [13] I. S. Duff and J. K. Reid (1978) **An implementation of Tarjan's algorithm for the block triangularization of a matrix**. ACM Transactions on Mathematical Software, Vol. 4, pp. 137-147
- [14] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen (1995): **LAPACK Users' Guide**. SIAM Philadelphia, Pennsylvania, 2 edition
- [15] H. Olsson, H. Tummescheit, and H. Elmqvist (2005): **Using Automatic Differentiation for Partial Derivatives of Functions in Modelica**. Proceedings of the 4th International Modelica Conference, Hamburg, pp. 105-112
- [16] A. Pop and P. Fritzson (2003): **ModelicaXML: A Modelica XML Representation with Applications**, Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, pp. 419-430
- [17] J. Fredriksson, J. Larsson, J. Sjöberg and P. Krus (2006): **Evaluating Hybrid Electric and Fuel**

- Cell Vehicles using the CAPSim Simulation Environment.** The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition, October 23-28
- [18] D. M. Auslander D.M (1968): **Distributed System Simulation with Bilateral Delay-Line Models.** Journal of Basic Engineering, Transactions of ASME, pp. 195-200
- [19] J. Larsson (2007): **A Framework for Implementation-Independent Simulation Models.** Accepted for publication in Simulation: Transactions of the Society for Modeling and Simulation International

Dymola interface to Java - A Case Study: Distributed Simulations

José Díaz López

Hans Olsson

Dynasim AB

Research Park Ideon, S-223 70 Lund, Sweden

{jose.diaz, hans.olsson}@dynasim.com

Abstract

Running multiple simulations for the same model is useful for studying the influence of parameters, calibrating parameters from measurement series, and optimizing designs to be robust with respect to operating conditions. Dymola's [1] Design-package allows a user to specify such experiments for experimenting, calibrating and optimizing simulation studies [2].

Many of the simulations are independent allowing them to be run in parallel, i.e. coarse-grained parallelism. This paper describes the extension that distributes the simulations to multiple CPUs, while keeping the original setup for the simulation study. This makes the distributed nature transparent for the user, and the only difference is that the studies are done faster.

The implementation is implemented in Java using a Modelica external interface, but in contrast to [3] with direct interface inside Dymola. Implementing the distributed simulations in Java makes it possible to leverage the multi-threading, graphical, and communication capabilities of Java; while using Modelica's strengths for modeling, setup of simulations, and numeric algorithms.

Keywords: dynamic simulation, distributed simulations, Java, Modelica

1 Introduction

Dymola broadens the external interface possibilities to Java.¹ A case study is distributed simulations for model experimentation.

¹ Java and any other Sun trademarks that are referred to or displayed in the document are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

Dymola is a trademark of Dynasim AB, and a registered trademark of Dynasim AB in Sweden.

There are several needs for interfaces between Dymola/Modelica and other tools/languages that can be met in different ways. In this case we need to call external functions inside Dymola and in those functions have access to Dymola's API in a "safe" way.

Java with Java's Native Interface allows this in a natural fashion, and Dymola allows a user to directly call external functions written in Java from the scripting environment. In those functions written in Java it is also possible to access all of Dymola's API functions and normal Modelica functions.

This tight coupling requires that Dymola internally runs a Java Virtual Machine, where Dymola adds implementations of native functions to allow access to Dymola's API.

2 Dymola Interface to Java

2.1 Comparison with previous work

Previous implementation of interfaces to Java (based on top of the external C-interface) did not allow functions written in Java to call Modelica functions as seamlessly, but sufficed for calling functions written in Java inside models. The new implementation also allows this and ensures consistency since one external declaration in Modelica can be used for calling the function both in the scripting environment and inside models.

This implies that models using functions written in Java will include a special header and automatically link with the Java Virtual Machine.

Previous implementation of access to Dymola's API functions e.g. via DDE [3] only allows the external tool to control Dymola, but not Dymola to call the external tool.

Modelica is a registered trademark of the Modelica Association.

The new implementation allows calls from Modelica functions to Java, *and* allows this function in Java to call Dymola/Modelica.

2.2 Mapping of functions

The interface in Modelica to functions written in Java is limited to static member functions of classes. The declarations in Modelica are of the kind:

```
function foo
  input Real in1;
  input String in2;
  output Boolean out;
external "Java" out='P.C.S'(in1,in2);
end foo;
```

In this example P is a package, C a class and S a static function in this class.

This is according to the Modelica specification (including the use of quoted identifiers for the name of the function) except that Java is not one of allowed external language in the specification. The package-name can be a hierarchical name, with dot-notation.

For the future it might be possible to extend Modelica's external objects to also handle objects in Java.

Calls of Modelica functions (and Dymola's API-functions callable as Modelica functions) from Java go through one generic function accessible in Java as `com.dynasim.dymola.interpretMainStatic`. For other Modelica functions a wrapper in Java can be constructed in a mechanical way that maps arguments, calls this bridge function, and maps the result.

Having one entry point to Modelica from Java makes it straightforward to transparently redirect all calls to a remote instance of Dymola, i.e. remote method invocation.

This interface is only intended for accessing one instance of Dymola. Distributed simulations require access to multiple instances of Dymola, and thus require additional efforts as will be explained later.

Asserts and other errors in Modelica are mapped to exceptions to in Java and vice versa. Specific exceptions are thrown for errors specific to calling Modelica functions from Java (illegal types for arguments, unknown Modelica function, etc).

2.3 Mapping of data-structures

Simple types in Modelica (e.g. Real) are normally mapped to corresponding simple types in Java. Strings are non-simple in Java, the mapping is still

direct, and is made simpler by the fact that JVM and Dymola internally use the same UCS-8 implementation of Unicode strings.

Note: The UCS-8 mapping is the result of applying the UTF-8 mapping to UTF-16 strings, and the recommendation is that even though it can be used internally in programs it should not be used for interfaces. In this case we make an exception in order to be compatible with the pre-existing C-interface of JVM.

Arrays in Modelica correspond to (possibly nested) arrays in Java. Special care is needed to detect heterogeneous arrays in Java, and convert zero-sized matrices from Java.

Records in Modelica are mapped to a class implementing a map interface in Java. This ensures that the semantics of Modelica records (named based type equivalence) is preserved.

To summarize we first present how arguments are mapped when calling a function written in Java from Modelica.

Modelica	External Java
Real	double
Integer	int
Boolean	boolean
String	java.lang.String
Record	com.dynasim.record
Real[]	double[]
Integer[]	int[]
Boolean	boolean[]
String[]	java.lang.String[]
Record[]	com.dynasim.record[]

The mapping when a function in Java calls `interpretMainStatic` is similar, but has special handling of simple types as presented below. This is necessary since the simple types such as double are not objects and thus cannot be part of the generic argument list of `interpretMainStatic`.

Modelica	interpretMainStatic
Real	java.lang.Double
Integer	java.lang.Integer
Boolean	java.lang.Boolean
String	java.lang.String
Record	com.dynasim.record
Real[]	double[]
Integer[]	int[]
Boolean	boolean[]
String[]	java.lang.String[]

Record[]	com.dynasim.record[]
----------	----------------------

The record class, `com.dynasim.record` implements the map-interface, and the contents is mapped as for `interpretMainStatic` (also when calling a function written in Java from Modelica). The reason is the same as for `interpretMainStatic`. However, for easy access to simple variables there are also special functions, `getDouble`, `getInt`, and `getBoolean`.

The map for records is straightforward to use and by being name-based avoid issues with declaration order and future extensions of the records in Modelica.

2.4 Mapping of errors

Exceptions thrown from Java called from Modelica are automatically mapped to assertions, which is the normal error handling primitive in Modelica. Currently an assertion stop Dymola's interpreter are there is no way of catching the error inside Modelica.

When an assertion (or other error) is triggered in Modelica originating from a call to `interpretMainStatic` this is mapped to an exception in Java as follows:

- `com.dynasim.DymolaException` base-class of the other exception – introduced in order to make it easy to catch all exceptions.
- `com.dynasim.DymolaNoSuchFunction(<name of function>)` when the function is not found by `interpretMainStatic`.
- `com.dynasim.DymolaIllegalArgumentException` for problems with transforming results or argument between Java and Dymola, and incorrect type of arguments to function.
- `com.dynasim.DymolaEvaluationException` when evaluation fails – e.g. assertions and division by zero.

These exception classes all inherit from `java.lang.RuntimeException`, this ensures that no 'throws' clause is needed for routines calling Dymola-functions.

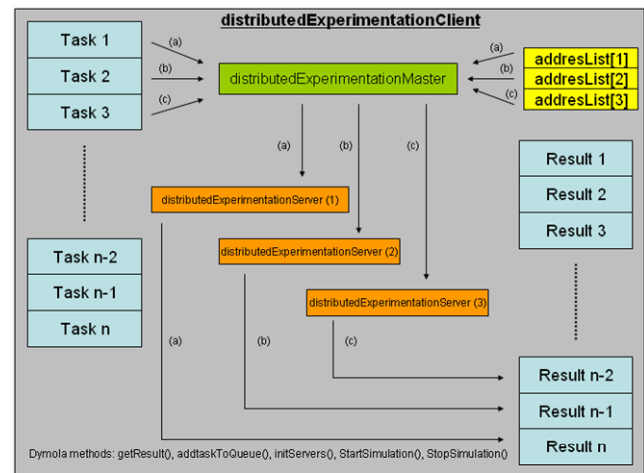
This corresponds to the Modelica environment where no "assert-clause" is needed.

3 Distributed simulations with Java

The new facilities accessible from Java for calling a Dymola instance, allows the use of transparent RMI [4] for distributed tasks. Since Dymola simulations for parameter experimentation are independent, long simulation times can be shortened by using several processors. A version of the Experimentation package [5] has been adapted to distributed simulations.

3.1 Setup

The command setup is of distributed Experimentation package is identical to the one described in [5].



The main architecture implemented in Java is depicted in the previous figure.

The architecture of the Java code is simple:

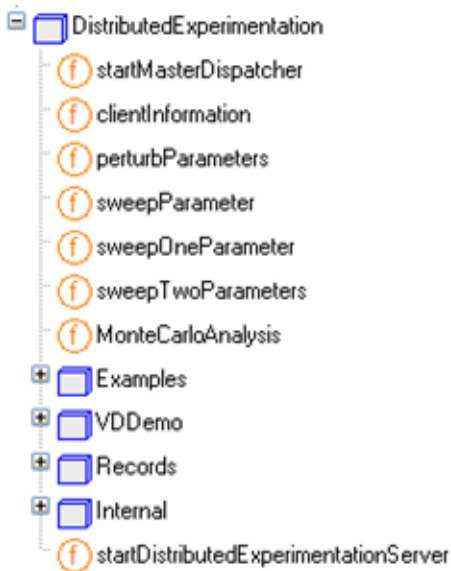
- A master dispatcher with three lists working as queues: Task Queue, ResultQueue and AddressList
- Independent threads for each server. The thread starts, monitors and reports results to the static lists in the dispatcher.
- External Java functions in Modelica to access the different Queues.

Single signals of whole trajectory files can be sent back to the master computer if requested, for animation purposes.

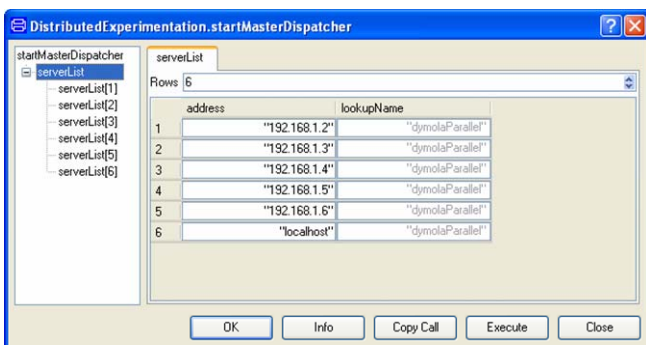
Having all this functionalities at hand, the Distributed Experimentation package was written, and we describe it in the following.

3.2 Distributed Experimentation package

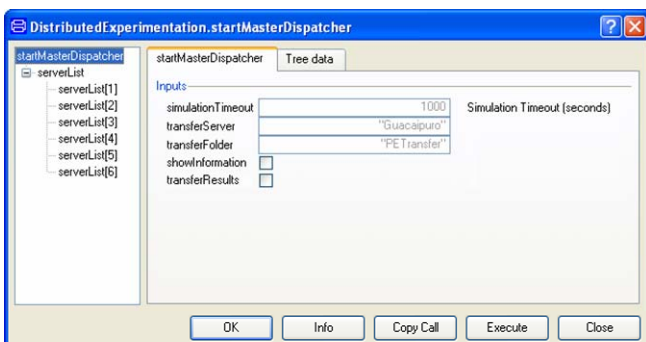
The structure of the Distributed Experimentation package is very similar to the Experimentation package. The functionalities are the same, as shown below.



The main difference to the experimentation package is the distribution of tasks from client to server. The GUI and setup are the same, and the function `startMasterDispatcher` has to be run before any simulation is performed. The information of the servers and their lookup names is reported to the dispatcher with this function. See the figure below.



The names “192.168.1.2”, “192.168.1.3” and so on are IP numbers of the server hosts. DNS name resolving is also supported, and therefore possible to use names instead. The name “localhost” represents the address 127.0.0.1 as usual. Use it to include the own computer which also runs a server.



In the view above it is included the set up for trajectory file transferring. Check “transferResults” if transfer files to the dispatcher computer is to be done. The variable “transferServer” is the network name of the client computer and the “transferFolder” is the **network name** of the shared folder.

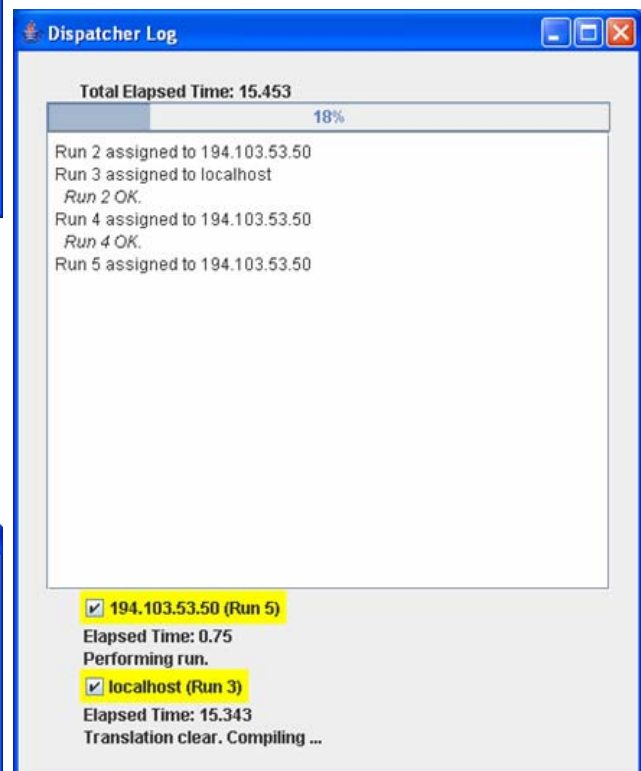
The flag `showInformation` makes Dymola show a window with the registered servers for debugging purposes.

The functionalities of `perturbParameter`, `sweepParameter`, `sweepOneParameter`, `SweepTwoParameters` and `MonteCarloAnalysis` are described in more detail in Dymola Additions document, and can be used directly here. We will focus here on the setup and running of the examples. We consider coupled-Clutches as a reference case study.

The subpackage `VDDemo` has two functions: `vehicleSweepParameter` and `animateResultFile`, used for demonstration purposes.

3.2.1 Progress Monitoring

The task dealer presents in this window which task was sent to which server, as well as the overall progress. The progress bar indicates the percentage of work done and received by the client.

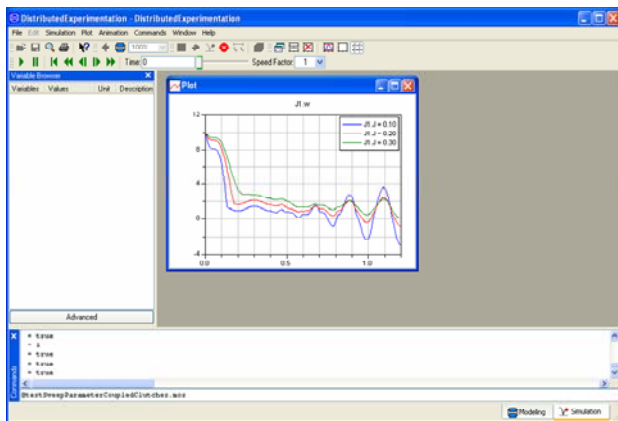


If any task could not be performed by a server, it will be back to the task queue and the respective server will be disabled (marked with red background). Yel-

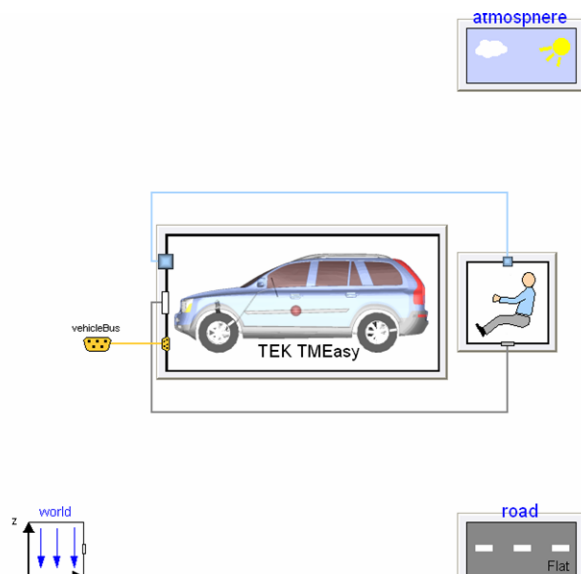
low background denotes ongoing simulation on server. Green background denotes free server.

3.2.2 Collecting results

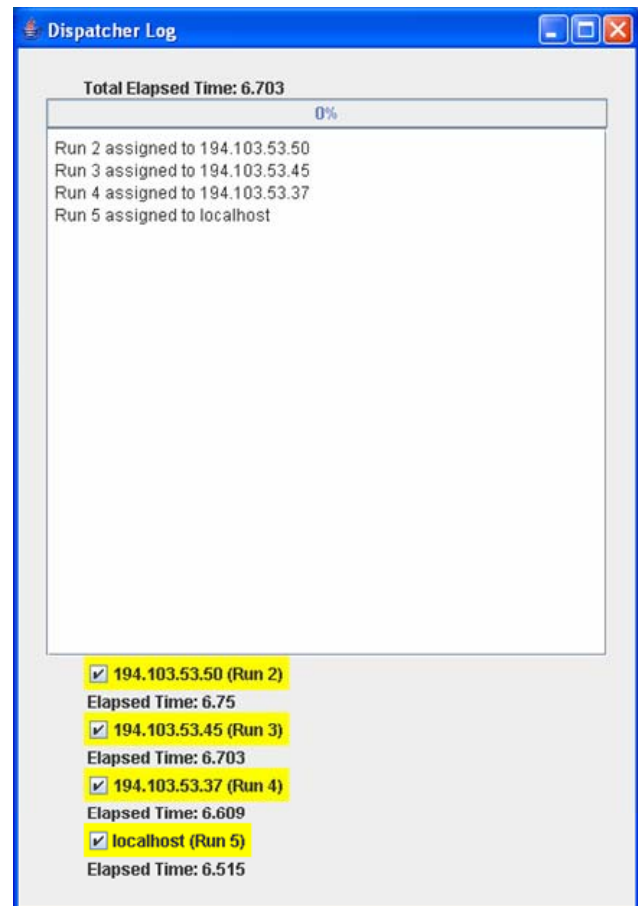
During simulation time, Dymola inspects the result queue in the master dispatcher waiting for results with a simple Modelica Function call. For instance, results are presented directly in sweepParameter as the results are arriving. It is up to the Modelica function whether to present the incoming results or not. In the figure below, the results of sweepParameter on CoupledClutches example are depicted, after three results are arrived.



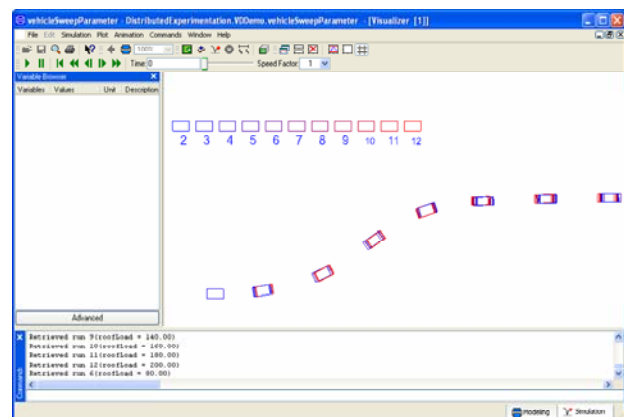
Running a sweep with Payloads-example can be done by clicking on “Commands/Load (11 cases) with weights from 0 to 200 Kg”, and then execute. The model is depicted below.



For this example we used four servers. The dispatcher log evolves as depicted in the figure below.



Dymola shows the following view when all 11 cases are completed. The visualisation is performed by a special visualizer written for this example. In the figure below, we observe a rectangle defined by the position of the four wheels and the identifier of the case.



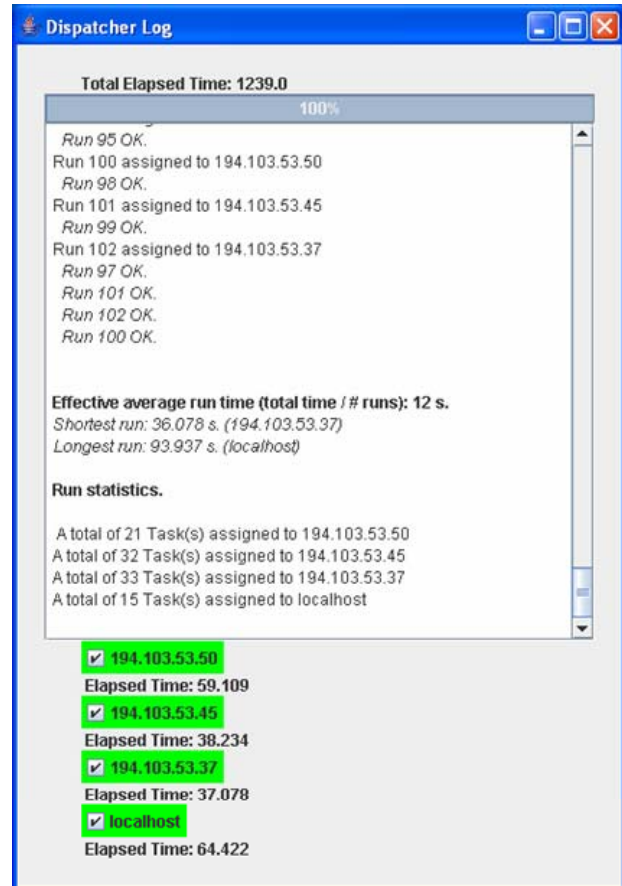
We observe with his experiment that the car is stable with regard to a payload from 0 to 200 Kg.

The final log window has the following information and statistics. The first simulation was ready after 86.5 seconds (translation, compilation with Visual Studio 2003 and running). We analyse now two runs of sweepParameter.



To interpret the results, we consider the total time against the best time. Using the best computer, the total time for eleven simulations would be 411 seconds. As a result of the distributed simulation, the whole task needed only 202.5 seconds. This means that the speedup factor was of 2,036. We observe also that the localhost and the first server are slowing down the simulation.

For a second run, we make the same sweep but for one hundred simulations. In this way, we get rid of initialisation overhead. Observe the final dispatcher log.



We analyse this log in the next section.

3.3 Asymptotic resulting speed up

Confirmation of theoretical Speed up factor (approx. number of servers) has been observed using long simulation times using VehicleDynamics 1.0.2.

The effective time used per simulation for the second sweep is of 12 seconds. The fastest simulation was 36.07 seconds. If we just only used the fastest computer, the total elapsed time would have been 3607 seconds. The total elapsed time with these two servers was 1239 seconds. The resulting speedup factor is 2.911 compared to the most optimistic scenario.

It is our experience that localhost behaves slower than its equivalent server, since the dispatcher runs also in the same computer. We added a computer double so slow than the other two. Equivalently, we were running with three equally fast servers in total, reflected in the example.

4 Conclusions

A new interface between Modelica and Java has been implemented. This enables Modelica and Dymola to take advantage of all Java features, making possible the incorporation of java objects in Modelica.

As such an example of Java applications with Dymola, distributed simulations with Dymola were implemented using the transparent RMI Java package. High speed up factors were observed with low overhead. The main advantage is that RMI handles directly all concerning network communication, while Dymola handled all simulation aspects.

References

- [1] Dymola User's Manual, www.dynasim.com
- [2] Elmqvist H., Olsson H., Mattsson S.E., Brück D., Schweiger C., Joos D., Otter M., Optimization for Design and Parameter Estimation, Proceedings of the 4th International Modelica Conference, Hamburg-Haburg, Germany, 2005.
- [3] Olsson H., External Interface to Modelica in Dymola, Proceedings of the 4th International Modelica Conference, Hamburg-Haburg, Germany, 2005.
- [4] Gur-Ari, G. Empower RMI with TRMI. <http://www.javaworld.com/javaworld/jw-08-2002/jw-0809-trmi.html>
- [5] Dymola Users Manual – Dymola 6.0 Additions, shipped together with Dymola distribution.

Simulation of complex systems using Modelica and tool coupling

Roland Kossel, Wilhelm Tegethoff, Michael Bodmann, Nicholas Lemke

TLK-Thermo GmbH
Hans-Sommer-Straße 5, 38106 Braunschweig, Germany
r.kossel@tlk-thermo.de

Abstract

Basically there are two approaches of modeling and simulating complex systems composed of subsystems. Approach A is based on established simulation tools of the respective domain in order to conduct a co-simulation by tool-coupling. Approach B consists of modeling the whole system in only one suitable language like Modelica. Using approach A the existing software and the in-house and external expert know-how can be applied. Approach B has significant advantages because of the description of the complete system in one standardized and open language like Modelica and employing an efficient simulation environment such as Dymola. When simulation environments which support approach B are coupled with existing software based on approach A, the advantages of both approaches can be utilized. Considering for example the simulation of a passenger compartment, the feasibility of a coupling of Modelica/Dymola, Flowmaster2, Simulink and THESEUS-FE as well as different utility programs is demonstrated in this presentation.

Keywords: co-simulation; tool coupling; middleware; TISC

1 Simulation of thermal systems in an automotive environment

When simulating a vehicle, its thermodynamic subsystems like engine, engine-cooling system, oil cooling, air conditioning cycle, HVAC unit and passenger compartment are connected with each other especially by fluid- and heat-flows. In order to describe the thermodynamic behavior of the complete system, the models of the subsystems have to be coupled using the variables heat flow rate, mass flow rate, temperature and pressure and if necessary other, non-thermal variables like motor rotation speed. It makes

sense to choose the best suited simulation tool for each subsystem. The following list gives a fragmentary overview of tools that may be used for the respective problems:

Air conditioning cycle	Modelica-Dymola
HVAC unit	CFD / Simulink
Passenger compartment	CFD / THESEUS-FE
Cooling cycle	Flowmaster2 / KULI
Oil cooling	Flowmaster2 / KULI

In the literature, several different approaches are presented for describing the complete system. The approaches range from simple couplings of simulation tools over a co-simulation with different simulation programs to modeling the whole system in one software environment [1], [2].

Figure 1 shows the typical current state of simulation tool coupling in the area of automotive thermal systems. For transferring CFD-information to a 1d-simulation program files are used in most cases [3]. In case of a bidirectional coupling this approach has limitations especially concerning synchronization. When only two programs are coupled, different techniques for tool integration like DLLs or communication channels through COM or CORBA can be employed. As Figure 1 shows, using pair-oriented software coupling quickly meets its limitations. The number of pair-oriented interfaces increases greatly with every integrated tool. Without a standardization and control of the tool-coupling, this technique will be difficult to use. An appropriate synchronization seems to be hard to introduce. Additionally, the implementation and maintenance of the numerous interfaces involves high costs (see [4]).

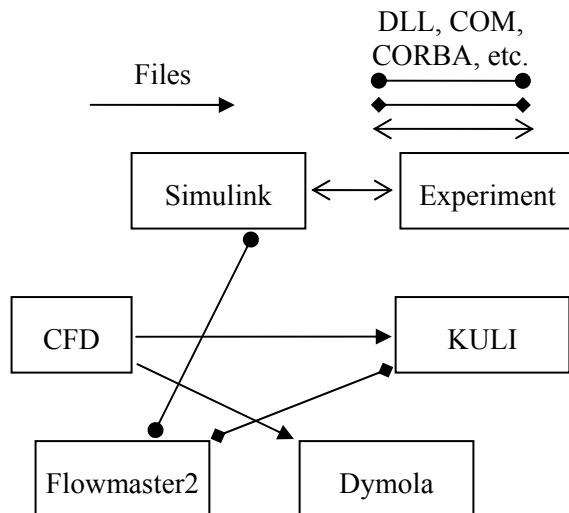


Figure 1: Exemplary current state of tool coupling of simulation programs and HIL-applications

It follows from the above, that the coupling technique for simulation programs must be standardized. The following chapter describes a possible way for this. Afterwards an alternative way describing the thermal behavior of a vehicle holistically is presented.

2 Coupling of simulation tools

As pointed out in Chapter 1, special coupling software (middleware) should be employed when simultaneously coupling more than two programs (see Figure 2). The middleware provides the systematic coupling of the different simulation programs and takes care of the synchronized data communication. In difference to the pair-oriented technique from Chapter 1, only one interface is needed for every program when using a middleware concept.

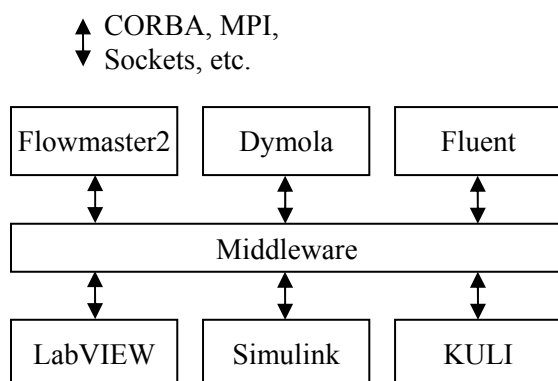


Figure 2: Coupling of simulation tools and data acquisition software through a middleware

A typical middleware is for example the program EXITE [5], which is widely used in the field of vehi-

cle electronics. EXITE's data exchange is based on CORBA for normal and MPI for HIL applications. In the field of thermal management Puntigam [1] describes the middleware of the VIF in Graz. This software relies on CORBA and has numerical means for coupling the differential equations of the coupled programs. The MPI based middleware MpCCI [6] provides a solution for coupling 3D programs. MpCCI aims at mesh-mesh-coupling. At the present date MpCCI support only the coupling of two applications at one time.

The configuration displayed in Figure 2 is currently not realizable with any of the described middleware. Due to the need of a short-term availability and special demands concerning the data synchronization and platform independent data exchange, the middleware TISC was developed.

TISC allows platform independent exchange of commands (strings), floating point numbers (double) and integers as scalar values as well as vectors and matrices. The data exchange is realized through sockets. As Figure 3 shows, TISC consists of a server and the respective clients, which are integrated into the coupled programs. For this integration, interfaces are available in C, C++, C# and FORTRAN. A connection over COM is also supported and used for example for Flowmaster2 and KULI.

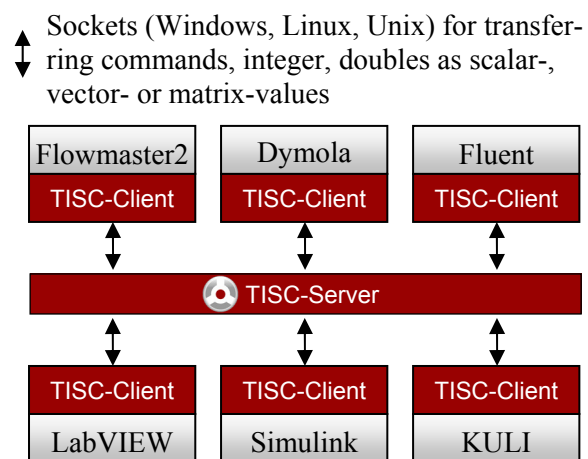


Figure 3: Coupling of simulation tools and data acquisition software with TISC

TISC-integrations are currently available for the following programs:

- Flowmaster2
- Fluent
- KULI
- LabVIEW
- Matlab/Simulink
- Modelica/Dymola
- StarCD
- THESEUS-FE
- Trnsys

The following interfaces are currently under development:

- Ansys
- Excel
- CFX
- COM

Furthermore utility programs exist for visualizing the exchanged data. TIM – the TISC Information Monitor – receives and saves all data sent by the coupled programs. The StateViewer displays a pressure-enthalpy-diagram at run time of the co-simulation or the measurement. A special client for controlling the next time increment (if the coupled programs allow this) and for coupling the variables numerically efficient is in development (see [1]).

2.1 Advantages of coupling simulation tools

When coupling simulation tools with a middleware, several advantages arise besides from the saving of time. Employing a middleware for the simulation of the complete system allows the utilization of present expert knowledge which is bound to existing software. Using this knowledge can speed up integrating the examination of the complete vehicle into the product development process. Beyond this, the best tool can be chosen for describing each subsystem. By using a standardized definition of the interface variables, the single simulation tools can be easily exchanged (compare Figure 4). Even if the approach of using a unified language for building the models is used, it can make sense to couple sub models with a middleware to decouple the time Steps of the solver.

2.2 Arising problems

Besides the mentioned advantages, there are also drawbacks to consider. Since events have to be triggered to exchange the data between the simulation tools, the solvers of those programs have to be stopped and restarted at every exchange time. Since the received variables show a discrete behavior, this restart can become difficult. Further on numerical oscillations can occur, when the interdependence of the coupled variables is very high. To reduce these effects, interpolation methods are currently explored in order to convert the discrete behavior of the received variables into a continuous one or to achieve a pseudo multistep integration method.

3 Unified simulation with Modelica

An alternative way to the coupling of simulation tools is describing the complete system including all

subsystems in one unified programming language and mathematics. In the following it is assumed that the subsystems and therefore the complete system can be described by hybrid systems of differential equation (ADE-systems). A hybrid ADE-systems consists of the following three equation types:

1. Ordinary differential equations (ODE) with the respective differential variables whose derivatives explicitly appear in the system of equations.
2. Linear and non-linear ordinary algebraic equations (OAE) with the respective algebraic variables.
3. Event equations with the respective discrete variables whose values can only be changed at an event. The corresponding simulation technique is highly developed in digital electronics. Because of the combination with the continuous ADE-system it is referred to as hybrid ADE-system.

By assembling the sub models to the complete thermal system, a big hybrid ADE-system is created which can be solved with appropriate numerical techniques. Methods for coupling partial differential equations like FEM or finite volume methods form a separate technique. From the authors point of view this technique can not be integrated into the ADE-systems by a standardized description with a unified programming language at this time. Coupling using co-simulation seems to make sense in this case.

The thermal models for engine, engine cooling cycle, oil cooling, air conditioning cycle, HVAC unit and passenger compartment can be described using hybrid ADE-systems. Highly developed computer languages have been developed for this purpose. Besides VHDL-AMS [7] which is mainly used in the field of electronics, the language Modelica [8, 9] is currently widely propagated. The language Modelica is developed by the non-profit Modelica Association with the goal to describe hybrid ADE-systems from diverse physical and engineering domains. Already numerous commercial and non-commercial libraries are available. For example the AirConditioning library [10] is being used by the A.K.2.4.3 of the German OEMs with participation of Audi, BMW, DaimlerChrysler and Volkswagen for unified steady state and transient simulation of air conditioning cycles. In this context the OEMs require the suppliers previous to the project assignment to build the respective component models in the language Modelica [11].

The physical equations are set up in the component models. The transformation of the equations as well

as the numerical solution is left to the used working environment. With means of object-oriented modeling, component libraries can easily be built up. Defining standard interfaces for the components allow a simple model exchange.

At the moment, Dymola is the most powerful simulation environment which supports the language Modelica [12]. Other simulators like Mosilab [13], Amesim [14] and SimulationX [15] are being extended to also support Modelica. The free working environment OpenModelica can be employed for a subset of simulation problems [16].

The open and unified programming language allows an optimal communication and standardization between departments and suppliers. By using the object-oriented techniques of Modelica, the work in teams is eased. Extending and reusing existing models of the component library is also eased. A substantial training of the employees is necessary, because otherwise the object-oriented design can easily cause the opposite effect. Because of the open language the sustained realization of the own expert knowledge is possible with less dependence of software companies and engineering service providers. By employing the right numerical and object-oriented techniques, shorted times for method development and calculation can be achieved. Furthermore the goal of a working environment providing the pre-processing, simulation and post-processing of the complete system can be reached more quickly.

In order to profit from the mentioned advantages and the benefits of a co-simulation – and to be capable of working without a complete Modelica model library – it makes sense to combine a co-simulation with a unified programming language.

4 Example: Passenger compartment

The tool-coupling using TISC is demonstrated for a simulation of the cool down of a passenger compartment. Figure 4 shows the programs participating in the co-simulation. The central element of the co-simulation is the TISC-server to which every program connects via its interface, the so called TISC-client.

The passenger compartment is being simulated with THESEUS-FE [17]. The model has been provided by P+Z Engineering and describes a British sedan including the driver.

The compartment is being cooled by an R134a air conditioning cycle built in Modelica using the AirConditioning library.

A PID-controller assembled in Matlab/Simulink tunes the temperature within the compartment.

Using TLK's utility program StateViewer, the ph-diagram for the air conditioning cycle can be displayed while simulating. The TISC Information Monitor – short TIM – logs all data exchanged between the different programs over TISC.

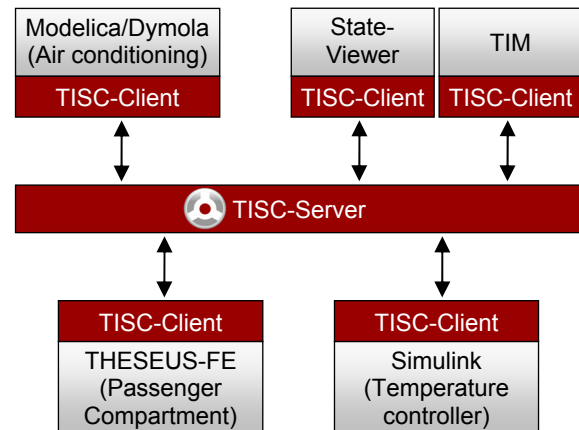


Figure 4: TISC-setup for co-simulation example

As Figure 4 shows, the single programs are not connected among each other but only with the TISC-server. During initialization the simulation tools register the variables they are sending and receiving at the server. The data exchange and synchronization during the simulation is handled by the server. By using this communication setup, an exchange of the simulation tools can be easily arranged – as long as the exchanged tools send and receive the same variables, the other tools' configurations do not have to be altered. For example, the model of the passenger compartment could be changed this way to describe a different vehicle.

Figure 5 lists the variables sent and received by the simulations tools for the co-simulation.

	Sends	Receives
Dymola	Output evaporator (T, mdot)	Input evaporator (T, mdot)
		Relative displacement
Simulink	Relative displacement	Signal of temperature sensor
THESEUS-FE	Input evaporator (T, mdot)	Output evaporator (T, mdot)
	Signal of temperature sensor	

Figure 5: Variables exchanged in co-simulation

The Modelica-model describing the air conditioning cycle is shown in Figure 6. The TISC-interface is integrated as a single block into the example “OrificeTubeCycle” of the AirConditioning library.

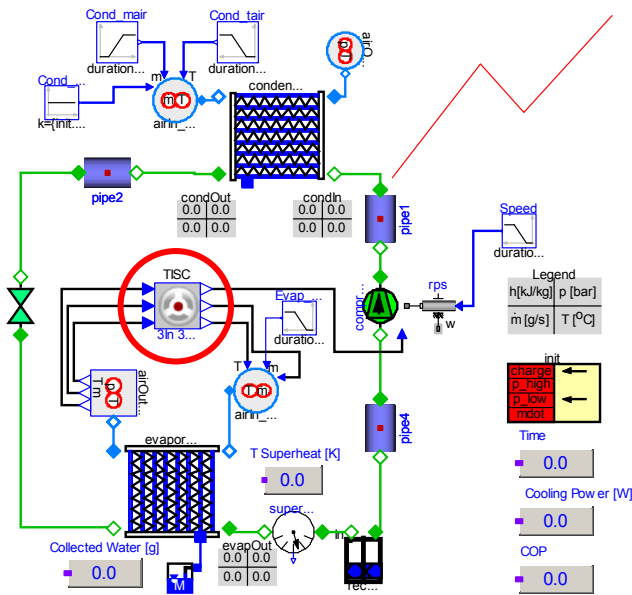


Figure 6: Modelica model in Dymola for co-simulation example

The TISC-block is a composition of single blocks for the different functions of the tool coupling. These are

- Creating an event when data needs to be exchanged
- Managing the connection to the TISC-server
- Sending variables
- Synchronizing with the TISC-server
- Receiving variables

Figure 7 shows the internal composition of the TISC-block from Figure 6.

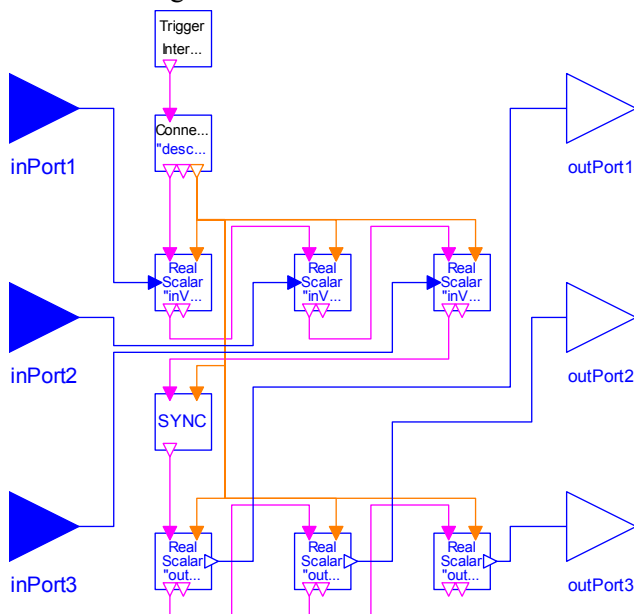


Figure 7: Internal composition of TISC-block

The block sends three variables to the TISC-server and receives three variables from it. Every block in

the assembly stands for one function. A trigger token is passed from one block to the next one to assure the correct execution sequence of the blocks. The token is created in the “Trigger”-block every time the variables need to be exchanged. The sequence shown in Figure 7 is:

1. Create Trigger-token
2. Manage connection to TISC-server
3. Send three variables
4. Synchronize
5. Receive three variables

Every block sending or receiving a variable allows the user to specify an offset and a factor. By doing so, unit conversions between the coupled programs can easily be conducted.

Figure 8 shows the passenger compartment at the beginning of the cool down co-simulation. The initial conditions are calculated with a heat soak simulation with one hour of solar irradiation at 1000 W/m^2 .

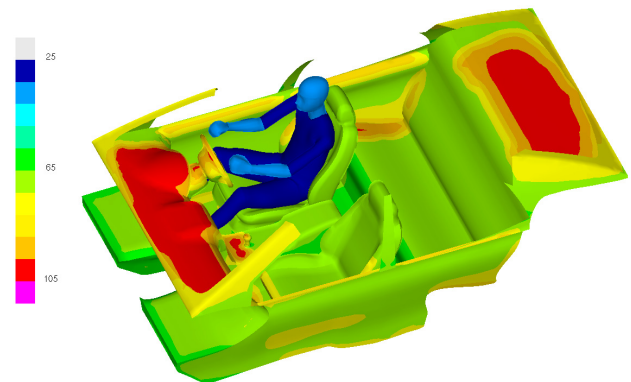


Figure 8: THESEUS-FE result at simulation start

The temperature distribution in the passenger compartment after the coupled cool-down simulation of 2400 s duration is shown in Figure 9.

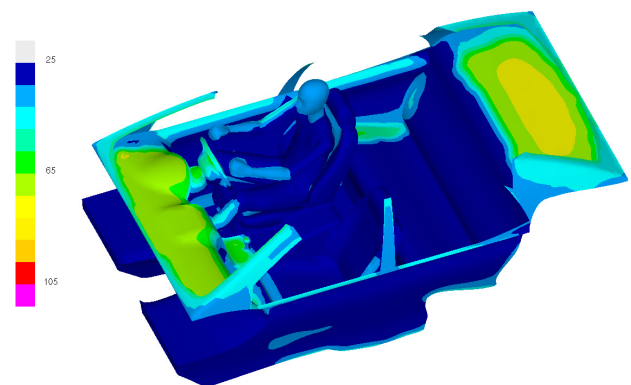


Figure 9: THESEUS-FE result at simulation end

5 Conclusion

When simulation tools supporting the approach of a unified programming language are coupled with existing software using a middleware, the advantages of both approaches can be utilized. Using co-simulation, established programs and the respective knowledge can still be used and combined with standardized code replacing non-satisfying software or describing new functionality.

In the process of moving to a unified language, co-simulation can make sense as the first steps. In the presented example, the temperature controller modelled in Simulink can be replaced by one modelled in Modelica. The next step of integrating the controller in the air conditioning system is not far. Only the co-simulation of models with extremely different 3d approaches like the passenger compartment using the finite element method in THESEUS-FE or the 0d and 1d simulation of the air conditioning cycle in Modelica can hardly be replaced. Even if all models are described in one language, a co-simulation can still be employed i.e. to decouple systems with different time constants.

To allow the definition of company- and tool-spanning interfaces for co-simulation that ensure the numerical and physical compatibility, techniques for example from the CapeOpen standard [18] could be used as suggestions. The CapeOpen standard is commonly used in the field of process engineering and describes the coupling of partial hybrid ADE-systems. However this mathematically very ambitious method has no practical application yet.

Concluding we like to thank P+Z Engineering for providing the model used in the shown example and for the help in developing the TISC-interface for THESEUS-FE.

References

- [1] Puntigam G., Balic J., Almbauer R., Hager J.; Transient Co-Simulation of Comprehensive Vehicle Models by Time Dependent Coupling, SAE 2006-01-1604
- [2] Duhme M., Flögel H.-H., Braun M., Säger U.; Parameterstudie im thermischen Übertragungspfad Motor – Kabine; in: Wärmemanagement im Kraftfahrzeug III; Hrsg. Deussen N. ; S.1 - S.11; Essen; expert Verlag 2002
- [3] Pollack J.; CFD Simulation des Motorraums als Beitrag zur Verbesserung des Wärmemangements; in: Wärmemanagement im Kraftfahrzeug III; Hrsg. Deussen N. ; S.1 - S.11; Essen; expert Verlag 2002
- [4] Betz, J., Anzenberger T., Kobs T.; Entwicklungstendenzen der Wärmemanagementsimulation im Bereich Kühlung und Klimatisierung bei Audi; in: Wärmemanagement im Kraftfahrzeug IV; Hrsg. Steinberg P.; S.126 - S.140; Essen; expert Verlag 2004
- [5] Extessy AG; Exite Manual; Version 1.4.3.; Wolfsburg, 2005, <http://www.extessy.com>
- [6] <http://www.mpcci.de>
- [7] Hessel E.; Standardisierte Modellbibliotheken für die Automobilindustrie – Aktueller Stand der Arbeiten des AK30 in der FAT; Vortragsband der Virtual Vehicle Creation, Stuttgart 2006 und <http://fat-ak30.eas.iis.fraunhofer.de>
- [8] <http://www.modelica.org>
- [9] Fritzson P.; Object-oriented Modelling and Simulation with Modelica 2.1; Wiley 2004
- [10] Tummescheit H., Eborn J., Pröbß K., Försterling S., Tegethoff W.; AirConditioning: Eine Modelica Bibliothek zur dynamischen Simulation von Kältekreisläufen; in: PKW-Klimatisierung IV; Hrsg. Schlenz D.; S.196 - S.214; Starnberg; expert Verlag 2005
- [11] Schneider F.; Bunzel A., Hofhaus J., Braun M., Limperich D., Cäsar R., Arntz K.-D., Schröter A., Specht B., Mönkediek T.; Entwicklung und Einführung eines einheitlichen Kältekreislaufsimulationsprogramms; in: PKW-Klimatisierung IV; Hrsg. Schlenz D.; S.185 - S.194; Starnberg; expert Verlag 2005
- [12] <http://www.dynasim.se>
- [13] <http://www.mosilab.de>
- [14] <http://www.amesim.com>
- [15] <http://www.iti.de>
- [16] <http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html>
- [17] <http://www.theseus-fe.com>
- [18] www.colan.org

Session 5b

Thermodynamic Systems for Cooling Applications

Optimization of a Cooling Circuit with a Parameterized Water Pump Model

Dragan Simic Christian Kral Hannes Lacher
Arsenal Research

Giefinggasse 2, 1210 Vienna, Austria

phone +43-50550-6347, fax +43-50550-6595, e-mail: dragan.simic@arsenal.ac.at

Abstract

In this work an electrically operated water pump in a cooling circuit is implemented, simulated and optimized. The presented simulation results are accomplished with *Dymola*. This simulation tool is based on the modeling language *Modelica*.

The model of the water pump is realized with time domain differential equations. The parameters of the differential equations of the water pump are defined by geometrical data. The water pump is driven by an electric machine. A generator model and a battery model are used as power sources for the electric machine driving the water pump. In the proposed application the thermal behaviour of all components of the cooling circuit (internal combustion engine, pipeline, cooler...) is implemented.

The measured and the simulated characteristics of the water pump are compared. The efficiency improvement of the internal combustion engine when using an electrically operated water pump instead of a mechanically operated pump is presented.

Keywords: cooling circuit; water pump

1 Introduction

In this work a cooling circuit of an internal combustion engine (ICE) with an electrically operated water pump is presented. The results of the simulation lead to an optimized size of the water pump. All components of the cooling circuit including the water pump are modeled with the *Modelica.Thermal.FluidHeatFlow* package of the *Modelica* standard library [1]. The mechanical and rotational components as well as the ICE are modeled with the *Modelica* standard library, too. For the electrical drive operating the water pump a model of an electrical machine including converter from the

SmartElectricDrives (SED) library is used. From the SED, an idealized battery and a model of a generator serve as electrical power sources.

The focus of this work is on the optimization of the size of the water pump for the cooling circuit. The cooling circuit model includes the following components: an ICE, a thermostat, a cooler, a fan, a pipeline and an electrically operated water pump. For the verification of the water pump model, a test bench with a mechanically operated water pump was built.

2 Water pump model

Basically, the model of the water pump is implemented by differential equations. The parameters and coefficients of the differential equations are determined by geometrical parameters of the water pump. Losses of following kind were considered in the model: flow losses including hydraulic losses of the operating fluid; hydraulic losses of the impeller wheel; impact losses of the operating fluid flowing through the distributor. Mechanical friction losses contain the mechanical losses of the bearing and the seal.

2.1 Water pump design

The water pump can be seen as a fluid flow machine. The action principle of a fluid flow machine is based on the energy theorem. Using this theorem the water pump can be modelled. The driving torque of the water pump shaft is transmitted to the curved shovels of the impeller wheel which exert a pressure on the operating fluid. On account to the centrifugal effect, the operating fluid is hurled outwards in radial direction of the impeller wheel and therefore, leaves the impeller wheel at the outside extent with higher speed. Because of the diffuser effect in the shovel channels of the impeller wheel, the operating fluid leaves the impeller

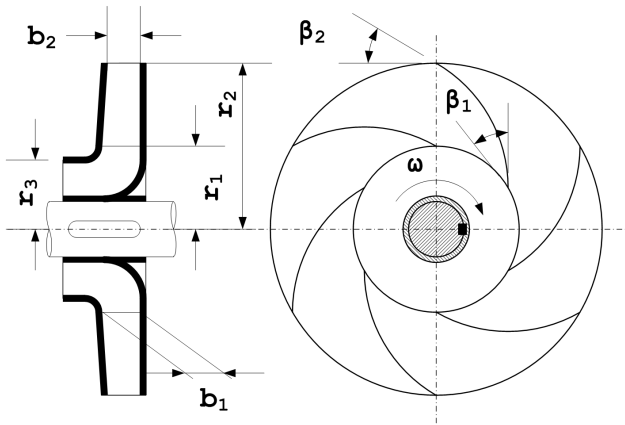


Figure 1: The impeller wheel design and parameters

wheel with increased pressure, as well. The incoming operating fluid gets sucked because the mass movement of the operating fluid outwards causes a negative pressure at the inlet of the impeller wheel. With this energy theorem the energy transformation from the driving energy of the water pump shaft to the potential energy of the operating fluid is defined [2].

2.2 Main equation of the fluid flow machine

An ideal model of the water pump is considered as a fluid flow machine with an infinite number of shovels. In this case the energy balance between the converted energy in the impeller wheel of the water pump and the kinetic energy of the operating fluid is fulfilled. Therefore, the main equation of the ideal fluid flow machine and the ideal water pump represent the relationship between mechanical energy of the input shaft with respect to the impeller wheel and the specific energy of the operating fluid [3]. The main equation of the ideal fluid flow machine and the ideal water pump [2, 4] is represented in (1):

$$\dot{m} \cdot \omega = 2 \cdot \pi \cdot b_2 \cdot \rho \cdot \tanh(\beta_2) \cdot (r_2^2 \cdot \omega^2 - Y_\infty) \quad (1)$$

In this equation r_2 is the outer radius of the impeller wheel of the water pump, b_2 is the outer width of the impeller wheel, ρ is the density of the operating fluid and β_2 is the outlet angle of the shovels of the impeller wheel. The mass flow of the operating fluid in the water pump is \dot{m} and the specific energy of the impeller wheel for an infinite number of impeller shovels with Y_∞ respectively. The basis impeller wheel design of a water pump [5] is represented in the Figure 1.

2.3 Decrease of the specific energy

A rate of the mechanical rotational energy of the impeller wheel gets reduced by the decreasing twist in the outlet area of the impeller wheel. This reduction of the energy is equal to the difference between the release energy of the impeller wheel and the absorption energy of the operating fluid. This energy reduction, Y_k , of the water pump can be calculated with a loss factor k . The energy reduction of the water pump [2] is represented by (2). With an empirical number, known as *Pfleiders* number, p , the loss factor, k , can be found using (3). For the calculation of *Pfleiders* number in practice there exist two equations: one equation for single curved and one equation for double curved impeller shovels [2]. In this paper a water pump for cooling circuit of the ICE is modeled. This type of the water pump is designed with single curved impeller shovels [3]. For calculating *Pfleiders* number (4) is used.

$$Y_k = (1 - k) \cdot Y_\infty \quad (2)$$

$$k = \frac{1}{1 + p} \quad (3)$$

$$p = \frac{a}{z} \cdot \left(1 + \frac{3 \cdot \beta_2}{\pi}\right) \cdot \frac{r_2^2}{r_2^2 - r_1^2} \quad (4)$$

In these equations z is the number of the impeller shovels of the water pump, r_1 is the radius of the inlet area of the impeller wheel, and the constant parameter, a , is an experimental coefficient, with a value between 1.2 and 2.0, depending on the design of the impeller wheel and the diffuser [3].

2.4 Hydraulic losses in shovel channels

A particular part of the energy stored in the operating fluid between the shovels of the impeller wheel is reduced by flow resistances outside of the shovel channels. These losses, known as hydraulic losses of operating fluid, are determined by hydraulic friction, changes of the operating fluid flow direction, and changes of the flow cross sectional area in the shovel channels. The calculation of the hydraulic losses is not trivial. A simplified calculation of the hydraulic losses of the operating fluid in the shovel channels is represented by (5), where Y_h determines the hydraulic losses and c is the average velocity of the operating fluid in the shovel channels. The parameter ξ is the friction factor of the hydraulic losses and can be calculated using the flow equation, taking the geometry

of the impeller wheel and the shovels into account. In this case the friction factor, ξ , is considered as a constant parameter [3].

$$Y_h = \xi \cdot \left(\frac{c^2}{2} \right) \quad (5)$$

2.5 Impact losses

In an arbitrary operating state deviating from the nominal operating state, impact losses occur. In such an arbitrary operating state the fluid flow direction is not tangential with respect to the shoveled channels of the impeller wheel and the control device. The impact losses occur if the impeller wheel is overloaded or underloaded. The proportion of the impact losses is depending on speed and load of the impeller wheel. For calculating the impact losses (6, 7) are used [2, 4]. In the (6), Y_i represents the impact losses, a_i is an experimental value according to (7), r_3 is the radius of diffuser and \dot{m}_A is the fluid mass flow at the nominal operating point of the water pump.

$$Y_i = \frac{a_i}{2} \cdot \left[r_2^2 \cdot \omega^2 + \left(\frac{r_2 \cdot \omega}{1+p} \right)^2 \cdot \left(\frac{r_2}{r_3} \right)^2 \right] \cdot \left(\frac{\dot{m} - \dot{m}_A}{\dot{m}_A} \right)^2 \quad (6)$$

$$a_i = (0.3 \dots 0.6) \cdot \frac{2 \cdot \beta_2}{\pi} \quad (7)$$

2.6 Friction losses of the impeller wheel

Between the faces of the impeller wheel and the housing of the water pump a fluid film exists. This fluid film is responsible for the friction losses, known as friction losses of the impeller wheel, P_r . These losses can be determined using (8) according to [2]. The variable ν is the kinetic viscosity of the operating fluid which equals the viscosity of coolant in this case.

$$P_r = 8 \cdot 10^{-4} \cdot \left(\frac{10^6}{\omega \cdot r_2 / \nu} \right)^{1/6} \cdot \rho \cdot \omega_2 \cdot r_2^5 \quad (8)$$

3 Configuration of the test bench

The test bench of the mechanical water pump is needed for the determination of variables that cannot be measured if the water pump is embedded in the vehicle. For measuring the family of characteristics like pressure difference and flow, as well as power consumption and hydraulically efficiency, sensors are

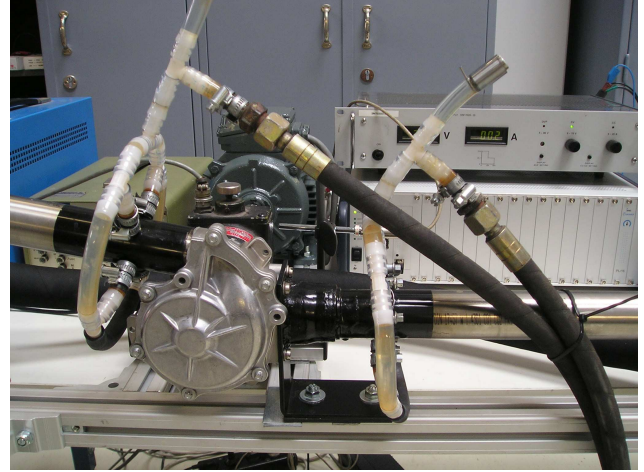


Figure 2: Test bench of the water pump

used that cannot be applied in the tight ICE compartment. The more important reason, of course, for using a test bench is that the measurements are not restricted to just one characteristic curve due to the cooling circuit of the vehicle.

For the test bench, a low resistance water circuit with 5/4 inch diameter tubes and controllable valve was built, as shown as in Figure 2. To minimize measurement errors of the differential pressure and volume several rules of measuring hydrodynamics values have to be considered. The diameter of the inlet and outlet pipe have to be the same size as well as changes in diameter must not lead to angles larger than 8 deg. At the existing water pump a rectangular inlet and an circular outlet with different cross section surfaces require changeover solutions to standardized pipes. The measurement of pressure and volume need defined inlet and outlet paths for repeatable results. The water pump delivery is measured with a magnetic inductive flow meter for electrically conductive fluids. This kind of flow meter for coolants causes the smallest backlash to the system and works accurate enough in an effective range from zero to 300 lit/min.

A pressure vessel is installed to operate the water pump with constant pressure, similar to the conditions in the vehicle, and to prevent operating fluid cavitation and degasifying of the media. If negative pressure occurs at the suction side of the water pump the media starts degasifying. Therefore a pressure of about two bar absolute was held in the system. This is equivalent to a coolant temperature of approximately 120 °C in the coolant circuit of the ICE, a typical water temperature in a modern ICE. The pressure vessel acts as expansion tank too.

During building the test bench attention had to be

drawn to the possibility of bleeding the system entirely. The torque was detected with a regular rotating torque sensor located between the water pump and the electric machine. The appropriate drive for the test bench was not to find as easy as a torque sensor. Without using a transmission, what would have brought an additional effort, a drive with an rotational speed spread from 84 rad/s to 733 rad/s had to be used. A controlled, converter fed asynchronous motor drive is operating the pump. All measurements are made in the described arrangement and beside little adjustments due to leakage the test bench worked properly during the whole measurement procedures.

The measurement results obtained from the test bench deduce accurate statements about the behavior of the water pump and the energy saving potential with an optimized electrically operated water pump.

4 Evaluation of the water pump model

For the evaluation of the water pump a simple circuit is modeled. The circuit model is implementet according to the test bench. The main components of circuit model are: the water pump, the pipeline systems and an electric drive. The following geometrical parameters of the water pump are measured and applied to the simulation:

$$r_1 = 42/2 \text{ mm}$$

$$r_2 = 62/2 \text{ mm}$$

$$r_3 = 51/2 \text{ mm}$$

$$b_1 = 9 \text{ mm}$$

$$b_2 = 6.6 \text{ mm}$$

$$\beta_1 = 15 \text{ deg}$$

$$\beta_2 = 50 \text{ deg}$$

$$\dot{m}_A = 3.5 \text{ kg/s}$$

Figure 3 shows the circuit of the test bench modeled in *Modelica*. The valve creates the friction losses in the circulation. While the water pump is driven at constant speed, the friction losses are adjusted by the valve. The specific speed of the water pump equals the specific speed measured at the test bench. The circuit is investigated for five different valve settings.

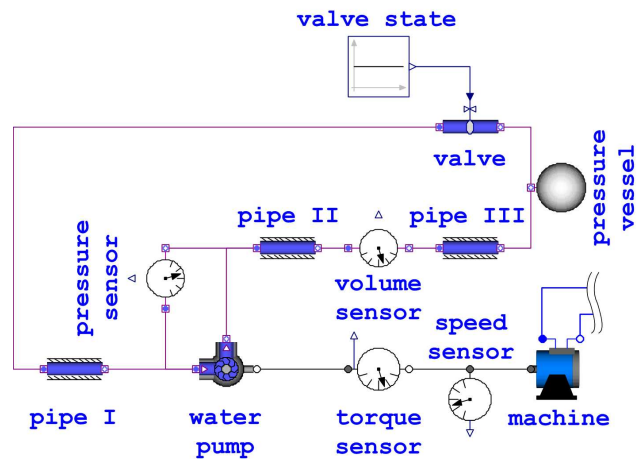


Figure 3: *Modelica* model of test bench

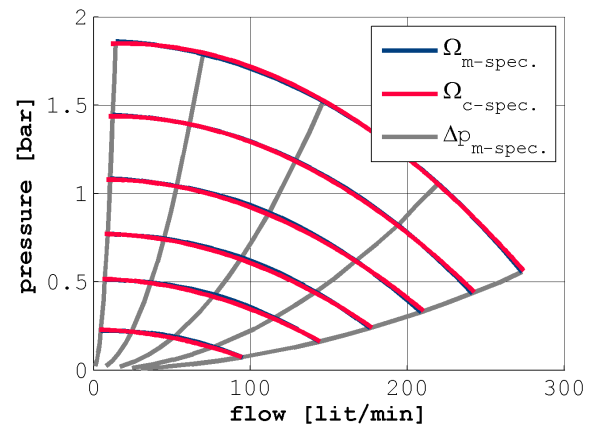


Figure 4: The measured and simulated characteristic curves of the water pump

Figure 4 represents the characteristic flow and pressure diagram of the water pump. Quantity $\Omega_{m-spec.}$ represents the specific speed of the water pump, which is measured at the test bench, $\Omega_{c-spec.}$ is the specific speeds of the water pump, which is simulated with the *Modelica* model. The $\Delta p_{m-spec.}$ is the specific flow resistances of the circuit, which is defined with the setting of the valve.

A comparison between the test bench and the simulation shows good coherence. It can be seen that the measured and calculated characteristic curves of the water pump look alike with differences in a band of merely a few percent. The difference between the two characteristic curves is very low. Figure 4 confirms therefore the good implementation of the water pump model. Furthermore, the *Modelica* model of the water pump can be used as a verified *Modelica* model for further implementations and designs of the ICE cool-

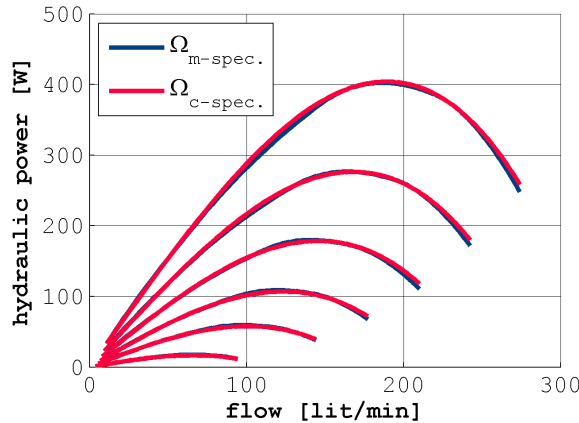


Figure 5: The measured and simulated characteristic curves of the water pump

ing circuit.

Figure 5 represents the hydraulic power of the water pump versus flow. In this figure $\Omega_{m-spec.}$ is the measured specific speed and $\Omega_{c-spec.}$ is the simulated specific speeds of the water pump in the flow and hydraulic power diagram. The very low difference between the measured and simulated hydraulic power of the water pump approve the good implementation of the *Modelica* water pump model.

5 Cooling circuit

All components of the cooling circuit are implemented in *Modelica*. Each of the implemented models of the cooling circuit considers flow equation. The mass flow balance of the operating fluid from the fluid inlet and the fluid outlet is defined in the equations stated in [6]. The friction losses of the pipe components and the valves are implemented with characteristic curves and the models are extended from *Modelica.Thermal.FluidHeatFlow* library. All another components of the cooling circuit are implemented as physical models.

The cooler of the cooling circuit is modeled by means of discrete volume elements [7, 8]. In the cooler model the coefficients of convection for the coolant, the air and the steel tubes of the cooler are determined by calculation. The effect of the cooler fan was determined by calculation of the coefficients of convection. The discrete volumes, which flow through the cooling fan area of the cooler have a higher coefficient of convection. In Figure 6 the temperature distribution of the discrete cooler volumes is shown. One significant results is that the discrete volumes in the cooling fan area

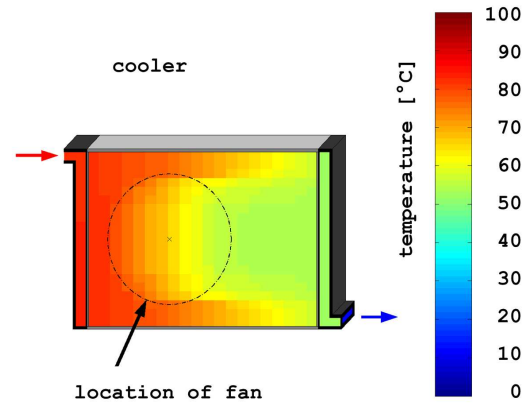


Figure 6: Temperature distribution of the cooler

have a higher temperature decrease.

5.1 Electrically operated water pump

Figure 7 shows the cooling circuit with the electrically operated water pump and cooling fan. The electric power source of the cooling circuit is an idealized battery model. The battery feeds the pump motor, which is controlled by a pump speed controller (PSC) model. The PSC calculates the reference speed of the pump motor for the actual temperature of the ICE. The pump is driven by this pump motor. The pipeline configuration is design according to a conventional ICE cooling circuit. The cooler fan, marked as fan, is driven by a fan motor fed by the same electric sources than the water pump. A generator, marked as gen., is used as electrical energy source, mechanically coupled with the ICE and electrically coupled with the battery. The generator torque controller (GTC) model calculates the torque state of the generator depending of the state of the charge (SOC) of the battery. The ICE thermal model is implemented with the characteristic curves of the ICE. With this concept of the cooling circuit it is possible to control the speed of the water pump independent from the speed of the ICE. The speed of the cooling fan is constant in this model of the cooling circuit. All electric machines are taken from the SED library using the quasi stationary permanent magnet synchronus machine models.

The efficiency of electric machines at the nominal operating point is 0.95, and the efficiency of battery is 0.87. The efficiency of generator belt drive is taken as the efficiency from a flat belt drive and accounts for 0.96 [9].

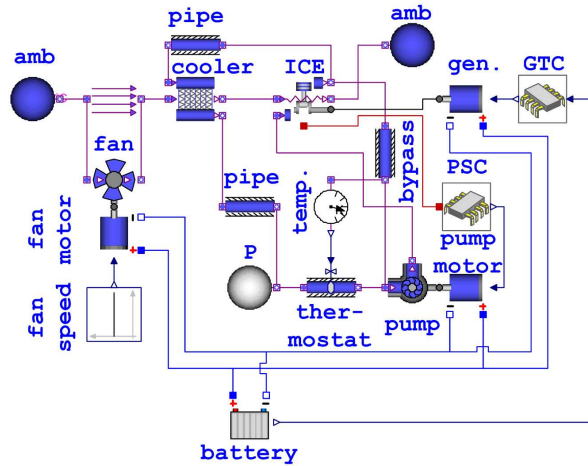


Figure 7: *Modelica* model of cooling circuit with electrically operated water pump and cooling fan

5.2 Mechanically operated water pump

The mechanical model of the cooling circuit contains the mechanically operated water pump. The major difference to the model of Figure 7 is the mechanically operated water pump. The water pump is driven with a belt drive from the ICE shaft. The efficiency of the belt drive is approximately 0.96 [9].

6 Cooling circuit optimization

For optimizing the cooling circuit the different concepts are simulated und compared. First, the cooling circuit with the mechanically operated water pump is simulated and then the electrically operated water pump is considered. The ICE is driven with specific speed and torque. The speed band is used from idle speed to upper speed limit of the ICE and the torque is used from zero to maximal torque at according to the actual speed of the ICE. Therewith the ICE is driven in the entire speed and torque range of the characteristic map. In the simulation the maximal acceptable temperature of the ICE is used. The speed of the cooling fan for both concepts is the maximal speed of the fan.

Figure 8 shows the different efficiency maps of the ICE. Quantity $\eta_{mech.}$ is the efficiency of the ICE with the mechanically operated water pump, the $\eta_{elec.}$ is the efficiency of the ICE with the electrically operated water pump and the $\tau_{max.}$ the maximal torque of the ICE which is depending on the ICE speed.

In Figure 8 the difference between the ICE efficiency maps can be determined. The increase of ICE efficiency in case of a cooling circuit with electrically

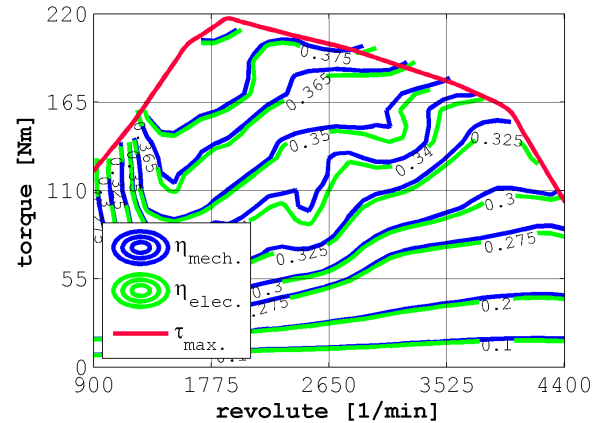


Figure 8: Efficiency comparison of the ICE with electrically and mechanically operated water pump

operated water pump can be identified clearly. There is an improvement of efficiency with respect to almost the complete map of the ICE. The efficiency increase is higher in upper speed and torque regions of the ICE. This increase can be expected, because the nominal operating point of the mechanically driven water pump is designed for lower speeds of the ICE.

7 Conclusions

The performed simulation of cooling circuits allows the determination of the fuel consumption as well as a declaration of the economic saving potential of an electrically operated water pump. The fuel and energy consumption of an ICE can be reduced by means of an electrically operated water pump.

The efficiency increase of the ICE was demonstrated for stationary operation. For dynamic operation, however, a similar improvement can be expected. Using the concept of an electrically operated water pump a thermal shock of the ICE and the cooling circuit can be avoided. The concept of an electrically operated water pump increases the efficiency of a conventionally cooled ICE and makes energy saving potentials possible this way.

The experimental verification of the water pump model is presented. The presented model of the water pump can therefor also be applied to other thermal management models like battery cooling, fuel cell cooling or cooling of electric machines.

8 Abbreviations

ICE	internal combustion engine
SED	<i>SmartElectricDrives</i>
PSC	pump speed controller
GTC	generator torque controller
SOC	state of charge

References

- [1] C. Kral, A. Haumer, and M. Plainer, “Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library”, *Modelica Conference*, pp. 213–218, 2005.
- [2] W. Kalide, *Energieumwandlung in Kraft und Arbeitsmaschinen*, Carl Hanser, München, 6 edition, 1982.
- [3] Johann Friedrich Gülich, *Kreiselpumpen*, Springer Verlag Berlin Heidelberg New York 2004, Heidelberger Platz 3, 14197 Berlin, 2004.
- [4] J. Fischer T. Wagner and D. Frommann, *Strömungs und Kolbenmaschinen*, Vieweg, Braunschweig, 3 edition, 1990.
- [5] D. Simic, C. Kral, and F. Pirker, “Simulation of the cooling circuit with an electrically operated water pump”, *IEEE Vehicle Power and Propulsion Conference, VPPC*, 2005.
- [6] I. E. Idelchik, *Handbook of Hydraulic Resistance*, Begell House, 3rd edition edition, 1996.
- [7] G.P. Merker and C. Eiglmeier, *Fluid- und Wärmetransport – Wärmeübertragung*, B.G. Teubner, Stuttgart, Leipzig, 1999.
- [8] G.P. Merker and C. Baumgarten, *Fluid- und Wärmetransport – Strömungslehre*, B.G. Teubner, Stuttgart, Leipzig, Wiesbaden, 2000.
- [9] F. Sass, Ch. Bouche, and A. Leitner, *Dubbels Taschenbuch für den Maschinenbau, Teil 2*, vol. 2, Springer Verlag, Berlin, 12 edition, 1963.

Using Modelica as a Design Tool for an Ejector Test Bench

Christoph Richter, Christian Tischendorf, Ricardo Fiorenzano, Peterson Cavalcante,
 Wilhelm Tegethoff, Jürgen Köhler
 Technical University Braunschweig, Institute for Thermodynamics
 Hans-Sommer-Straße 5, 38106 Braunschweig, Germany
 ch.richter@tu-bs.de

Abstract

As a result of the Kyoto Protocol, the use of R134a in air conditioning system of new cars is going to be forbidden in the EU due to the high Global Warming Potential (GWP) of this substance. Carbon dioxide (CO₂) is one of the possible alternatives as a cooling agent in mobile air conditioning applications and is desirable since it is a natural refrigerant. Cooling cycles using CO₂ are currently achieving COP similar to those of R134a cycles but there are promising options to further improve COP. One possibility is the application of an ejector instead of the valve. A Modelica library was developed that allows computation of ejector cooling cycles in steady state with simplified component models that can be used as a design tool for the construction of an ejector prototype. The library uses a new object-oriented library that serves as an interface to external medium libraries to compute thermodynamic and transport properties for the refrigerant.

Keywords: Fluid properties, cooling cycle, ejector

1 Introduction

As a result of the discussion about reducing the worldwide emission of greenhouse gases, carbon dioxide (CO₂) was re-discovered as a natural refrigerant with promising thermodynamic properties. Among a few others, Lorentzen [1] pointed out early that trans-critical CO₂ refrigeration cycles encounter significant throttling losses reducing their coefficient of performance (COP). One way to overcome this problem is to use an ejector instead of the throttling valve. A lot of research is currently carried out in this field [2], [3] and the obtained results are very promising [4].

This paper describes the development of a Modelica library that was used in the design process of a test bench for an ejector cooling cycle and that will be extended to allow the deeper analysis of the ejector

refrigeration process. The developed library is kept as simple as possible. The fluid properties for CO₂ are computed using a new fluid property library that provides a convenient interface to an external library written in C.

2 Problem description

To be able to simulate an ejector cooling cycle one needs models for the components of the cooling cycle as well as for the fluid properties of the chosen refrigerant. Although it would be possible to have one library for both requirements it usually is a good idea to have separate libraries for the components and for fluid properties. Well-known Modelica libraries such as Modelica.Media/Modelica_Fluid and AirConditioning Library/ThermoFluidPro follow the same basic concept.

Before looking at libraries that are capable of modeling the given problem the most important requirements are compiled in the following list:

- The component models should be as simple as possible for the first rough analysis during the design process. The component models should only require the definition of a minimal set of parameters and efficiencies because exact geometries are usually not known in the first rough analysis.
- There are a lot of engineers that can only spend a certain amount of their work time on understanding Modelica libraries. The developed library should enable those users as well as students with no or little background in modeling to get started with developing new models as easily as possible.
- The simple models used in the first steps of the design process should be designed in such way, that they can be replaced by more detailed models in future steps when an in-depth analysis of experimental results is required.

- The fluid property library should be as flexible as possible. It should be possible to use the same fluid properties in different applications (i.e. Modelica, Matlab, LabVIEW, CFX) and the use of the library should be as simple and straightforward as possible.
- The fluid property library should not contain any compiler specific elements and should be as flexible to use as possible.

The following subsections give a brief overview over existing libraries and point out their advantages and drawbacks within the scope of simulating a CO₂ ejector cooling cycle.

2.1 Modelica_Fluid/Modelica.Media

The Modelica_Fluid library is a free library for describing 1-dimensional thermo-fluid flow and is developed by the Modelica Association. The library is still under development and a new beta version is going to be presented at this conference [5]. The Modelica_Fluid library provides models for standard fluid components such as pipes, control valves and pumps. There are currently no models for compressors or simple heat exchangers available. Modelica_Fluid uses fluid models from Modelica.Media which is part of the Modelica Standard Library since version 2.2. Modelica.Media provides medium models for ideal gases and water. Refrigerants are currently not implemented within Modelica.Media. Modelica.Media can be extended to allow the use of external fluid property libraries implemented in FORTRAN or C.

2.2 AirConditioning Library/ThermoFluidPro

The AirConditioning Library is a commercial library developed and maintained by Modelon AB [6]. This library allows transient and steady-state simulations of air conditioning systems at a very high model complexity level. It is used as a development tool by several large automotive companies. The AirConditioning Library uses ThermoFluidPro as fluid property library that is also developed and maintained by Modelon AB. ThermoFluidPro offers high-precision equations for the fluid properties of CO₂ using the Helmholtz equation given by Span and Wagner [7].

2.3 TIL/TILFluids

TIL (TLK-Ift-Library) is a library for simulating refrigeration systems developed and maintained by the Institute for Thermodynamics (IfT) at TU Braunschweig in close cooperation with TLK (TLK-Thermo GmbH). TIL contains models with very dif-

ferent levels of complexity. TIL.HVAC contains very simple models that are used for educational and training purposes and that are especially designed to allow an easy access to this library for new users. More detailed models that are used in R&D projects are also available within TIL. TIL uses TILFluids to compute fluid properties. TILFluids does not provide fluid models implemented in Modelica but offers general interfaces to employ existing external libraries written in C and FORTRAN.

2.4 Summary and Conclusion

All three libraries could be extended to allow the computation of CO₂ ejector refrigeration cycles. Fluid properties for CO₂ are readily available in the ThermoFluidPro library and in TILFluids.

The main drawback of Modelica_Fluid was its development status when starting this project. Another drawback is the lack of simple standard components (i.e. compressors, simple gas coolers) that are needed to build up a cooling cycle. Many of the models are already too complex for the given application. Modelica_Fluid is furthermore taking into account advanced flow situations such flow reversal which are not required within the scope of this work.

The models from the AirConditioning Library are very detailed due to the intended field of application for that library. They seem to be too detailed for a rough analysis during the design process of a test bench. ThermoFluidPro is a very advanced and powerful fluid property library completely implemented in Modelica. It is an extension of the free Modelica.Media library and offers a variety of different models including models for pure refrigerants, binary mixtures, pseudo-pure fluids, and water.

TIL offers very simple as well as advanced models for components used in air conditioning systems. The simple models in TIL.HVAC are especially suitable for projects that involve students having limited or no previous experiences with Modelica.

It was therefore decided to use TIL and to implement a simple ejector model within the framework TIL.HVAC, the part of TIL that is used for educational purposes. The fluid property library TILFluids was considerably improved during this project and offers a very user-friendly object-oriented approach for fluid properties.

Developing libraries that are easy to understand and simple in their basic concept seems to be a very important task. A lot of the available libraries seem to be unnecessary complicated which makes it hard for new users to get started with Modelica. We experienced this problem with students that were asked to

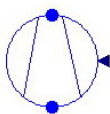
work with Modelica.Media and Modelica_Fluid. They had quite some difficulties to get started with those libraries. The same thing seems to happen with commercial partners where an engineer is asked to start modeling using one of these libraries. It is very common that engineers can only dedicate a small percentage of the work time to this task which makes it very hard to get started.

Concerning the Fluid-Properties it seems to be important that companies can reuse their code that they are already employing in other problems. These are usually FORTRAN or C libraries that they might have developed over decades. Now start looking at Modelica.Media to figure out how to implement something like an external interface for a two-phase medium. You would probably start with inheriting from PartialTwoPhaseMedium in which case one is focusing an inheritance structure that is 4 levels deep. This is not really human-readable and far from being easy to understand. One would also probably have a closer look at the implemented water model implementing the IF97 standard. The problem with this model is that it actually incorporates late inlining and a hidden property record to improve performance significantly when using Dymola. The structure of this two-phase library is not easy to understand for new users.

3 TIL.HVAC

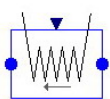
TIL.HVAC is the part of TIL that contains simple models for each component of a refrigeration cycle. The following section gives a brief overview over the most important components.

3.1 Compressor



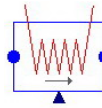
The simple compressor model uses constant volumetric, isentropic and energetic efficiencies to describe the change of state, the mass-flow rate and the energy consumption. It also allows setting the rotational speed using a standard RealInput.

3.2 Gas cooler



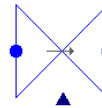
The gas cooler model assumes that the pressure drop within the gas cooler is negligible. It uses a static mass balance and a static momentum balance and allows the user to set an adiabatic efficiency. This model takes the ambient temperature as an input.

3.3 Evaporator



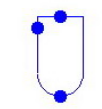
The evaporator model is quite similar to the gas cooler model. It also assumes negligible pressure drop and static mass and momentum balances.

3.4 Valve



The valve model assumes an isenthalpic throttling process. The effective flow area can be specified using the RealInput interface.

3.5 Separator



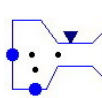
The separator model assumes that the separation process works ideal yielding saturated liquid at the liquid outlet and saturated vapor at the vapor outlet. The pressure drop is neglected which might not be realistic.

3.6 phDiagram



The phDiagram is a sensor that can be used to record pressure and specific enthalpy anywhere in the cycle. It allows for simple post processing using phMonitor, a program developed at the IfT in cooperation with TLK, to visualize the results in a ph-diagram.

3.7 Ejector



The ejector model is based on a very simple model initially developed by Kornhauser [8]. It is composed of two nozzles, a mixing section and a diffuser. The main purpose of the ejector is to recover some of the kinetic energy otherwise converted into friction and lost during the throttling process. The refrigerant is entering the primary (motive) nozzle at high pressure and is accelerated. The high velocity stream leaving the primary nozzle is used to entrain refrigerant from the secondary (suction) nozzle. Both streams are mixed in the mixing section. The diffuser is used to convert parts of the remaining kinetic energy of the mixed stream into a pressure increase that helps to improve the overall performance of the cycle.

The following assumptions were made for the ejector model. The analysis is one-dimensional and the refrigerant is in thermodynamic-equilibrium at all times. Those first two assumptions correspond to

what is called homogeneous equilibrium model in two-phase flow. It is furthermore assumed that the deviations from adiabatic reversible processes that occur in the nozzles and the diffuser can be expressed in terms of efficiencies. Any shock effects that might occur are included in these efficiencies. The kinetic energy was considered to be significant only within the ejector not within the rest of the cycle.

Some of these assumptions will have to be reconsidered for a more detailed analysis. One of the main problems of the implemented model is the missing relationship between mass flow rate and pressure drop for the nozzles which requires the explicit specification of at least one pressure level (i.e. the mixing pressure). Other models such as the one developed by Groll [2] suffer from the same problems. Experimental analysis of the ejector refrigeration cycle which is currently performed at the IfT might overcome this drawback.

Figure 2 shows the internal structure of the developed ejector model.

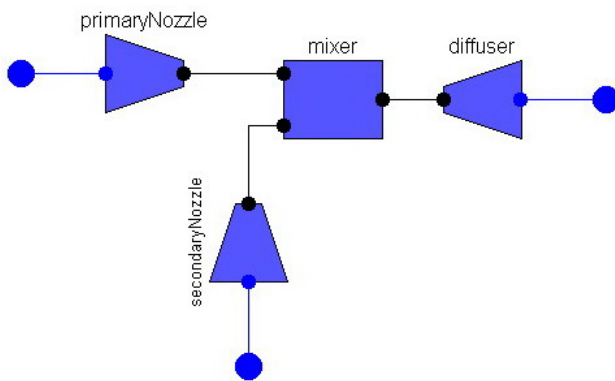


Figure 2: Internal structure of the ejector model.

4 TILFluids

TILFluids is an object-oriented fluid property library. The main design goal when developing this library was to create a simple interface that can easily be extended by users to fit their individual needs. Figure 1 shows the general structure of the library. The package Common contains common elements such as types and records that are used throughout the library. The package Icons contains the library icon. The package Internal is the core of the library and contains all functions that are needed to access the external library. The contained functions are not used directly in the component model but rather accessed using a fluid object such as Gas, Liquid or Refrigerant.

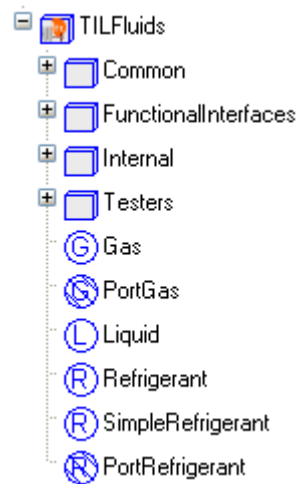


Figure 1: Structure of TILFluids

The provided fluid objects such as “Refrigerant” and “Gas” are ready-to-use implementations of medium models that can be used in a wide range of applications. They can also be changed or extended by developers to best meet their own requirements.

Figure 3 shows an example for a simple refrigerant model. The unqualified import statement ensures that the internal functions required for computing the fluid properties (i.e. “pressure_dT()”) are available within the model “MyRefrigerant”. The “refrigerantName” is a character string with a couple of predefined choices specified in TILFluids.Common. The “uniqueID” is an integer that uniquely identifies each instance of “MyRefrigerant”. All functions in the external library require the “uniqueID” as an input. The external library creates a map of refrigerants and stores information about the refrigerant and the results from the last computation within this map. This allow for an optimization regarding the computational time requirements. The library can be designed in such a way that it returns the values computed at the last call for each refrigerant if the inputs did not change. This is very important especially in the case when an inverse iteration is required for the computation.

```
model MyRefrigerant "Example refrigerant (inputs are d and T)"
import TILFluids.Internal.Refrigerant.*;

parameter TILFluids.Common.RefrigerantName refrigerantName;

SI.Density d "density";
SI.SpecificEnthalpy h "specific enthalpy";
SI.AbsolutePressure p "pressure";
SI.Temperature T "temperature";

Integer uniqueID(final start=0) "unique ID number";
algorithm
when {initial()} then
  if {uniqueID == 0} then
    uniqueID := getUniqueRefrigerantID();
    setRefrigerantName(refrigerantName, uniqueID);
  end if;
end when;
equation
p = pressure_dT(d, T, uniqueID);
h = specificEnthalpy_dT(d, T, uniqueID);
end MyRefrigerant;
```

Figure 3: Code example for a simple refrigerant model that uses function calls to an external library to compute fluid properties

The above example is instantiated in the component model and contains all required fluid properties. Actual refrigerant models will contain more thermodynamic and transport properties than the simple “MyRefrigerant” from example above. Figure 4: Code example for a simple component model to demonstrate the use of the fluid object. Figure 4 shows a simple component model with an instantiated refrigerant model. Density and temperature are specified for “myRefrigerant” and the pressure and specific enthalpy are computed within “MyRefrigerant” and are attributes of “myRefrigerant”. The object-oriented approach used in TILFluids was found to be very user-friendly and allows for a clear separation of component and fluid model. The TILFluids Users’ Guide [11] offers more detailed information on using and extending TILFluids.

There are situations where a functional approach might be handier than the object-oriented approach used in TILFluids. TILFluids therefore provides functional interfaces that can be called without instantiating a medium. The functional interfaces are contained in TILFluids.FunctionalInterfaces. However the user should be aware of certain limitations that apply to the functional interfaces (see [11] for more details).

```

model MyComponent "Example component"
  MyRefrigerant myRefrigerant {refrigerantName="CO2"}
    "refrigerant object";

  SI.Density d "density";
  SI.Temperature T "temperature";

  SI.AbsolutePressure p "pressure";
equation
  myRefrigerant.d = d;
  myRefrigerant.T = T;

  d = 10 + 990*time;
  T = 273.15;

  p = myRefrigerant.p;
end MyComponent;

```

Figure 4: Code example for a simple component model to demonstrate the use of the fluid object.

The current external library that is used with TILFluids provides interfaces to two different libraries. The first interface uses the fluid properties used in a platform for steady-state computation of cooling cycles that was developed at IfT in the past [9]. The second interface calls the current beta version of Refprop (for details see [10]) to obtain fluid properties. An additional interface to a library that implements the IF97 standard for water is currently under development. The TILFluids users’ manual offers more detailed information on how to use TILFluids [11].

The interface to a C library of fluid property data developed at IfT is available for free, the Refprop interface upon request. Additional interfaces can be implemented if required.

One of the advantages of having medium models implemented in Modelica is that the simulation software can take full advantage of the entire equation. It can use this information to compute analytical Jacobians or to perform index reduction. Using an external library might require the explicit definition of derivative functions using annotations. This is currently not implemented in TILFluids but can be added at a later time. It is however very often possible and desirable to formulate the problem in such a way that no index reduction is required to solve it. The Bridgman tables [12] can be used in many thermodynamic problems to accomplish this task.

5 Example

The main goal when developing the presented libraries was to perform preliminary computations for an ejector cooling cycle. A comparison of a conventional cooling cycle and an ejector refrigeration cooling cycle was carried out in a first step to demonstrate the theoretically achievable COP improvements. In a second step a certain design point was chosen for the test bench and all required geometrical parameters were computed (see [13] for more details).

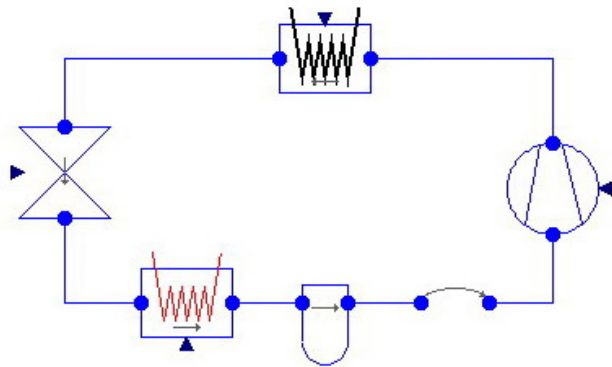


Figure 5: Schematic diagram of a standard cooling cycle (the extra component behind the receiver breaks the algebraic loop for the mass flow rate)

An ejector refrigeration cycle is compared to a standard trans-critical CO₂ cooling cycle to determine the possible gain in COP for the ejector cycle. The COP is defined as

$$COP = \frac{Q_{evap}}{W_{p,el}}$$

where Q_{evap} is the evaporator refrigeration capacity and $W_{p,el}$ is the electrical power consumption of the compressor. Figure 5 shows the standard cooling cycle and Figure 6 the ejector cooling cycle.

The same conditions were applied to both cycles. The simple compressor model assumes a volumetric efficiency of 70% and an isentropic efficiency of 70%. The displacement was set to 30 cm³ and the speed to 20 Hz. The evaporation temperature was set to 0°C and the gas cooler was assumed to work with 100% efficiency, this means that the CO₂ outlet can be cooled to ambient temperature.

For the one stage CO₂ cycle, the throttling valve was assumed to operate isenthalpic. The opening area of the valve was varied to achieve maximal COP.

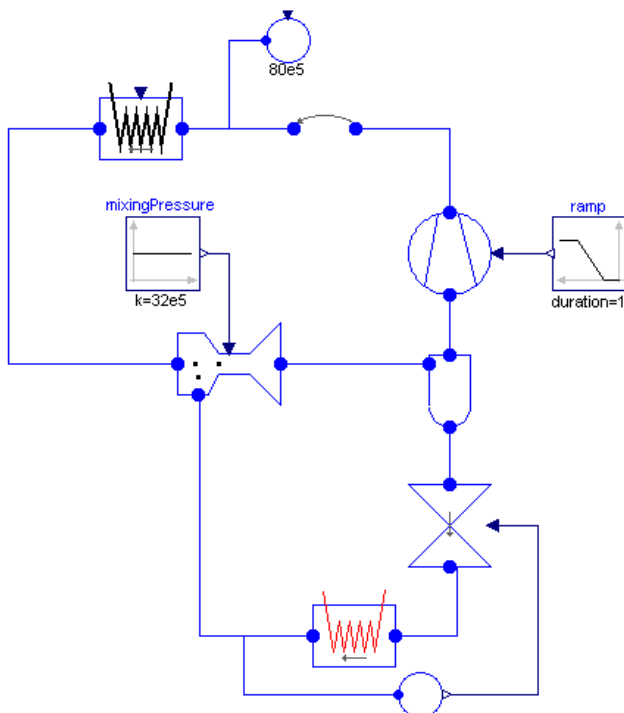


Figure 6: Schematic diagram of ejector cooling cycle

In the ejector cycle, the ejector efficiencies were set to 100% for the first cycle computations to estimate the maximum COP improvements for the ejector cycle compared with a conventional cycle. That means, that all three nozzles are working isentropic and that the mixing occurs without any losses. However, this will not be the case for the real ejector. Experimental results from [2], [3] show, that there are significant losses especially in the suction nozzle and during the mixing process. The mixing pressure for the ejector cycle was set to 32 bar. Computations to determine the effects of the mixing pressure on cycle efficiency and to determine the optimal mixing pressure are currently carried out at the IFT.

COP computations were carried out for ambient temperatures from 26°C to 40°C and compressor outlet pressures from 86 bar to 120 bar. The COP for each ambient temperature that yielded the best COP was chosen for the comparison of the two cycles.

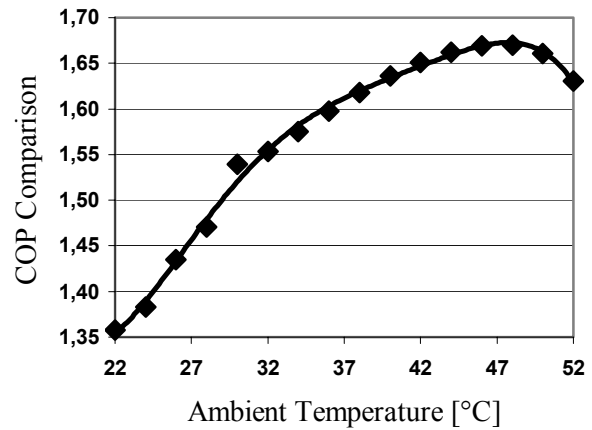


Figure 7: Comparison of COP of the conventional and the ejector cooling cycle.

Figure 7 shows a comparison of the COP of the two cycles. The performance of the ideal ejector cycle is clearly better than the performance of the conventional cycle for all operating conditions. However, one should keep in mind that ideal working conditions were assumed and that Figure 7 does not represent the expected improvement in COP for a real cycle. However, it shows the potential of the ejector cooling cycle.

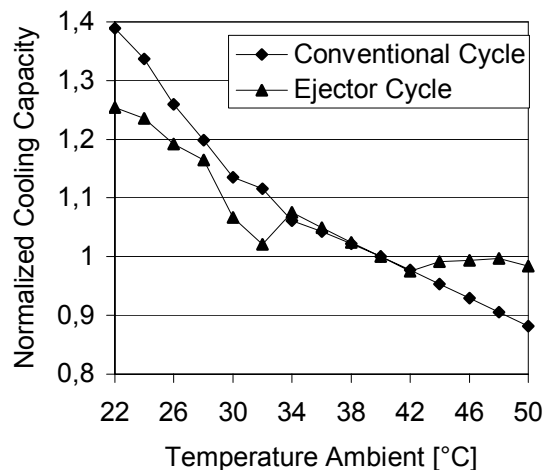


Figure 8: Comparison of normalized cooling capacity of the conventional and the ejector cooling cycle

Figure 8 shows a comparison of the normalized cooling power of a conventional and an ejector cooling cycle for different ambient temperatures.

The cooling power of the conventional refrigeration cycle is declining drastically for high ambient temperatures due to the raising losses in the throttling process. The ejector cooling cycle can regain some of this otherwise dissipated energy.

The analyzed ejector cycle does not use an internal heat exchanger which might actually improve the performance in some regions. An analysis of the effect of using an internal heat exchanger will be investigated in detail in the future.

Using the same analysis it could be shown that the ejector efficiencies have to be better than 70% to achieve a better overall performance than for the conventional cooling cycle.

One design point for a heat pump application was chosen for the test bench and all geometric parameters have been computed for that design point. The gas cooler outlet temperature was assumed to be 20 – 25°C with a gas cooler outlet pressure of 95 bar. The gas cooler power was set to 4 – 5 kW and the evaporator pressure to 37 – 40 bar. A driving mass flow rate of approximately 0.02 kg/s is required to obtain the designated heating power. The ejector geometry was determined using the results from the cycle analysis for this operating point.

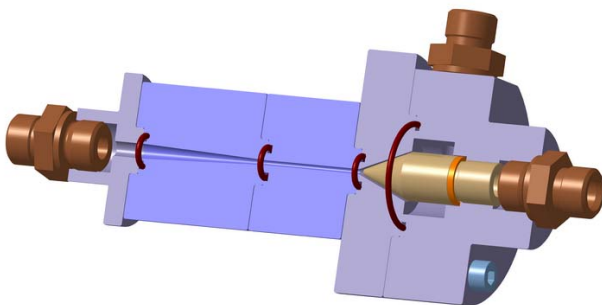


Figure 9: Cross-sectional view of the designed ejector

Figure 9 shows the designed ejector in a cross-sectional view. The prototype is currently tested at IfT. The experimental results will be used to further improve the ejector model.

6 Conclusion and future perspective

An existing simple library with standard elements to compute a cooling cycle has been extended to allow the steady-state computation of ejector cooling cycles. A new fluid property library for Modelica has been developed that is easy to use and that offers the possibility to use existing external libraries to compute fluid properties. The results from the first com-

putations were used to set up a test bench for ejector cooling cycles at the IfT.

Transient models for the cooling cycle components and for the ejector are currently developed to allow a detailed analysis of the cooling cycles. The library is also extended to allow an exergy analysis based on the second law of thermodynamics. The results from the designed test bench will be used to further improve the existing models.

References

- [1] Lorentzen G., 1983, Throttling – the Internal Haemorrhage of the Refrigeration Process, *Proc. Inst. Refrig.*, Vol. 80:39-47.
- [2] Li D. and Groll E.A., 2006, Analysis of an Ejector Expansion Device in a transcritical CO₂ Air Conditioning System, *Proc. 7th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Trondheim, Norway.
- [3] Elbel S. and Hrnjak P., 2006, Experimental Validation and Design Study of a Transcritical CO₂ Prototype Ejector System, *Proc. 7th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Trondheim, Norway.
- [4] Ozaki Y., Takeuchi H. and Hirata, 2004, Regeneration of Expansion Energy by Ejector in CO₂ Cycle, *Proc. 6th IIR Gustav Lorentzen Conference on Natural Working Fluids*, Glasgow, UK.
- [5] Casella F., Otter M., Proelß K., Richter C., Tummescheit H., 2006, The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks, *Proc. 5th International Modelica Conference*, Vienna, Austria.
- [6] Modelon AB, <http://www.modelon.se>, Sweden
- [7] Span, R., Wagner, W. 1996, A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa, *Journal of Physical and Chemical Reference Data*, 25/6: 1509-1596.
- [8] Kornhauser A., 1990, The Use of an Ejector as an Refrigerant Expander, *USNC/IIR Purdue Refrigeration Conference*, Purdue, Indiana
- [9] Tegethoff W., 1999, Eine objektorientierte Simulationsplattform für Kälte-, Klima- und Wärmepumpensysteme, *Diss. TU Braunschweig*, Germany

- [10] Lemmon E.W., McLinden M.O., Huber M.L., 2002, NIST Reference Fluid Thermodynamic and Transport Properties – REFPROP, Version 7.0, Users’ Guide
- [11] Richter C.C., Cavalcante P., Tegethoff W., 2006, TILFluids – Users’ Manual, Version 0.9
- [12] Bejan A., 1988, Advanced Engineering Thermodynamics, Wiley-Interscience
- [13] Tischendorf, C., 2006, Aufbau eines Prüfstandes für Ejektoren und Ejektor-Kältekreisläufe, TU Braunschweig, Diplomarbeit

Modeling of Frost Growth on Heat Exchanger Surfaces

Katrin Prölss, Gerhard Schmitz

Hamburg University of Technology, Applied Thermodynamics Group
Denickestr. 17, 21075 Hamburg

Abstract

A Modelica model to study the frost growth on parallel heat exchanger plates is developed. The coupled heat and mass transport phenomena involved in the process are described. A physical model which includes the distributed densification of the frost layer is compared to a simple lumped model approach. The frost structure and thus the resulting properties such as density and thermal conductivity are found to have a strong influence on the model results. Empirical correlations for properties and transport coefficients are used in both models. The model describing the air-flow is taken from the *Modelica_Fluid* library and extended to handle condensation and freezing of water vapor. The *Modelica.Media* library provides the moist air property model.

1 Introduction

Frost formation on heat exchanger surfaces in air cooling applications always occurs when the coil surface temperature drops below the freezing point of water and the dew point temperature of the air in contact with the cold surface. The main problems associated with frost growth are the decline in heat exchanger efficiency resulting from an insulating effect of the frost layer and the rising pressure drop due to a decreasing hydraulic diameter of the flow channel, which in return increases the energy consumption of the fan. Usually a system operating under frosting conditions needs a defrost from time to time. In order to estimate the allowable frost growth period and the energy required to melt down the frost layer, its stored amount of frozen water, thickness and thermal behavior have to be known. Several studies exist which try to formulate empirical approaches to predict the behavior of certain types of heat exchangers and surfaces, see e.g. [1], [2]. Hoffenbecker *et al.* [3] investigated defrost cycles with fin-and-tube heat exchangers and developed a numerical model describing the process with

given initial frost conditions. Over the past decades several attempts have been made to develop physical models of the transport phenomena which govern the general formation of a frost layer. In 1974 Sanders [4] already proposed a system of partial differential equations describing the coupled heat and mass transport processes in his detailed study on frost structure and provided several simplified approaches. Le Gall *et al.* [5] and Tao *et al.* [6] developed more detailed models and focused on the diffusion process of water vapor through the porous medium. But due to the complex structure and variety of ice crystal formation an entirely physical approach is hardly possible and the quality of the model still relies on some empirical correlations.

The transport phenomena involved in the frost formation process are sketched in figure 1. Warm moist air

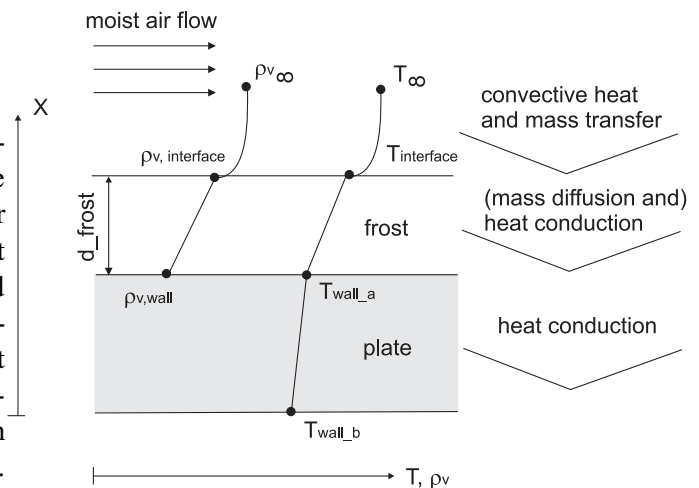


Figure 1: Transport phenomena in a porous frost layer

passes a cold surface with a temperature below the freezing point of water and the dew point of the air. Heat transfer by forced convection occurs from the bulk flow to the cryosurface, driven by a temperature gradient. A concentration gradient causes a water vapor flux from the bulk flow to the frost surface where part of it freezes and adds to an increase of the layer while releasing latent heat. Part of the water vapor dif-

fuses into the frost layer leading to a densification of the structure. The sum of sensible and latent heat is transported to the cold wall interface by thermal conduction.

2 Modeling the frost formation process

2.1 Frost formation types

During the process of frost deposition on a cold surface several stages of frost formation may be distinguished. Hayashi *et al.* [7] found three periods describing the formation of a frost layer:

- the crystal growth period
- the frost layer growth period
- the frost layer full growth period

The first corresponds to the first formation of ice crystals on the cryosurface, which basically form parallel columns or needles growing in one dimension. In the frost layer growth period they start interbranching which leads to a meshed more uniform frost layer. The surface temperature of the frost gradually increases with increased frost thickness due to a rising thermal resistance. When the triple point temperature is reached the frost surface begins to melt, water soaks into the frost and decreases its thermal resistance, which again leads to additional frost deposition and under continuous melting and freezing eventually to a dense and tight layer. This stage is referred to as the full growth period. In the following model only the frost layer growth period will be considered.

Hayashi *et al.* classified four types of frost crystal structure with respect to the cryosurface temperature and the water concentration difference between bulk flow and surface. Generally, higher cryosurface temperatures and lower concentration gradients lead to denser and more homogeneous frost layers. In the same study the frost type was found to have a strong influence on frost properties like density and thermal conductivity.

2.2 Energy and mass balances

Heat and mass transport through the ice layer are considered one-dimensional, because a prevailing temperature gradient and a frost dimension very small compared to the other two is assumed perpendicular to wall and air flow. Further assumptions are that temperature

variations of moist air and ice properties are neglected, the total gas phase pressure is constant throughout the porous medium, radiation effects are neglected and thermodynamic equilibrium prevails between ice and air in the pores.

The water mass balance over the ice phase of the frost layer can be written as the change of accumulated ice and water vapor equal to the spatial change of the diffusive vapor transport rate [5].

$$\frac{\partial(\rho_{\alpha} \epsilon_{\alpha})}{\partial t} + \frac{\partial(\rho_{v,sat} \epsilon_{\beta})}{\partial t} = -\frac{\partial \dot{m}_v}{\partial x} \quad (1)$$

where \dot{m}_v is the water vapor mass flow per unit area and ρ_{α} is the density of ice. The moist air in the frost pores is always assumed to be saturated with vapor, the vapor density $\rho_{v,sat}$ in the gas phase is therefore determined by the local temperature applying the ideal gas law and the saturation pressure curve of water. ϵ_{α} and ϵ_{β} are the ice and moist air volume fractions, respectively. The driving potential for the diffusive water transport is the gradient of vapor partial pressure in the gas phase p_v , the resulting vapor flux for molecular diffusion is then obtained from [8]

$$\dot{m}_v = -\frac{D_{eff}}{R_v T} \frac{\partial p_v}{\partial x} \quad (2)$$

with D_{eff} as the effective coefficient of diffusion and R_v as the ideal gas constant of water, T is the local temperature.

Equation 1 may then be transformed into

$$\begin{aligned} \frac{\partial \rho_{v,sat}}{\partial T} (1 - \epsilon_{\alpha}) \frac{\partial T}{\partial t} + (\rho_{\alpha} - \rho_{v,sat}) \frac{\partial \epsilon_{\alpha}}{\partial t} \\ = \frac{\partial}{\partial x} \left(\frac{D_{eff}}{R_v T} \frac{\partial p_v}{\partial x} \right) \end{aligned} \quad (3)$$

The energy balance over the frost element is

$$\frac{\partial u_f^*}{\partial t} = \frac{\partial}{\partial x} \left(\lambda_f \frac{\partial T}{\partial x} \right) - \frac{\partial \dot{m}_v}{\partial x} \Delta h_{sv} \quad (4)$$

The first term on the right handside of equation 4 corresponds to the conductive heat flow, the second represents the enthalpy transport associated with the diffusive vapor flux with Δh_{sv} as the heat of fusion. The frost energy per unit volume u^* is given by

$$u_f^* = c_f \rho_f T + (1 - \epsilon_{\alpha}) \rho_{v,sat} \Delta h_{sv} \quad (5)$$

where c_f is the specific heat capacity of frost, ρ_f is the frost density and T is the local temperature.

A simplifying assumption to the problem is the neglect of water vapor diffusing into the frost layer. The

energy balance then reduces to the basic formulation of unsteady heat conduction [8].

$$\frac{\partial(\rho_f c_f T)}{\partial t} = \frac{\partial}{\partial x} \left(\lambda_f \frac{\partial T}{\partial x} \right) \quad (6)$$

2.3 Boundary conditions

2.3.1 Wall

The boundary conditions at the wall side of the frost layer are:

$$\begin{aligned} T(x = x_w) &= T_w \\ \dot{m}_v(x = x_w) &= 0. \end{aligned} \quad (7)$$

The index w denotes the position at the wall interface, \dot{m}_v is the water vapor flux.

2.3.2 Airside heat and mass transfer

The heat transferred from the airflow to the frost surface is expressed as the sum of sensible and latent heat flows.

$$\dot{q}_{\text{sens}} + \dot{q}_{\text{lat}} = \lambda_f(x_s) \frac{\partial T_s}{\partial x} \quad (8)$$

$$\alpha(T_b - T_s) + \Delta h_{sv}(\dot{m}_t - \dot{m}_{v,s}) = \lambda_f(x_s) \frac{\partial T_s}{\partial x} \quad (9)$$

where α is the coefficient of heat transfer, T_b is the bulk air temperature and the index s denotes surface properties. \dot{m}_t and $\dot{m}_{v,s}$ are the total vapor flux from the air to the frost layer and the vapor flux into the frost layer, respectively. They may be determined from

$$\dot{m}_{v,s} = \dot{m}_v(x = x_s) = \left(\frac{D_{\text{eff}}}{R_v T} \right)_s \frac{\partial p_{v,s}}{\partial x} \quad (10)$$

$$\dot{m}_t = \rho_a \beta (X_b - X_s) \quad (11)$$

where β is the mass transfer coefficient in m/s and X is the absolute humidity in in the bulk flow and at the frost surface, respectively and ρ_a is the air density. Subtracting the two fluxes yields the frost deposition rate at the frost surface.

$$\rho_{f,s} \frac{\partial x_s}{\partial t} = \dot{m}_t - \dot{m}_{v,s} \quad (12)$$

The surface density may be calculated from an empirical correlation by Hayashi *et al.* [9]

$$\rho_{f,s} = 650 e^{0.227(T_s - 273.15)} \quad (13)$$

where T_s is the surface temperature in K. In [6] a zero gradient for the ice volume fraction is assumed at the surface.

$$\frac{\partial \epsilon_{\alpha,s}}{\partial x} = 0; \quad (14)$$

If a simplified model is used that does not account for water diffusion into the frost, $\dot{m}_{v,s}$ is set to zero and the entire water mass flow towards the frost surface is assumed to contribute to the increase of frost layer height.

Empirical correlations are used to determine the transport coefficients. In the literature the Chilton-Colburn analogy between heat and mass transfer seems to be widely accepted also under frosting conditions [2] [11]. It relates the mass transfer coefficient β to the heat transfer coefficient α using the Lewis number Le :

$$\beta = \frac{\alpha}{c_p \rho_a} Le^{2/3} \quad (15)$$

ρ_a is the moist air density and c_p is the specific heat capacity. According to [2] the heat transfer coefficients are normally higher under frosting conditions compared to a smooth surface due to an increased surface roughness. However, it is unclear if the reduction of the hydraulic diameter also adds to this reported increase. The following correlation suggested by [2] for air flow between parallel plates for Reynolds number ranging from 6000 to 50000 was used:

$$Nu_f = 0.034 Re^{0.8} \quad (16)$$

where the Reynolds number is based on the hydraulic diameter of the parallel plate channel.

2.4 Frost properties

As described above the prediction of frost layer thickness and its thermal behavior largely depends on frost properties such as density, thermal conductivity and the effective diffusion coefficient of water vapor in the air filled porous medium. The diversity of possible frost structures as indicated in section 2.1 clearly shows that an accurate determination of those properties for a wide range of operating conditions and heat exchanger types may be a difficult task. The wide range of empirical correlations and theoretical models that have been established sometimes are mostly applicable only in a very limited operating range. A literature review of empirical correlations for frost properties and frost layer thickness can be found e.g. in [2].

2.4.1 Thermal conductivity

The significant thermal resistance added to the heat exchanger wall by the growing frost layer results from the rather low thermal conductivity of the ice-air composite. The air in the pores largely contributes to the

insulating effect. Also the structure and orientation of the ice crystals influence the material property. There is an agreement in the literature that thermal conductivity is a strong function of frost density with only a weak dependence on temperature. An empirical correlation based on the frost layer density from [10] is used for the proposed models:

$$\lambda_f = 0.02442 + 7.214 \cdot 10^{-4} \rho_f + 1.1797 \cdot 10^{-6} \rho_f^2. \quad (17)$$

Another correlation suggested by [2] is

$$\lambda_f = 1.202 \cdot 10^{-3} \rho_f^{0.963} \quad (18)$$

More theoretical approaches take into account the (generally assumed) frost structure and sometimes even the surface roughness as in [11]. All correlations are found between the two ideal assumptions of ice crystals parallel and perpendicular to the prevailing direction of heat transport. The first orientation corresponds to thermal resistances in series, the second to a parallel arrangement. Their conductivities can be obtained as follows

$$\begin{aligned} 1/\lambda_{\text{per}} &= (1 - \varepsilon_\alpha)/\lambda_a + \varepsilon_\alpha/\lambda_\alpha \quad (\text{perpendicular}) \\ \lambda_{\text{par}} &= (1 - \varepsilon_\alpha)\lambda_a + \varepsilon_\alpha\lambda_\alpha \quad (\text{parallel}) \end{aligned} \quad (19)$$

where λ_a and λ_α are the thermal conductivities of air and ice, respectively. A comparison of this approach with eq. (17) and (18) is given in figure 2. It shows that the presented empirical correlations are found within the two ideal structure models, but still reveal significant deviations in one to another. It is expected that differences in frost structure which cannot be related to density or ice volume fraction alone that are caused by variable frost growth conditions and aging effects also play an important role.

2.4.2 Density and specific heat capacity

The way to determine the frost density depends on the assumptions made for the vapor transport in section 2.2. If equation 1 is used, the density distribution in the frost layer can be determined from the ice volume fraction ε_α

$$\rho_f = \varepsilon_\alpha \rho_\alpha + (1 - \varepsilon_\alpha)(\rho_a + \rho_v) \quad (20)$$

$$\approx \varepsilon_\alpha \rho_\alpha \quad (21)$$

The volumetric heat capacity may be computed in a similar way and with neglect of the gas phase impact reduces to:

$$\rho_f c_f \approx \varepsilon_\alpha \rho_\alpha c_\alpha \quad (22)$$

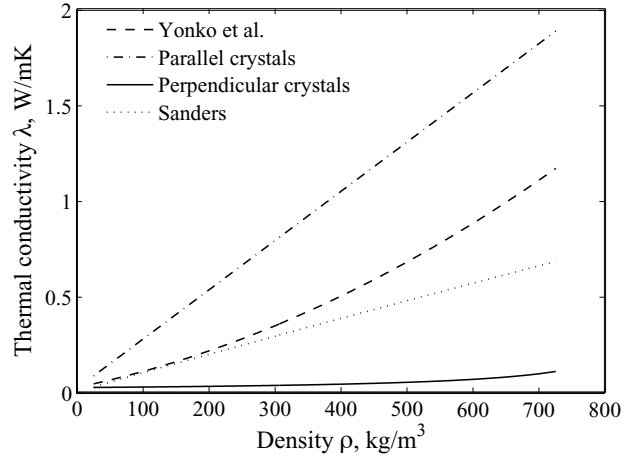


Figure 2: Thermal conductivity vs. frost density for the empirical correlation by Yonko et al. (17), the correlation by Sanders (18) as well as perpendicular and parallel ice crystal orientation, respectively.

If the frost layer is assumed to have a uniform density over the entire frost height and equation 6 is used, the average frost density $\bar{\rho}_f$ is computed from a mass balance over the total frost layer,

$$\frac{\partial(\bar{\rho}_f \delta_f)}{\partial t} = \dot{m}_t \quad (23)$$

with \dot{m}_t from eq. (10) and the frost thickness δ_f .

2.4.3 Coefficient of diffusion

Following from (10) the frost density largely depends on the water vapor that diffuses into the frost layer. The effective diffusion coefficient D_{eff} in (3) accounts for several complex mechanisms. Those involve besides molecular diffusion the tortuosity of the porous frost structure, the variation of pore diameters, phase transitions and others. The effective diffusion coefficient may be expressed in terms of the binary diffusion coefficient D_{AB}

$$D_{\text{eff}} = \mu D_{AB} \quad (24)$$

Several works focus on the determination of the μ factor with significantly different results. The first models dealing with effective diffusion factors for frost always assumed lower values than those for molecular diffusion ($\mu \leq 1$), others including Tao *et al.* in [6] report values several times larger. Le Gall *et al.* propose a correlation that combines both approaches and try to determine the factor F in the following equation empirically with respect to selected boundary conditions

[5].

$$\mu = \frac{1 - \varepsilon_\alpha}{1 - 0.58 \varepsilon_\alpha} + 10 F \varepsilon_\alpha (1 - \varepsilon_\alpha)^{10} \quad (25)$$

F will also be treated as a constant parameter input to (25) in the distributed model and discussed further below.

3 Implementation of the model in Modelica

A Modelica model is developed to investigate the impact of frost growth on parallel plates passed by a moist airflow. A model structure that allows a flexible and easy change of the level of detail and used correlations is aimed for.

The transport phenomena involved can be divided into groups with respect to their location in the combined problem.

- heat and mass transport by forced convection - spatial resolution in air flow direction required
- heat and mass transport through thermal and concentration boundary layers - transport phenomena are lumped by using transport coefficients from empirical correlations
- conductive heat and diffusive mass transport through the frost layer - spatial resolution perpendicular to airflow direction required
- conductive heat transport through the solid wall - spatial resolution perpendicular to airflow required

The first two are combined in an air flow model partly taken from the Modelica_Fluid library, frost layer and wall are combined in a second component for the reasons specified further below.

3.1 Airflow model

The model that describes the moist airflow in a channel formed by two flat plates is extended from the base class `PartialDistributedFlow` in the *Modelica_Fluid* library. Conservation of energy, total mass, substance mass and momentum as well as flow reversal are handled in this model applying a one-dimensional finite volume approach with upwind discretization. The medium model for moist air was taken from the *Modelica.Media* library. The energy and

mass balances contain source (or sink) terms which can be used to include heat and massflow across the volume boundaries. Those heat and mass flow rates are determined in a replaceable subcomponent `heat` which is added to the extended air model. The total volume and hydraulic diameter depend on the internal dimensions of the flow channel, which may change with an increasing frost layer.

3.2 Wall and frost model

3.2.1 Implemented frost models

Three frost models are implemented in Modelica reflecting different levels of detail.

- I Distributed frost densities (ice volume fractions), diffusive vapor transport into frost. Balance equations (3) and (4), boundary conditions (7) - (12),(14) and property calculations (19), (23), (24) and (25) are used.
- II Frost density changes only due to temperature dependent surface density during frost deposition. Water vapor diffusion in the frost layer is neglected. Balance equations (6) and (23), boundary conditions (7) - (13) and property correlation (17) are used.
- III Steady state heat transport. The same equations as in model II are used, the energy storage term in eqn. (6) is set to zero.

3.2.2 Spatial resolution of transport equations

The spatial resolution of the partial differential equations of heat and mass transport are implemented in model I applying a finite volume method [12] for a flat plate geometry. The number of nodes is kept constant throughout the computational procedure, the moving grid therefore requires the inclusion of convective terms in the heat and mass balances which are expressed using an upwind scheme. A spatial discretization of 10 segments seems to give a satisfactory accuracy and was used in all results presented further below. The frost properties on the right hand side of equations (4) and (3) representing the inverse of the transport resistances are then defined at the boundary of two neighboring finite volumes. In case they are not constant in the spatial domain, they are determined using the harmonic mean of the terms computed for the two neighboring cells. This corresponds to finite resistances connected in series.

Combining wall and frost layer in one component is advantageous if the model should also be capable of simulating situations, when no frost layer is present due to a surface temperature higher than the triple or the dew point. In this case the proposed equation system would become singular. Including the thermal resistance of the frost in the energy balance of the first wall element until a certain frost thickness is reached makes it possible to determine the onset of frost formation by boundary conditions. The internal temperatures are set to constant during this period and are reinitialized using the `reinit` operator as soon as a threshold value for the frost layer thickness is reached. A linear distribution of the difference between wall and surface temperatures is used as start values. This approach simply neglects the energy storage capacity of the frost layer in the initial growth period. The frost porosity also remains constant during this time. The initial frost density is assumed to be $\bar{\rho}_{f,0} = 25 \text{ kg/m}^3$ as suggested by [5].

Both, wall and frost layer are also discretized in air flow direction without any interdependence of neighboring cells in this dimension.

3.3 Air - frost/wall interface

When the surface temperature is above the triple point and below the dew point of the air flow, water condenses at the cold surface and is assumed to leave the system immediately. The liquid water volume is therefore not considered. The model does not include situations of water condensation on a present frost surface. The liquid would soak into the porous medium, its distribution would be very hard to predict. The connector variables required to combine the two components, airflow and frosted wall, are:

```
connector FrostPort
  SI.HeatFlowRate Q_flow "Sensible heat flow";
  SI.Temperature T "Surface temperature";
  SI.MassFlowRate m_flow "Water mass flow rate (condensing or freezing)";
  SI.MassFraction xs "Saturated absolute humidity at surface";
  SI.EnthalpyFlowRate H_flow "Latent heat flow rate";
  SI.SpecificEnthalpy h_gas "Specific enthalpy of vapor in bulk flow";
  SI.Velocity xflow "Frost layer growth rate";
  SI.Length x_f "Frost layer thickness";
end FrostPort;
```

The frost layer model versions I-III are organized as replaceable components in the combined airflow/wall model. They extend from a base class which contains the parts all versions have in common, like e.g. the interface and some boundary conditions. All frost property correlations are written as functions.

4 Results and Discussion

4.1 Lumped frost layer model

Figure 3 shows the average values for frost thickness, frost density and surface temperature with respect to time and different air inlet velocities for model I. A discretization of 8 segments was chosen for the air flow dimension. If the timescale of frost growth is of primary interest the steady state model (model III) gives results similar to model II and even on a shorter timescale the heat capacity of the frost layer may be negligible. However, introducing additional numerical states may be advantageous in a more complex application to break down systems of nonlinear equations.

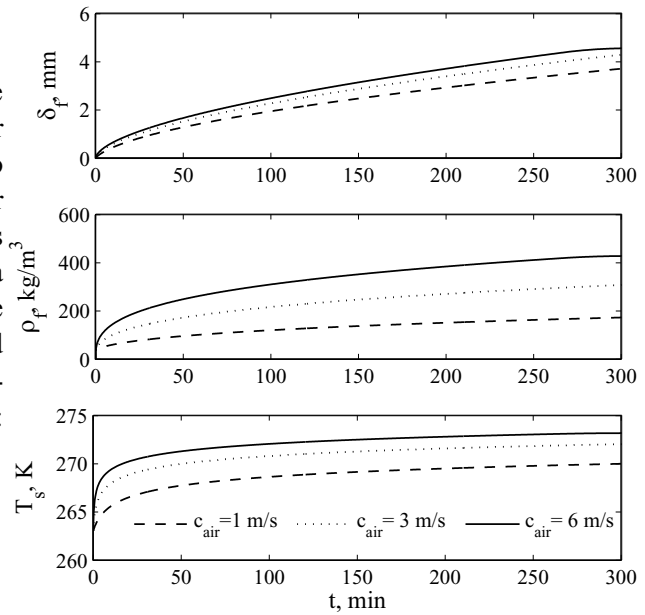


Figure 3: Frost thickness δ_f , mean frost density $\bar{\rho}_f$ and surface temperature T_s at different air inlet velocities. $T_b = 293 \text{ K}$, $A_{\text{channel},0} = 40 \text{ cm}^2$, $T_w = 263 \text{ K}$, $\phi_{\text{air}} = 42 \%$.

An increasing air flow rate leads to a higher frost deposition rate due to an increased heat and mass transfer. This creates a higher thermal resistance leading to a greater surface temperature. At the same time the density of the frost layer increases with a rising sur-

face temperature according to the used correlation for the surface density (eq. 13) which in turn reduces the thermal resistance of the total frost layer. The total thermal resistance responsible for the thermal behavior of the frosted heat exchanger surface is therefore a result of the counteracting effects of densification and frost growth.

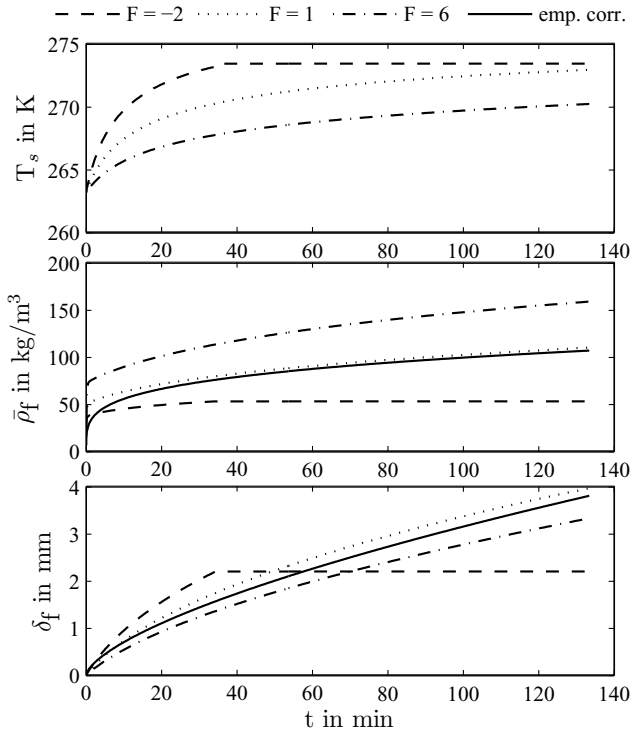


Figure 4: Surface temperature T_s , frost density $\bar{\rho}_f$ and frost layer thickness δ_f at a distance of $y = 0.5$ m from the leading edge over time with the diffusion enhancement factor F as a parameter. Boundary conditions: $v_{\text{air}} = 2.5$ m/s, $T_{\text{air}} = 283$ K, $T_w = 263$ K, $X = 0.006$ kg/kg.

4.2 Distributed frost layer model

The distributed frost layer model provides a spatial resolution of frost density with frost layer height which makes it possible to compute the local thermal resistance in the frost. However, because measurement data of these variations are scarce, the overall values of frost thickness and average frost density will be further discussed. Mao *et al.* conducted a wide range of measurements under various conditions with an air-flow channel (300 mm wide and 20 mm high) that contained one cooled surface. They derived empirical correlations from their results (with a root mean square error of 0.22 for density and frost height), which are

used in this work to adapt the diffusion enhancement factor F in (25). Figure 4 presents simulation results from the distributed model for different values of F . It can be seen that frost growth and density development are strongly influenced by its value. Unfortunately Mao *et al.* did not measure the frost surface temperature, but assumed it always to be equal to 0 °C, while the model presented here does not account for melting water permeating the frost which would occur under those circumstances. Instead the frost layer growth stops as soon as the triple point temperature is reached as can be observed at $t = 35$ min for $F = -2$ caused by the low water diffusion resulting in a larger frost thickness, higher porosity and large thermal resistance of the material.

4.3 Model comparison

A comparison of model I (distributed) and model II (lumped) is given together with results from empirical correlations found by Mao *et al.* for their experimental data in figure 5 for different wall surface temperatures. The frost thickness computed with the lumped model is in good agreement with the values obtained from the empirical correlations. The average density is larger than that computed using the empirical correlation in all lumped model cases. This would also lead to a thermal resistance predicted too low. However, the deviation from the given error interval of the empirical correlation is not very large. The distributed model shows a close agreement with the empirical correlations for both, frost thickness and average density, the diffusion factor F was set to 0, 3 and 8, respectively with rising wall temperature to give a good fit. A trend towards an increasing enhancement of the diffusion factor with rising plate temperatures in a non-linear way is also reported by [6]. However, the exact values may not be transferable to different boundary conditions.

A similar trend on the average frost density can be observed with the lumped model which relates the surface density to the surface temperature. Since density has a major impact on all frost properties including the thermal conductivity its correct determination seems important. However, in both approaches some assumptions were made that strongly influence the development of frost density with time. In the distributed approach a zero density gradient is applied at the surface which is similar to the approach in [6], while correlation (13) is used for the lumped model. But it is believed that the actual structure and at the same time the density of the new frost depositing at the surface may actually differ from these assumptions.

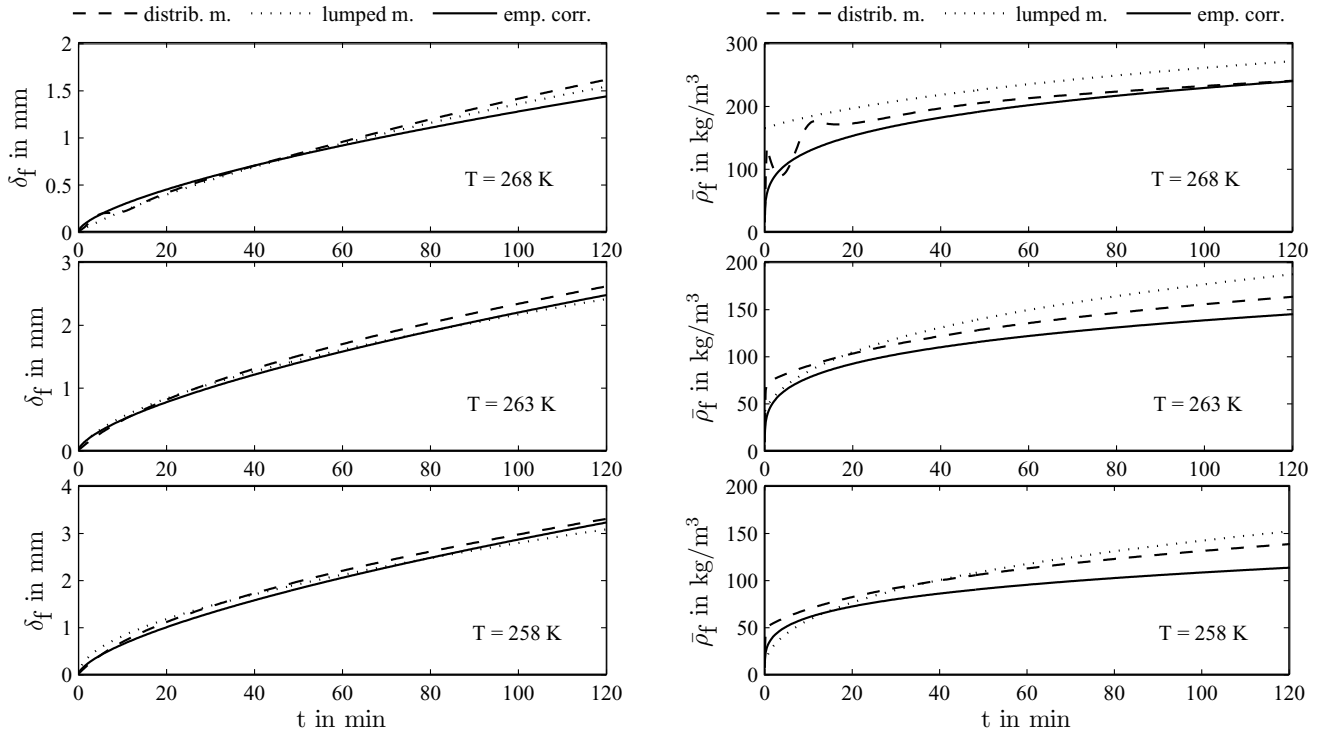


Figure 5: Frost layer thickness δ_f and average frost density $\bar{\rho}_f$ over time for the lumped and the distributed model and an empirical correlation from [13]. Boundary conditions: $v_{\text{air}} = 2.5$ m/s, $T_{\text{air}} = 291$ K, $X = 0.006$ kg/kg, $y = 0.5$ m.

5 Conclusion

Resolving heat and mass transport phenomena in the spatial domain of the frost layer as it is done in model I gives only higher accuracy in predicting frost growth and average density if information on the diffusion enhancement factor F is present. The simple lumped approach (model II) is suitable to predict trends of frost formation with respect to variable boundary conditions, such as air and wall temperature, humidity and air velocity. A surprisingly good agreement is obtained for the predicted frost thickness when compared to an empirical correlation. More accurate absolute values of density and thermal resistance with regard to operating time may perhaps be obtained with a refinement of the model in terms of densification of the frost layer with time and surface density of the deposit. In addition more empirical data is required, especially in terms of the frost surface temperature, to validate the model in a wide operating range.

Modeling the frost formation process correctly provides information on the amount of frost and its ice fraction at the start of the defrost process, which gives an idea of the minimum energy required for a complete defrost. Further work is needed to describe the defrost

process with time depending on the chosen strategy. The most common method of heating the solid wall material using hot gas or electrical current poses the problem of predicting the detachment of the frost layer from the wall and the associated heat transport through gaps filled with liquid water or air. In addition, heat transfer to the surrounding air by natural convection must be taken into account if the air supply is stopped during this period.

References

- [1] Xia, Y., Hrnjak, P. S. and Jacobi A. M, Air-side thermal hydraulic performance of louvered-fin, flat-tube heat exchangers with sequential frost-growth cycles. *ASHRAE Transactions*, 2005, **111**, 487-495.
- [2] O'Neal, D. L. and Tree D. R., A review of frost formation in simple geometries. *ASHRAE Transactions*, 1992, **98**(Part 2), 65-78.
- [3] Hoffenbecker, N., Klein, S.A. and Reindl, D.T., Hot gas defrost model development and validation. *International Journal of Refrigeration*, 2005, **28**, 605-615.

- [4] Sanders, C. T., Frost Formation: The influence of frost formation and defrosting on the performance of air coolers. PhD thesis, Delft University of Technology, Delft, 1974.
- [5] Le Gall, R., Grillot, J. M. and Jallut, C., Modelling of frost growth and densification. *International Journal of Heat and Mass Transfer*, 1997, **40**, 3177-3187.
- [6] Tao, Y. X., Besant, R., W. and Rezkallah, K. S. A mathematical model for predicting the densification and growth of frost on a flat plate. *International Journal of Heat and Mass Transfer*, 1993, **36**, 353-363.
- [7] Hayashi Y., Aoki, A., Adachi, S. and Hori, K., Study of frost properties correlating with frost formation types. *Jornal of Heat Transfer*, 1977, **99**, 239-245.
- [8] Baehr, H. D., Stephan, K., *Heat and Mass Transfer*. 2nd Ed., Springer Verlag, Berlin, 2006.
- [9] Hayashi Y., Aoki, A. and Yuhara, H., Study of frost formation based on a theoretical model of the frost layer. *Heat Transfer - Japanese Research*, 1977, **6**, 79-94.
- [10] Yonko, J. D. and Sepsy, C. F., An investigation of the thermal conductivity of frost while forming on a flat horizontal plate. *ASHRAE Transactions*, 1967, **73**, 1.1-1.11.
- [11] Yun, R., Kim, Y. and Min, M., Modeling of frost growth and frost properties with airflow over a flat plate. *International Journal of Refrigeration*, 2002, **25**, 362-371.
- [12] Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publ. Co., New York, 1980.
- [13] Mao, Y., Besant, R.W., Rezkalla, K.S., Measurement and correlations of frost properties with airflow over a flat plate. *ASHRAE Transactions*, 1992, **1**, 65-78.

Multizone Building Model for Thermal Building Simulation in Modelica

Michael Wetter
United Technologies Research Center
411 Silver Lane
East Hartford, CT 06108

Abstract

We present a room model for thermal building simulation that we implemented in Modelica. The room model can be used for controls analysis and energy analysis of one or several rooms that are connected through airflow or heat conduction. The room model can assess energy storage in the air and in the building construction materials, heat transfer between the room and the outside environment and the humidity and CO₂ release to the room air. The humidity storage in the building construction materials is not modeled. We also describe a novel separation of heat transfer mechanisms on which our room model is built on. The separation allowed a significant reduction in model development time, and it allows using state-of-the-art programs for computing prior to the thermal building simulation certain energy flows, such as solar heat gain of an active facade without breaking feedback loops between the HVAC system and the room.

Keywords: Multizone thermal building model, building energy analysis

1 Introduction

Many control design processes require frequency domain analysis, time domain simulations with time steps in the order of seconds or plant model inversion. Commercially available detailed building energy simulation programs, such as EnergyPlus [2], TRNSYS [8], and DOE-2 [20], are not applicable for such analysis. For example, the numerical solution algorithms used in the building envelope model of EnergyPlus, TRNSYS and DOE-2, are all based on a discrete time representation of the building envelope dynamics that does not allow time steps in the order of seconds. In addition, none of the above cited building energy simulation programs can be used as a plant model in a frequency domain controls analysis because they

cannot be integrated into MATLAB/Simulink nor can model that are defined in those programs be automatically inverted. Thus, we developed a physics-based room model in Modelica to allow advanced control design. For the model to be applicable to analyze a large class of problems, we developed it such that it satisfies the following requirements:

1. To be applicable in trade-studies, it is parameterizable with building geometry and material properties.
2. To be applicable for controls design, it captures the building dynamics and the coupling between rooms through air flow and heat conduction in solids.
3. To allow analyzing novel HVAC systems, such as hydronic concrete core cooling systems [9], it allows linking models for HVAC systems including radiant heating and cooling systems to the building.

Developing a detailed thermal building simulation model may take several months if not a year of development time. We therefore identified a model separation approach that allows us to develop a physics-based room model in Modelica at reduced model development time. Our modeling separation approach essentially separates those phenomena that are described by short-wave radiative heat exchange¹ from those phenomena that are described by long-wave radiative heat exchange, heat conduction and heat convection. To describe the state variables of the building envelope and of the room air temperature, the room model uses as input time-series of short-wave heat gains. The short-wave heat gains are *independent* of the state variables, control actions and HVAC

¹By short-wave radiation, we mean radiation whose wave length is in the solar spectrum.

operation, and computing those heat gains requires models that are time-intensive to develop. Since those time-series are independent on the state variables and control actions, they can be precomputed by detailed thermal building simulation programs, such as IDA [14], TRNSYS or EnergyPlus and used as time-dependent disturbances to our room model.

In Section 2, we describe the model separation approach, in Section 3 we describe the equations used to construct the room model and in Section 4 we discuss the implementation in Modelica. A comparative model validation and a comparison between the model implementation in Modelica with a TRNSYS model of comparable complexity can be found in [16].

2 Model Separation Approach

To explain the model separation approach, we will now identify the dominant modes of heat transfer that need to be described adequately to quantify a building's energy consumption and the occupant's thermal comfort [5]. Figure 1 orders the modes of heat transfer on the horizontal axis by the time required to code and test a simulation model that describes the phenomena and on the vertical axis by the uncertainty associated with the modeling assumptions and/or with the model parameters that one typically encounters for an office building. While the exact location of each phenomenon model can be debated, the figure indicates the following characteristics:

- Models with large uncertainty in the modeling assumptions and/or in the model parameters require a short development time.
- Models with a small uncertainty but large development time are those models that describe heat transfer due to short-wave radiation.

Since the short-wave radiation is *independent* on the building's state variables and control actions², the models on the right hand-side of the dashed line in Figure 1 can be computed in a detailed thermal building simulation program, such as EnergyPlus, and their output be used as forcing functions of the Modelica model in the form of time series. Since short-wave radiative heat gains are decoupled from the room air

²An exception could be, for example, if the building occupants close a window blind if the room air temperature gets uncomfortably warm. Our room model could, however, be extended to describe such situations.

and the room's enclosure surface temperatures, there is no loss of accuracy in decoupling the short-wave radiation from the heat transfer by long-wave radiation, conduction and convection.

Furthermore, this model separation does not restrict analysis capabilities such as controls design and system optimization, as the following items illustrate.

- Since all models that depend on the state variables and the control actions are modeled in Modelica, for controls design MATLAB/Simulink need only be interfaced with Modelica but not with a building simulation program.
- Since the HVAC system as well as the building envelope is modeled in Modelica, cost function evaluations during a system optimization does not in general require running a detailed building simulation program. This considerably simplifies the optimization since the numerical noise of state variables with respect to system parameters is hard, if not impossible, to control in most thermal building simulation programs. Numerical noise has been shown to cause optimization algorithms to fail finding an optimal solution in building energy optimization studies [18, 19, 15].

3 Model Description

We will now present the implemented model equations in order to understand what physical phenomena can be analyzed with the room model, and what physical phenomena, such as room air stratification of displacement ventilation, require an extension of the here presented room model in order to be described adequately.

3.1 Distribution of a Room's Solar Gains

To compute the solar gains per unit area for the room surfaces, we assume that all solar radiation that enters the room first hits the floor, and that the floor diffusely reflects the radiation to all other surfaces. We neglect multiple reflections and instead of using view factors between the floor and the other surfaces, we use area-weighted solar distribution factors. Such an approach is common in detailed building simulation programs and has been successfully validated in [15].

Consider a room with $n_f \in \mathbb{N}$ floor area patches, $n_{nf} \in \mathbb{N}$ non-floor area patches, such as walls and windows, and $n_w \in \mathbb{N}$ windows. Let $\mathbf{N}_f \triangleq \{1, \dots, n_f\}$ be the

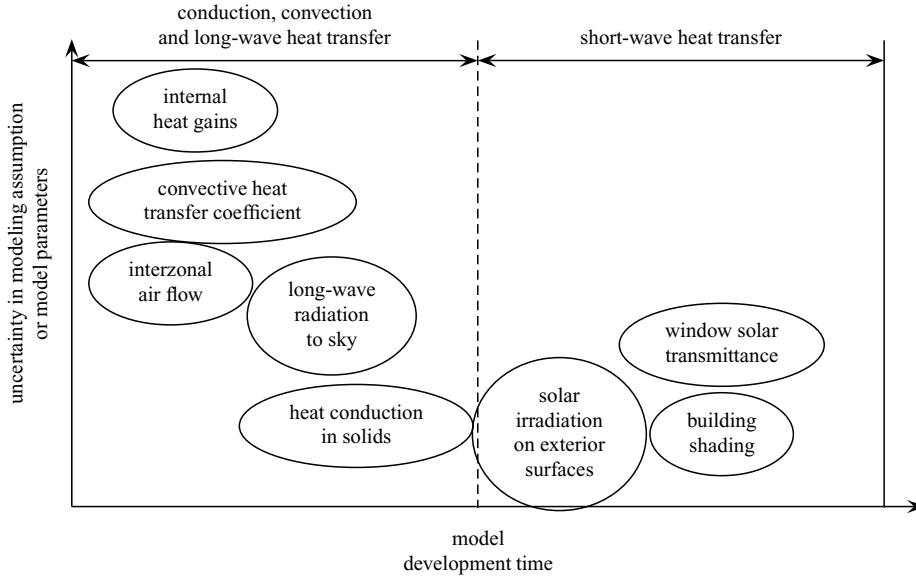


Figure 1: Dominant modes of heat transfer for a thermal building simulation.

set of indices of all floor area patches, let $\mathbf{N}_{nf} \triangleq \{1, \dots, n_{nf}\}$ be the index set of all non-floor surface patches and let $\mathbf{N}_w \triangleq \{1, \dots, n_w\}$ be the set of indices of all windows. Let the solar radiation transmitted from the exterior to the room through all windows be

$$H_{tot}(t) \triangleq \sum_{i \in \mathbf{N}_w} H^i(t) A^i, \quad (1)$$

where $H^i(t)$ denotes the solar radiation that is transmitted per unit area through the i -th window and A^i is the glazed area of the i -th window. The solar radiation that is absorbed by the k -th floor area patch per unit area is

$$q_{SW,abs,0}^k(t) = \frac{\epsilon_{SW}^k}{\sum_{n \in \mathbf{N}_f} A^n} H_{tot}(t), \quad k \in \mathbf{N}_f. \quad (2)$$

The solar radiation that is reflected from the k -th floor area patch is

$$Q_r^k(t) = \frac{A^k (1 - \epsilon_{SW}^k)}{\sum_{n \in \mathbf{N}_f} A^n} H_{tot}(t), \quad k \in \mathbf{N}_f. \quad (3)$$

Therefore, the solar radiation that is reflected by the whole floor area is

$$Q_r(t) = \sum_{n \in \mathbf{N}_f} Q_r^n(t) = \frac{\sum_{n \in \mathbf{N}_f} A^n (1 - \epsilon_{SW}^n)}{\sum_{n \in \mathbf{N}_f} A^n} H_{tot}(t). \quad (4)$$

We distribute $Q_r(t)$ to all non-floor areas, weighted by their solar absorptivity ϵ_{SW} and solar transmissivity for diffuse irradiation τ_{SW} (which is non-zero for windows). We compute the solar radiation that is absorbed

or transmitted by the k -th non-floor surface patch per unit area as

$$q^k(t) = Q_r(t) \frac{\epsilon_{SW}^k + \tau_{SW}^k}{\sum_{m \in \mathbf{N}_{nf}} A^m (\epsilon_{SW}^m + \tau_{SW}^m)}, \quad k \in \mathbf{N}_{nf}. \quad (5)$$

For opaque constructions, $q^k(t)$ is assumed to be completely absorbed and hence $q_{SW,abs,0}^k(t) = q^k(t)$. For windows, however, the absorbed radiation at the room-side pane is

$$q_{SW,abs,0,r}^k(t) = \epsilon_{SW}^k q^k(t), \quad (6)$$

and the remaining fraction of $q^k(t)$ is assumed to be transmitted to the exterior.³

3.2 Distribution of Internal Heat Gains

The radiative and sensible convective and latent heat gains caused by appliances, such as computers, and by people are defined by time series and are input to this model. For values between the support points of the time series, we use an interpolation to define the heat gains as a Lipschitz continuously differentiable function of time.⁴

The radiative internal heat gains are distributed to the room enclosing surfaces as follows: For some $n \in \mathbf{N}$,

³A fraction of $(1 - \epsilon_{SW}^k) q^k(t)$ would be absorbed by the exterior pane, but we neglect this effect.

⁴Using smooth functions to describe the heat gains can significantly reduce the computation time since smoothing eliminates time events.

let $\mathbf{N} \triangleq \{1, \dots, n\}$ denote the index set of all room surfaces. For radiative internal heat gains, the i -th surface is assumed to absorb the heat gain per unit area

$$q_{LW,abs,0}^i(t) = \frac{\epsilon_{LW}^i}{\sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k} Q_{int,LW}(t), \quad (7)$$

where ϵ_{LW} denotes the absorptivity for long-wave radiation, and $Q_{int,LW}(t)$ denotes the radiative internal heat gain of the zone. Since the long-wave emissivity of window glass is close to unity, we consider windows to be opaque in the long-wave radiation spectrum.

3.3 Humidity Gains

The humidity gain $\dot{m}_{H_2O}(t)$ in the room is computed as

$$\dot{m}_{H_2O}(t) = \frac{Q_{lat}(t)}{r_0 + c_p T}, \quad (8)$$

where $Q_{lat}(t)$ is the room's latent heat gain, $r_0 = 2501.6 \cdot 10^3 \text{ J/kg}$ is the enthalpy of evaporation of water vapor at 0°C , $c_p = 1860 \text{ J/(kg K)}$ is the specific heat capacity of water vapor and T is the temperature at which the water vapor is released. We assume that the temperature at which the water vapor is released is the average body surface temperature of a human, which is 34°C [1, p 8.1].

3.4 CO₂ Gains

Based on the room's latent heat gains, we compute the number of room occupants in the room and then, using the CO₂ release per person, we compute the CO₂ release in the room. Table 3.4 is a partial list of latent heat gains of occupants of conditioned spaces, adopted from the ASHRAE Fundamentals [1, p 28.8]. The number of people $n_p(t)$ is computed as

$$n_p(t) = \frac{Q_{lat}(t)}{q_{lat}}, \quad (9)$$

where $Q_{lat}(t)$ is the room's latent heat gain and q_{lat} is the latent heat gain per person, which is a user specified parameter and can be obtained from Table 3.4. Table 3.4 is adopted from [10] and specifies the CO₂ emissions of a person at different degrees of activity. Using the emitted mass flow of CO₂ per person and the number of occupants in the room as computed in (9), we compute the total CO₂ source in the room as

$$\dot{m}_{CO_2}(t) = n_p(t) \dot{m}'_{CO_2}. \quad (10)$$

3.5 Radiative Heat Exchange between Room Surfaces

Let $\mathbf{N} \triangleq \{1, \dots, n\}$ be as above. To compute the radiative heat transfer between the room surfaces, we define a radiation temperature $T^*(t)$ as

$$T^*(t) \triangleq \frac{\sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k T_{sur}^k(t)}{\sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k}, \quad (11)$$

where $T_{sur}^k(t)$ is the surface temperature of the k -th surface patch. We assume that each surface only exchanges radiation with an imaginary surface of much larger area which is at temperature $T^*(t)$. The radiative heat transfer to the surface $k \in \mathbf{N}$ can be described by

$$q_{LW}^k(t) = \sigma \epsilon_{LW}^k (T^*(t)^4 - T^k(t)^4), \quad (12)$$

where $\sigma = 5.670 \cdot 10^{-8} \text{ J/(K}^4 \text{ m}^2 \text{ s)}$ is the Stefan-Boltzmann constant. To reduce computation time, we linearize the nonlinear equation (12) around $T_0 = 273.15 \text{ K}$. By introducing a linearized heat transfer coefficient, defined as

$$h_{LW}^k \triangleq 4\sigma \epsilon_{LW}^k T_0^3, \quad (13)$$

we can write

$$q_{LW}^k(t) = h_{LW}^k (T^*(t) - T^k(t)). \quad (14)$$

Equations (14) and (11) are consistent with conservation of energy because

$$\begin{aligned} \sum_{k \in \mathbf{N}} A^k q_{LW}^k(t) &= 4\sigma T_0^3 \sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k (T^*(t) - T^k(t)) \\ &= 4\sigma T_0^3 \left(\sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k T^*(t) - \sum_{k \in \mathbf{N}} \epsilon_{LW}^k A^k T^k(t) \right) \\ &= 0. \end{aligned} \quad (15)$$

3.6 Room Air

We assume the room air to be completely mixed. This is a common model assumption in thermal building simulation programs and is used, for example, in [2, 8, 14, 20]. If the model were to be used for displacement ventilation, it would have to be modified to take into account the air stratification [13]. The time rate of change of the room air temperature is

$$\begin{aligned} m_r c_p \frac{dT_{air}(t)}{dt} &= Q_{int,con}(t) + Q_{aux,con}(t) \\ &+ \sum_{i=1}^m \dot{m}^i(t) c_p (T^i(t) - T_{air}(t)) \\ &+ \sum_{i=1}^n A^i h_{con,0}^i (T_{sur}^i(t) - T_{air}(t)), \end{aligned} \quad (16)$$

Table 1: Latent heat gains from occupants of conditioned spaces. Partial list adopted from ASHRAE Fundamentals [1, p 28.8].

Degree of activity	Typical usage	Latent heat q_{lat} in [W/person]
Seated, very light work	Offices, hotels, apartments	45
Moderately active office work	Offices, hotels, apartments	55
Standing, light work; walking	Department store; retail store	55
Walking, standing	Drug store; bank	70

Table 2: CO₂ emissions per person at different degrees of activity. Partial list adopted from [10].

Degree of activity	CO ₂ emissions \dot{V}'_{CO_2} in [dm ³ /h/person]	CO ₂ emissions \dot{m}'_{CO_2} in [kg/s/person]
Resting	12	0.00582
Sedentary, such as reading or writing	15	0.00728
Light work, standing, such as laboratory work, typesetting	23	0.0112

where m_r is the mass of the room volume, c_p is the air's specific heat capacity and $Q_{int,con}(t)$ is the internal convective heat gain. The term $Q_{aux,con}(t)$ represents convective heat gains caused by an auxiliary heating/cooling system that can be modeled outside of the room model, such as an electric baseboard heater. The term $\sum_{i=1}^m \dot{m}^i(t) c_p (T^i(t) - T_{air}(t))$ describes air flow into the room caused, for example, by the HVAC system and by interzonal air exchange. Any number of HVAC systems or interzonal air exchange models can be connected to the room. This makes the room model applicable for analyzing coupling between rooms due to static pressure difference or buoyancy driven air flow. The term $\sum_{i=1}^n A^i h_{con,0}^i (T_{sur}^i(t) - T_{air}(t))$ describes the convective heat transfer between the room air and the room's surface temperature. In (16), we assumed the convective heat transfer coefficient to be constant, but the model could be replaced by a model which computes the convective heat transfer coefficient as a function of the temperature difference or as a function of the air velocity near the surface.

3.7 Heat Conduction in Solids

Modeling the dynamics of the building envelope is important since the building envelope is typically a building's biggest thermal storage element which can be exploited for reducing cooling peak demand and energy.

The heat conduction in an opaque construction of thickness $L > 0$ over a time interval $(0, \tau)$ is computed using the Fourier equation

$$\frac{\partial}{\partial z} \left(k(z) \frac{\partial T_{sol}(z, t)}{\partial t} \right) = C(z) \frac{\partial T_{sol}(z, t)}{\partial t}, \quad (17)$$

on $(z, t) \in (0, L) \times (0, \tau)$, where $z \in (0, L)$ is the spatial coordinate, $k(z)$ is the thermal conductivity and $C(z) \triangleq \rho(z) c(z)$ is the thermal heat capacity per unit volume. For composite constructions, the material properties $k(\cdot)$ and $C(\cdot)$ typically vary from one layer to another and hence are step functions of the spatial coordinate z .

The boundary conditions are at the room-side surface (at $z = 0$)

$$\begin{aligned} -k(0) \frac{\partial T_{sol}(0, t)}{\partial z} &= h_{con,0} (T_{air}^0(t) - T_{sol}(0, t)) \\ &\quad + q_{SW,abs,0}(t) + q_{LW,abs,0}(t) \end{aligned} \quad (18)$$

and at the exterior surface (at $z = L$)

$$\begin{aligned} k(L) \frac{\partial T_{sol}(L, t)}{\partial z} &= h_{con,1} (T_{air}^1(t) - T_{sol}(L, t)) \\ &\quad + q_{SW,abs,1}(t) + q_{LW,abs,1}(t), \end{aligned} \quad (19)$$

where $h_{con,0}$ and $h_{con,1}$ are the convective heat transfer coefficients, $T_{air}^0(t)$ and $T_{air}^1(t)$ are the air temperatures

near the two surfaces, $q_{SW,abs,0}(t)$ and $q_{SW,abs,1}(t)$ are the short-wave radiation absorbed by the room-side and exterior surface, and $q_{LW,abs,0}(t)$ and $q_{LW,abs,1}(t)$ are the long-wave radiation absorbed by the room-side and exterior surface.

To compute a numerical approximation to the solution of (17), we spatially discretize (17) using a finite difference scheme.⁵ In composite constructions, the material properties of adjacent layers, say in material 1 and material 2, can be so that $k_1/k_2 \approx 10^{-1}$, $C_1/C_2 \approx 10^2$, and hence the ratio of the thermal diffusivities $\alpha \triangleq k/C$ can be $\alpha_1/\alpha_2 \approx 10^1$. Thus, the thermal diffusivity in adjacent layers can vary by an order of magnitude. Hence, if the spatial grid generation does not account for the material properties, then the time rate of change of the different nodes can be significantly different from each other, which can cause the system of ordinary differential equations to be stiff. Thus, we give the user the option to generate the spatial grid automatically so that under the assumption of equal heat transfer, each node temperature has a similar time rate of change. The automatic grid generation is done as follows.

From dimensionless analysis, one can obtain a characteristic time, called the *Fourier* number, as

$$Fo \triangleq \frac{\alpha t}{L^2}, \quad (20)$$

where α denotes the thermal diffusivity, t denotes time and L denotes the characteristic length [7]. It is favorable to generate the spatial grid so that the ratio (t/Fo) is equal to an arbitrary constant Π , which we define as

$$\Pi \triangleq \left(\frac{t}{Fo}\right)^{1/2} = \frac{L}{\sqrt{\alpha}}. \quad (21)$$

Now, let $K \in \mathbb{N}$ denote the number of material layers in an opaque composite construction, and let $\{l^k\}_{k=1}^K$ denote the thickness of each material layer. In view of (21), we compute the time constant of each material layer as

$$\Pi^k = \frac{l^k}{\sqrt{\alpha^k}}, \quad k \in \{1, \dots, K\}, \quad (22)$$

and we compute the estimated number of compart-

ments $\hat{N} \in \mathbb{R}$ for each layer as⁶

$$\hat{N}^k = N_{ref} \frac{\Pi^k}{\Pi_{ref}}, \quad (23)$$

where $N_{ref} \in \mathbb{N}$ is a user-specified number of compartments for a reference layer, which we define as a concrete layer with thickness $L_{ref} \triangleq 0.20$ m and thermal diffusivity $\alpha_{ref} \triangleq 3.64 \cdot 10^{-7}$ m²/s. Hence, $\Pi_{ref} \triangleq L_{ref}/\sqrt{\alpha_{ref}} = 331.4$ s^{1/2}. We compute the number of compartments for each material layer as $N^k = \lceil \hat{N}^k \rceil$, with $k \in \{1, \dots, K\}$, where the notation $\lceil \cdot \rceil$ is defined for $s \in \mathbb{R}$ as $\lceil s \rceil \triangleq \min\{k \in \mathbb{Z} \mid k \geq s\}$. Then, we divide each material layer $k \in \{1, \dots, K\}$ in compartments of length $\Delta^k \triangleq l^k/N^k$. Thus, the total number of compartments in the construction is $N = \sum_{k=1}^K N^k$.

3.8 Window Model

Detailed window models, such as the ones in EnergyPlus, consist of a model that computes the optical properties and a model that solves the heat balance equations. In the optical model, the solar transmittance, absorbance and reflectance of each window pane is computed first for direct irradiation at various incidence angles and then for hemispherical (diffuse) irradiation as if each pane would stand alone. Next, those data are used to compute the transmittance, absorbance and reflectance of each pane, but now taking into account multiple reflections between the individual panes and between the window and possible shading devices. Finally, those properties are passed from the optical to the heat balance model as a function of the solar incidence angle so that the heat balance model can compute the panes' temperature, long-wave radiative, conductive and convective heat transfer. Developing such models is time-intensive, see for example [15] for a detailed model description. We therefore implemented a simplified model for double pane windows that uses the solar radiation $H(t)$ that is transmitted per unit area through the window from the outside to the room as an input to a heat balance model. This transmitted solar radiation can be obtained from building energy simulation programs such as EnergyPlus in the form of a time series. The biggest heat gain of the room due to solar radiation is typically caused by the transmitted solar radiation, rather than by the solar radiation that is absorbed by the window and then conducted through the glass and convected to the room air. Therefore, we expect that the simplified model described below

⁵Since the coefficients of (17) are step functions, the partial differential equation has a weak but no classical solution for composite constructions. However, using a finite difference scheme is customary in thermal building simulation as it requires less development time than a finite element model.

⁶We use \hat{N} as a real number and will round later to an integer.

is accurate enough for many room configurations, except for rooms with a high ratio of glass to wall area.

We will use the same notation as in the Window 4 software manual [6]. In particular, “N,1” denotes that the radiation strikes the window from the outside pane “N” and is transmitted through the inside pane “1”, and “N,N” denotes that the radiation strikes the outside pane and is absorbed by the outside pane. The superscript “ \perp ” denotes irradiation normal to the window pane and the subscript “inc” denotes incident radiation. We use the following data as input to our optical model:

1. A time series, describing the transmitted solar radiation $H(t)$ per unit area.
2. The solar transmittance from the outside to the room for normal irradiation, defined as

$$T_{SW}^{N,1,\perp} \triangleq \frac{H_{\perp}^{\perp}}{H_{inc}^{\perp}}. \quad (24)$$

3. The solar absorbance for the room-side window pane for normal irradiation from the outside, defined as

$$A_{SW}^{N,1,\perp} \triangleq \frac{q_{SW,abs,0}^{\perp}}{H_{inc}^{\perp}}. \quad (25)$$

4. The solar absorbance for the exterior window pane for normal irradiation from the outside, defined as

$$A_{SW}^{N,N,\perp} \triangleq \frac{q_{SW,abs,1}^{\perp}}{H_{inc}^{\perp}}. \quad (26)$$

The transmittance and absorbance are functions of the incidence angle ϕ . We assume that the functional dependence of transmittance and absorbance is similar, i.e., we assume that there exists a function $g: [0, 90^\circ] \rightarrow \mathbb{R}$, such that for any $\phi \in [0, 90^\circ]$,

$$T_{SW}^{N,1}(\phi) = T_{SW}^{N,1,\perp} g(\phi), \quad (27)$$

$$A_{SW}^{N,1}(\phi) = A_{SW}^{N,1,\perp} g(\phi), \quad (28)$$

$$A_{SW}^{N,N}(\phi) = A_{SW}^{N,N,\perp} g(\phi). \quad (29)$$

Since the radiation that is absorbed in an infinitesimal glass element of thickness ds is proportional to the radiation that is transmitted through ds [12], assuming the existence of such a function $g(\cdot)$ is reasonable. Under this assumption, we can compute the absorbed radiation at the room-side window pane as

$$q_{SW,abs,0}(t) = \frac{H(t)}{T_{SW}^{N,1,\perp}} A_{SW}^{N,1,\perp} \quad (30)$$

and at the exterior window pane as

$$q_{SW,abs,1}(t) = \frac{H(t)}{T_{SW}^{N,1,\perp}} A_{SW}^{N,N,\perp}. \quad (31)$$

Note that (30) accounts only the absorption of the short-wave radiation from the outside to the room-side. The absorption of the short-wave radiation that is reflected by the room and radiated to the outside is described in (6).

In the thermal model, we neglect the window's thermal capacity since it is much smaller than the thermal capacity of the building envelope. This assumption is customary in thermal building simulation programs. The heat balance of the two glass layers is

$$\begin{aligned} 0 = & h_{con,0} (T_{air}(t) - T_{win}^1(t)) \\ & + q_{SW,abs,0}(t) + q_{SW,abs,0,r}(t) \\ & + q_{LW,abs,0}(t) + U (T_{win}^2(t) - T_{win}^1(t)) \end{aligned} \quad (32)$$

and

$$\begin{aligned} 0 = & h_{con,1} (T_{out}(t) - T_{win}^2(t)) + q_{SW,abs,1}(t) \\ & + q_{LW,abs,1}(t) + U (T_{win}^1(t) - T_{win}^2(t)), \end{aligned} \quad (33)$$

where $T_{win}^1(t)$ and $T_{win}^2(t)$ are the room side and exterior side glass temperatures, $q_{LW,abs,0}(t)$ and $q_{LW,abs,1}(t)$ are the long-wave radiation that is absorbed by the room side and the exterior side, $q_{SW,abs,0,r}(t)$ is as in (6) and U is the window gap's heat transfer coefficient. We assume U to be independent of the temperature difference $T_{win}^2(t) - T_{win}^1(t)$ since conduction through stagnant air is the dominant heat transport mechanism for air or argon gaps up to 10 mm thickness [3].

3.9 Long-wave Radiation between Exterior Surfaces and Environment

Exterior surfaces of opaque constructions and of windows participate in long-wave radiative heat exchange with the sky and the environment. The sky temperature $T_{sky}(t)$ is an input to the Modelica room model. We assume that the sky temperature is defined as the temperature that results in the same long-wave radiative heat exchange as if the sky would be a black body at temperature $T_{sky}(t)$ ⁷. We assume that the ground is at the temperature that is connected to the exterior heat flux port of the construction, which is typically the outside air temperature. The view factor φ_{s-g} between the

⁷The black body sky temperature can be obtained, for example, from EnergyPlus.

surface and the ground is assigned as

$$\varphi_{s-g} = \begin{cases} 1 & , \text{ for floors,} \\ 1/2 & , \text{ for walls,} \\ 0 & , \text{ for ceilings.} \end{cases} \quad (34)$$

Let $T_{ext}(t)$ denote the ground temperature. We define an environment temperature as

$$T_{env}(t) \triangleq (\varphi_{s-g} T_{ext}(t)^4 + (1 - \varphi_{s-g}) T_{sky}(t)^4)^{1/4}, \quad (35)$$

and we define a radiative heat transfer coefficient as

$$h_{LW}(t) \triangleq 4\sigma\epsilon_{LW} \left(\frac{T_{sur}(t) + T_{env}(t)}{2} \right)^3, \quad (36)$$

where $\sigma = 5.670 \cdot 10^{-8} \text{ W/m}^2/\text{K}^4$ is the Stefan-Boltzmann constant and $T_{sur}(t)$ is the exterior surface temperature of the construction. The long-wave radiative heat flux is computed as

$$q_{LW}(t) = h_{LW}(t) (T_{env}(t) - T_{sur}(t)). \quad (37)$$

4 Implementation in Modelica

We implemented the building model using Modelica 2.2 [11] and Modelica_Fluid [4]. Figure 2 shows the graphical view of the room model implementation in Dymola. There is a FluidPort (labeled “fluPor”) and a HeatPort (labeled “heaPor”) that connect to a JunctionVolume that models the room air (labeled “air”). The fluid port can be used to attach an airflow network that defines a model for an air-conditioning system or for interzonal air exchange [17]. The heat port can be used to add a convective heat source or sink to the room. There are models for interior surfaces “intSur” whose temperature and heat flow can be connected to an external model that may, for example, model heat transfer inside a radiant heating slab. The room-facing surface then automatically participates in the radiative heat transfer balance described in Section 3.5. The models “airCon” allows constructions to be modeled whose exterior surface have a prescribed air temperature as boundary condition, such as a floor above a crawl space. The models “parCon” describe the behavior of partition constructions, i.e., constructions where both surfaces are exposed to the same room. The models “extCon” and “winCon” describe exterior opaque constructions and windows. The models “intGai”, “qLatCon” and “intConGai” convert internal gains to heat, water vapor and carbon-dioxide flows. The models whose icons have a yellow bar are

replaceable placeholders for models that may be replaced at compile time with an array of models that have the actual physical implementation.

5 Conclusions

Our model separation approach led to shorter model development time and allows our model to be used with state-of-the-art software for daylighting simulation. The developed room model captures the dynamics required for controls design, and the energy flows required for building energy analysis simulations.

6 Acknowledgments

This research was supported by the U.S. Department of Commerce, National Institute of Standards and Technology, Advanced Technology Program under the agreement number 70NANB4H3024.

7 Nomenclature

7.1 Conventions

1. Vectors elements are denoted by superscripts.
2. $f(\cdot)$ denotes a function where (\cdot) stands for the undesignated variables. $f(x)$ denotes the value of $f(\cdot)$ for the argument x .

7.2 Symbols

$a \in A$	a is an element of A
\mathbb{N}	$\{1, 2, 3, \dots\}$
\triangleq	equal by definition

References

- [1] ASHRAE. *Fundamentals*. American Society of Heating, Refrigeration and Air-Conditioning Engineers, 1997.
- [2] Drury B. Crawley, Linda K. Lawrie, Frederick C. Winkelmann, Walter F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand, Richard J. Liesen, Daniel E. Fisher, Michael J. Witte, and Jason Glazer. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4):443–457, 2001.

- [14] Per Sahlin and Axel Bring. IDA solver – A tool for building and energy systems simulation. In J. A. Clarke, J. W. Mitchell, and R. C. Van de Perre, editors, *Proc. of the IBPSA Conference*, Nice, France, August 1991.
- [15] Michael Wetter. *Simulation-Based Building Energy Optimization*. PhD thesis, University of California at Berkeley, 2004.
- [16] Michael Wetter. Modelica versus TRNSYS – a comparison between an equation-based and a procedural modeling language for building energy simulation. Submitted to: *SimBuild 2006 Conference*, <http://ceae.colorado.edu/ibpsa/SimBuild06/index.html>, August 2006.
- [17] Michael Wetter. Multizone airflow model in modelica. Submitted to: *Modelica 2006 conference*, <http://www.modelica.org/events/modelica2006/>, September 2006.
- [18] Michael Wetter and Jonathan Wright. Comparison of a generalized pattern search and a genetic algorithm optimization method. In G. Augenbroe and J. Hensen, editors, *Proc. of the 8-th IBPSA Conference*, volume III, pages 1401–1408, Eindhoven, NL, August 2003.
- [19] Michael Wetter and Jonathan Wright. A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. *Building and Environment*, 39(8):989–999, August 2004.
- [20] F. C. Winkelmann, B. E. Birsdall, W. F. Buhl, K. L. Ellington, A. E. Erdem, J. J. Hirsch, and S. Gates. DOE-2 supplement, version 2.1E. Technical Report LBL-34947, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, November 1993.

Session 5c

Free and Commercial Libraries 1

The LinearSystems library for continuous and discrete control systems

Martin Otter

German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany

Abstract

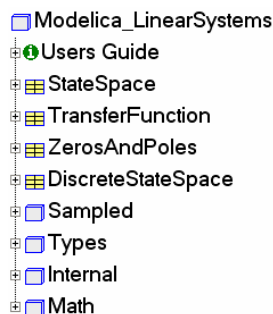
The free Modelica LinearSystems library provides basic data structures for linear control systems as well as operations on them, and includes blocks that allow quick switching between a continuous and a discrete representation of a multi-rate controller. The advantage is that fast simulations of the overall system can be performed when the modeling detail of the continuous controller is sufficient. It is planned to include this library in the Modelica standard library.

1 Introduction

Library *LinearSystems* is a free Modelica package providing different representations of linear, time invariant differential and difference equation systems, as well as typical operations on these system descriptions. In the right figure a screen-shot of the first hierarchical level of the library is shown. *StateSpace*, *TransferFunction*, *ZerosAndPoles*, and *DiscreteStateSpace* are records that provide basic data structures for linear control systems. Every record contains a set of utility functions that operate on the corresponding data structure. Especially, operations are provided to transform the respective data structures in to each other.

Sublibrary *Math* contains the basic data structures *Complex* and *Polynomial* that are utilized from the linear system descriptions.

Sublibrary *Sampled* contains a library of input/output blocks to conveniently model and simulate sampled data systems where it is convenient to quickly switch between a continuous and a discrete block representation.



2 Sublibrary Math

This sublibrary provides the basic data structures *Complex* and *Polynomial* that are utilized and needed from the linear systems data structures to be discussed in the next section.

In the Modelica Association there is currently a proposal to introduce operator overloading into Modelica 3.0. The data structures in the LinearSystems library are organized so that their usage becomes convenient once operator overloading is available.

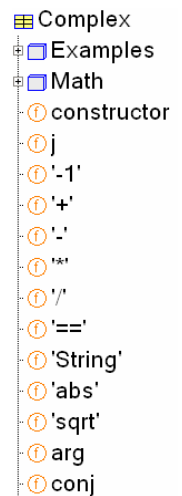
The basic approach will be explained at hand of the record *Math.Complex*. Its content is displayed on the right side. The record contains the basic definition of a complex number consisting of a real (re) and an imaginary (im) part:

```
record Complex
  Real re;
  Real im;
  // function definitions
end Complex;
```

Additionally, a set of functions is present that operates on this record. Especially, function *constructor(..)* constructs an instance of this record and all functions 'xx' provide basic operations such as addition and multiplication. Note, the apostrophe is part of the function name. In the operator overloading proposal, predefined function names are defined, such as '+' that will be automatically invoked when a *Complex* number is used in an arithmetic expression.

The *Complex* number record can be utilized in *current Modelica environments*, such as Dymola (Dynasim 2006), in the following way:

```
import Modelica.Utilities.Streams;
import LinearSystems.Math.Complex;
Complex c1=Complex(re=2, im=3) "= 2 + 3j";
Complex c2=Complex(3,4)        "= 3 + 4j";
```



algorithm

```
c3 := Complex.'+'(c1, c2) "= c1 + c2";
Streams.print("c3 = " +
  Complex.'String'(c3));
Streams.print("c3 = " +
  Complex.'String'(c3,"i"));
```

In the declaration, the two record instances *c1* and *c2* form *c3*. The value of record *c3* is printed with the `Complex.'String'` functions in different forms resulting in the following print-out:

```
c3 = 5 + 7j
c3 = 5 + 7i
```

Once operator overloading is available, the above statements can be simply written as:

```
c3 := c1 + c2;
Streams.print("c3 = " + String(c3));
Streams.print("c3 = " + String(c3,"i"));
```

It is then also possible to write:

```
Complex j = Complex.j();
Complex c4 = -2 + 5*j;
Complex c5 = c1*c2 / ( c3 + c4);
```

In record `Math.Complex` the basic operations for complex numbers are provided.

In a similar way all other data structures of the `LinearSystems` library are defined. They can all be at once used in current Modelica environments and they will be conveniently usable once operator overloading is available in Modelica and in Modelica tools.

Record `Math.Polynomial` defines a data structure for polynomials with real-valued coefficients and provides the most important operations on polynomials. Its content is displayed on the right side. A Polynomial is constructed by the command

```
Polynomial(coeff.Vector)
```

where the input argument provides the polynomial coefficients in descending order. For example, the polynomial $y = 2x^2 + 3x + 1$ is defined as

```
Polynomial({2,3,1})
```

Besides arithmetic operations, functions are provided to compute the zeros of a polynomial (via the eigen values of the companion matrix), to differentiate, to integrate and to evaluate the function value, its derivative and its integral. Find below a typical example from the scripting environment of Dymola:

```
import LinearSystems.Math.Polynomial;
p = Polynomial({6, 4, -3})
Polynomial.'String'(p)
// = "-6*x^2 + 4*x - 3"

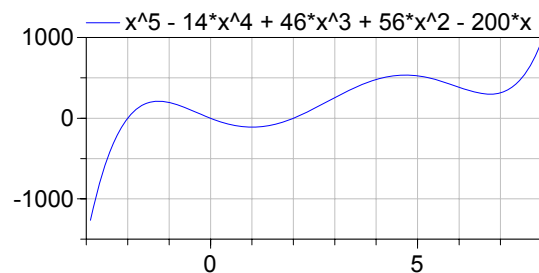
int_p = Polynomial.integral(p)
Polynomial.'String'(int_p)
// = "-2*x^3 + 2*x^2 - 3*x"

der_p = Polynomial.derivative(p)
Polynomial.'String'(der_p)
// = "-12*x + 4"

Polynomial.evaluate(der_p,1)
// = -8

r = Polynomial.roots(p, printRoots=true)
// = 0.333333 + 0.62361j
// = 0.333333 - 0.62361j
```

With function *fitting*, a polynomial can be determined that approximates given table values. Finally with function *plot*, the interesting range of *x* is automatically determined (via calculating the roots of the polynomial and of its derivative) and plotted. The plotting is currently performed with the *plot* function in Dymola. Once a Modelica built-in plot function is available, this will be adapted. A typical plot is shown in the next figure:



3 Linear System Descriptions

At the top level of the `LinearSystems` library, data structures are provided as Modelica records defining different representations of linear, time invariant differential and difference equation systems. In the record definitions, functions are provided that operate on the corresponding data structure. Currently, the following linear system representations are available:

3.1 Record StateSpace

This record defines a multi-input, multi-output linear time-invariant differential equation system in state space form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t)$$

The data part of the record contains the constant matrices A,B,C,D in the following definition:

```
record StateSpace
  Real A[:, :];
  Real B[size(A,1), :];
  Real C[:, size(A,1)];
  Real D[size(C,1), size(B,2)];
  // function definitions
end StateSpace;
```

The content of the StateSpace record definition is displayed on the right side. The *fromXXX* functions transform from another representation to a StateSpace record. Especially *fromModel(.)* linearizes a Modelica model and provides the linear system as output argument. *fromFile(.)* reads a StateSpace description from file and *fromTransferFunction(.)* transform a transfer function description into a StateSpace form.

The basic arithmetic operations are interpreted as series and parallel connection of StateSpace systems. Function *invariantZeros(.)* computes the invariant zeros. For single-input, single-output systems these are the zeros of the transfer function. Function *plotEigenValues(.)* computes and plots the eigenvalues of the system.

3.2 Record TransferFunction

This record defines the transfer function between the input signal *u* and the output signal *y* by the coefficients of the numerator and denominator polynomials *n(s)* and *d(s)* respectively:

$$y = \frac{n(s)}{d(s)} \cdot u$$

The order of the numerator polynomial can be larger as the order of the denominator polynomial (in such a case, the transfer function can not be transformed to a StateSpace system, but other operations are possible). For example, the transfer function

$$y = \frac{2 \cdot s + 3}{4 \cdot s^2 + 5 \cdot s + 6} \cdot u$$

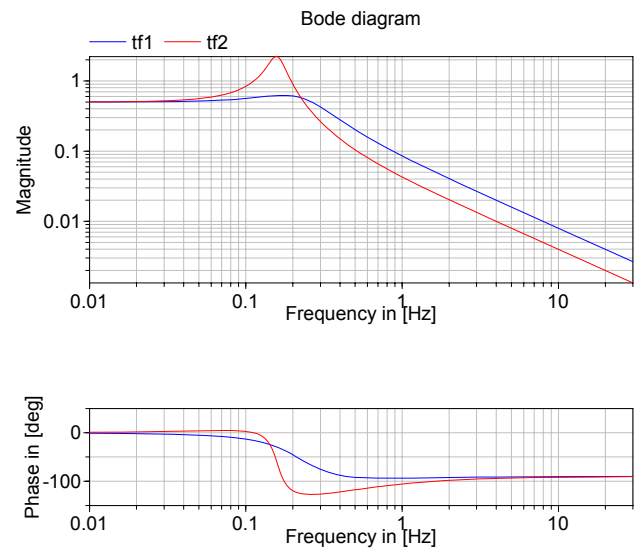
is defined in the following way:

```
import TF=LinearSystems.TransferFunction;
import Modelica.Utilities.Streams;
TF tf(n={2,3}, d={4,5,6});
print("y = " + TF.'String'(tf) + " * u")
```

The last statement prints the following string to the output window:

$$y = (2*s + 3) / (4*s^2 + 5*s + 6) * u$$

Besides arithmetic operations on transfer functions and constructing them optionally also from polynomials and from zeros and poles, the zeros and poles can be computed and a Bode plot can be constructed, as shown in the next figure:



The function call `plotBode(tf1, legend="tf1")` automatically selects an appropriate frequency interval and uses the selected legend. The useful frequency range is estimated such that the phase angle of the plot of *one* (numerator or denominator) zero is in the range:

$$\frac{\varphi_{\min}}{n} \leq |\text{phase angle}| \leq \frac{\pi}{2} - \frac{\varphi_{\min}}{n}$$

where *n* is the number of (numerator or denominator) zeros. Note, the phase angle of one zero for a frequency of 0 up to infinity is in the range:

$$0 \leq |\text{phase angle}| \leq \frac{\pi}{2}$$

Therefore, the frequency range is estimated such that the essential part of the phase angle (defined by φ_{\min}) is present in the Bode plot (default is 10^{-4} rad).

3.3 Record ZerosAndPoles

This record defines the transfer function between the input signal u and the output signal by its zeros, poles and a gain:

$$y = k \cdot \frac{\prod (s - z_i)}{\prod (s - p_j)} \cdot u$$

where the zeros and poles are defined by two Complex vectors of coefficients z_i and p_j . The elements of the two Complex vectors must either be real numbers or conjugate complex pairs (in order that their product results in a polynomial with Real coefficients).

A description with zeros and poles is problematic: For example, a small change in the imaginary part of a conjugate complex pole pair, leads no longer to a transfer function with real coefficients. If the same zero or pole is present twice or more, then a diagonal state space form is no longer possible. This means that the structure is very sensitive if zeros or poles are close together. Performing arithmetic operations on such a description therefore leads easily to transfer functions with non-real coefficients. For this and other reasons, the constructor transforms this data structure and stores it internally in the record as first and second order polynomials with real coefficients:

$$y = k \cdot \frac{\prod (s + n_{1i}) \cdot \prod (s^2 + n_{2j} \cdot s + n_{3j})}{\prod (s + d_{1k}) \cdot \prod (s^2 + d_{2l} \cdot s + d_{3l})} \cdot u$$

All functions operate on this data structure. It is therefore guaranteed that an operation results again in a transfer function with real coefficients. It is possible to construct a ZerosAndPoles record directly with these coefficients by using function `fromFactorized(..)`.

This data structure is especially useful in applications where first and second order polynomials are naturally occurring, e.g., as for filters: With function `filter(..)` this factorized form is directly generated from the desired low and high pass filters of type CriticalDamping, Bessel, Butterworth or Chebyshev. The filter options can be seen in Figure 1, a screenshot of the input menu of this function.

Besides type of filter and whether it is a low or high pass filter, the order of the filter and the cut-off frequency can be defined. With option "normalized", filters can be defined in normalized (default) and non-normalized form.

- ZerosAndPoles
- Examples
- constructor
- fromReal
- fromTransferFunction
- fromFactorization
- filter
- LaplaceVariable
- **
- 'String'
- numeratorDegree
- denominatorDegree
- evaluate
- zerosAndPoles
- plotBode

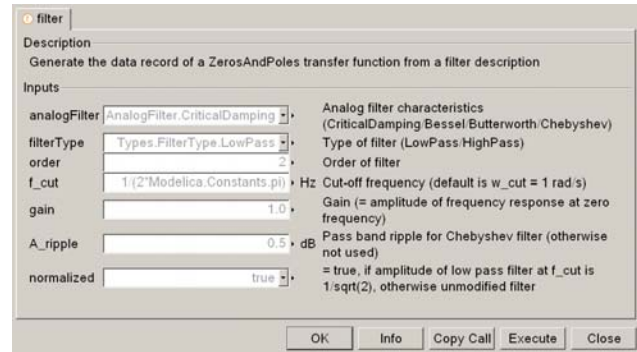


Figure 1: Input menu of function `filter(..)`.

In the normalized form, the amplitude of the filter transfer function at the cutoff frequency is $1/\sqrt{2}$ (≈ 3 dB). When trying out different filter types with different orders, the result of a comparison makes only sense if the filter is normalized.

Note, when comparing the filters of this function with other software systems, the setting of "normalized" has to be selected appropriately. For example, the signal processing toolbox of Matlab provides the filters in non-normalized form and therefore normalized = false has to be set.

In Figure 2, the magnitudes of the 4 supported low pass filters are shown in normalized form and in Figure 5 the corresponding step responses are given (generated with the `LinearSystems.Sampled` library).

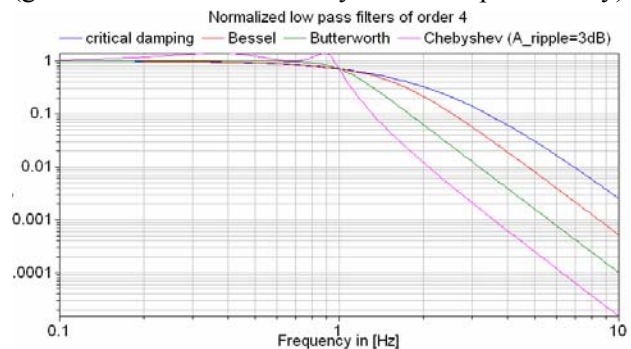


Figure 2: Magnitudes of normalized low pass filters

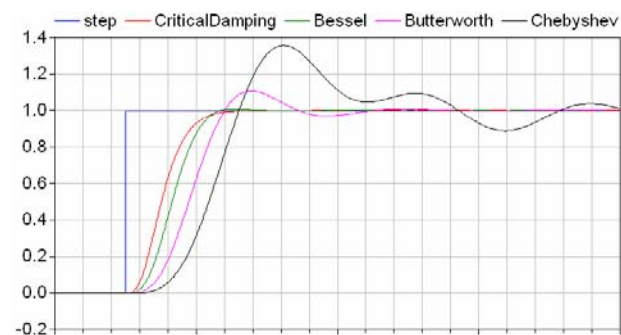


Figure 3: Step responses of norm. low pass filters

Obviously, the frequency responses give a somewhat wrong impression of the filter characteristics: Although Butterworth and Chebyshev filters have a significantly steeper magnitude as the CriticalDamp-

ing and Bessel filters, the step responses of the latter ones are much better since the settling times are shorter and no overshoot occurs. This means for example, that a CriticalDamping or a Bessel filter should be selected, if a filter is mainly used to make a non-linear inverse model realizable.

3.4 Record DiscreteStateSpace

This record defines a linear time invariant *difference equation system* in state space form. At the time of writing, this record contains only the core function `fromStateSpace(..)` to transform a `StateSpace` description in a `DiscreteStateSpace` form. The details of this record and of this function are discussed in section 4.3.

4 Sublibrary Sampled

4.1 Overview

The core of the LinearSystems library is sublibrary *Sampled* to model continuous and discrete multi-rate control systems. Experience shows that the combination of physical plant models that are controlled by digital controllers slow down the simulation speed significantly, if the sampling rates of the digital controllers are small compared to the step-size that could be used for the continuous plant. For example, at DLR detailed, calibrated models of robot systems are available. A stiff solver with variable step size integrates this system with step sizes in the order of 10 – 20 ms. When replacing the continuous approximation of the controllers by the actual digital controller implementation, the simulation time is **increased** by a factor of **20-30**. The reason is that the digital controllers have a sample time in the order of 1 ms and therefore the step size of the integrator is limited by 1 ms. Additionally at every sample point the integrator has to be restarted to reliably handle the discontinuous change of the actuator signal which reduces again the simulation efficiency.

For some design phases a continuous approximation of a controller might be sufficient, e.g., when tuning the controller parameters by parameter variation or multi-criteria optimization. By reducing the simulation time with a factor of, say 20-30, will then significantly reduce the optimization time.

It is also necessary to validate or fine tune the controllers with the most detailed models available. Then effects such as sampling, AD-/DA-converter quantization, resolver quantization and noise, com-

puting time of the control algorithms, signal communication times, as well as additional filters have to be taken into account.

Practical experience at DLR shows that it is difficult to maintain the consistency between a continuous and a digital representation of a control system model. For this reason, in a master thesis project (Walther 2002) a Modelica library was developed that allows to easily switch between a continuous and a discrete representation of a controller. This library has been in use at DLR for some years. Based on the gained experience in using this library, as well as new features in Modelica and in the Modelica simulation environment Dymola (Dynasim 2006), the library was considerably restructured, and completely newly implemented. Besides (Walter 2002), the books of (Aström and Wittenmark 1997) and (Tietze and Schenk 2002) have been helpful for the design and the actual implementation.

4.2 Introductory Example

In the left part of Figure 4 below a screenshot of the Sampled library and a simple example in using it is shown in the right part of the figure. The example is a simple controlled flexible drive consisting of a motor inertia, the gear elasticity and the load inertia. The angle and the angular velocity of the motor inertia are measured and are used in the position and speed controller. The output of the speed controller is directly used as the torque driving the motor inertia. The multi-rate controller consists of all blocks in the red square together with component *sampleClock*. The latter defines the base sampling time together with defaults for the blockType (Continuous or Discrete), the methodType (the used discretization method for a discrete block) and the block initialization (none, initialize states, initialize in steady state). All Sampled blocks on the same hierarchical level as *sampleClock*, and all blocks on a lower hierarchical level, use the *sampleClock* setting as a default. Every block can override the default and can use its individual settings.

In the example of Figure 5, block *filter* is a continuous filter that filters the reference motion of the motor (here: *ramp*). For this reason, *filter* uses explicitly the blockType *Continuous* whereas all other blocks use the blockType *UseSampleClockOption*, i.e., they use the setting defined in *sampleClock*. By changing the blockType from *Continuous* to *Discrete* in block *sampleClock*, the controller is automatically transformed from a continuous to a discrete representation.

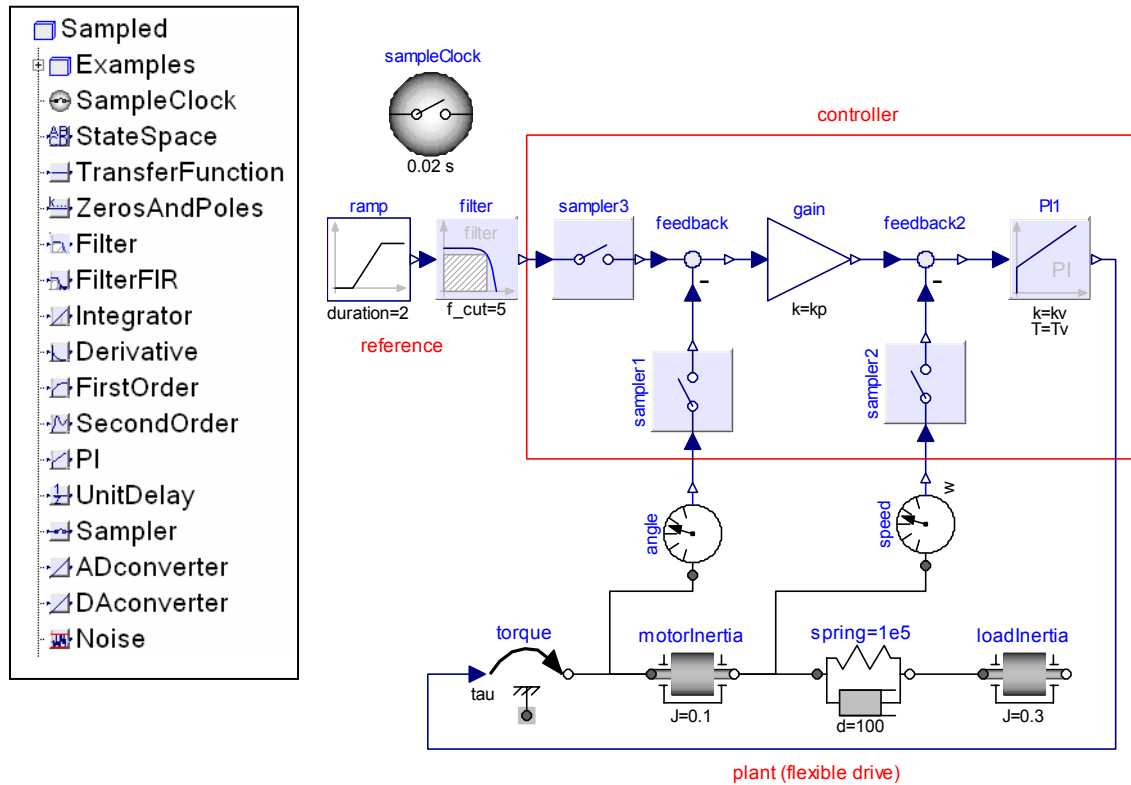


Figure 4: Blocks of Sampled library (left figure) and introductory example (right figure).

Every block of the Sampled library has a *continuous* input and a *continuous* output. Inside the respective block, the input and output signals might be sampled with the base sampling period defined in “sampleClock” or with an Integer multiple of it. For example, the PI controller in Figure 4 has the following parameter menu to define the continuous parameterization of the PI controller (i.e., gain k and time constant T):

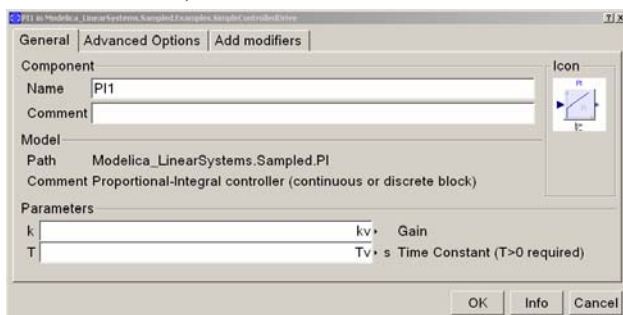


Figure 5: Parameter menu of PI controller

In the *Advanced Options* tab, see Figure 6, the remaining block settings are present, especially to define whether it is a continuous or a discrete block and in the latter case define also the discretization method. Since parameter *sampleFactor* is 1, the base sampling time of the *sampleClock* component is used. In other blocks of the controller (*Sampler1*,

Sampler2), parameter *sampleFactor* = 5 which means that the inputs and outputs of these blocks are sampled by a sampling rate that is 5 times of the base sample time defined in the *sampleClock* component. Note, the *samplerX* blocks in Figure 4 have only an effect if the controller is discrete. For a continuous representation, these blocks just state that the output signal is identical to the input signal.

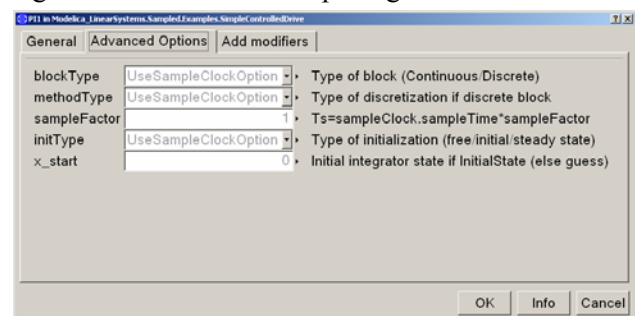


Figure 6: Advanced Options parameter menu

Finally, with parameter *initType* the type of initialization can be defined by using either the setting from the *sampleClock* component (default) or a local definition. The default of the *sampleClock* component initialization is “steady state”. This means that *continuous* blocks are initialized so that the *state derivatives are zero* and *discrete* blocks are initialized so that a re-evaluation of the discrete equations gives

the *same discrete states* (provided the input did not change). Since the equations of the blocks of the Sampled library are linear, the default setting leads to linear systems of equations during initialization that can be usually solved very reliably. The “steady state” initialization for a control system or a filter has the significant advantage that unnecessary settling times after simulation start are avoided.

4.3 Discretization Methods

The core of the Sampled library is function *DiscreteStateSpace.fromStateSpace* that transforms a linear, time invariant **differential equation** system in state space form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t)$$

into a linear, time invariant, **difference equation** system in state space form

$$\mathbf{x}_d((k+1) \cdot T_s) = \mathbf{A} \cdot \mathbf{x}_d(k \cdot T_s) + \mathbf{B} \cdot \mathbf{u}(k \cdot T_s)$$

$$\mathbf{y}(k \cdot T_s) = \mathbf{C} \cdot \mathbf{x}_d(k \cdot T_s) + \mathbf{D} \cdot \mathbf{u}(k \cdot T_s)$$

$$\mathbf{x}(k \cdot T_s) = \mathbf{x}_d(k \cdot T_s) + \mathbf{B}_2 \cdot \mathbf{u}(k \cdot T_s)$$

where

- t is the time
- T_s is the sample time
- k is the index of the actual sample instance ($k = 0, 1, 2, \dots$)
- $\mathbf{u}(t)$ is the input vector
- $\mathbf{y}(t)$ is the output vector
- $\mathbf{x}(t)$ is the state vector of the continuous system from which the discrete block has been derived.
- $\mathbf{x}_d(t)$ is the state vector of the discretized system

If the discretization method, e.g., the trapezoidal integration method, accesses actual and past values of the input \mathbf{u} (e.g. $\mathbf{u}(T_s \cdot k)$, $\mathbf{u}(T_s \cdot (k-1))$, $\mathbf{u}(T_s \cdot (k-2))$), a state transformation is needed to arrive at the difference equation above where only the actual value $\mathbf{u}(T_s \cdot k)$ is accessed.

If the original continuous state vector at the sample times shall be computed from the discrete equations, the matrices of this transformation have to be known. For simplicity and efficiency, library LinearSystems supports only the specific transformation as needed for the provided discretization methods: As a result, the state vector of the underlying continuous system can be calculated by adding the term $\mathbf{B}_2 \cdot \mathbf{u}$ to the state vector of the discretized system. In fact, since the **discrete** state vector depends on the discretization method, it is a protected variable that cannot be accessed outside of the block. This vector is also not

needed by the user of the block, because the continuous state vector is available both in the continuous and in the discrete case.

In Modelica notation, the difference equation above is implemented as:

```
when {initial(), sample(Ts,Ts)} then
  new_x_d = A * x_d + B * u;
  y = C * x_d + D * u;
  x_d = pre(new_x_d);
  x = x_d + B2 * u;
end when;
```

Since no "next value" operator is available in Modelica, an auxiliary variable "new_x_d" stores the value of "x_d" for the next sampling instant. The relationship between "new_x_d" and "x_d" is defined via equation " $\mathbf{x}_d = \text{pre}(\text{new_x_d})$ ".

The body of the when-clause is active during initialization and at the next sample instant $t = T_s$. Note, the when-equation is **not** active after the initialization at $t = 0$ (due to sample(T_s, T_s) instead of sample($0, T_s$)), since the state "x_d" of the initialization has to be used also at $t = 0$. Additional equations are added for the initialization to uniquely compute vectors "x" and "x_d" at the initial time::

```
initial equation
  if init == InitialState then
    x = x_start;
  elseif init == SteadyState then
    x_d = new_x_d;
  end if;
```

If option *InitialState* is set, the discrete state vector "x_d" is computed such that the continuous state vector " $\mathbf{x} = \mathbf{x}_{\text{start}}$ ", i.e., the continuous state vector is identical to the desired start value of the continuous state.

If option *SteadyState* is set, the discrete controller is initialized in steady state (= default setting in *sampleClock*). This means that the output $\mathbf{y}(T_s \cdot k)$, $k=0,1,2,\dots$, remains constant provided the input vector $\mathbf{u}(T_s \cdot k)$ remains constant. Most simulation systems support steady state initialization only for the continuous part of a model. Due to the equation based nature of Modelica, where basically everything is mapped to equations, it is possible in Modelica to initialize also a mixture of continuous and discrete equations in steady state.

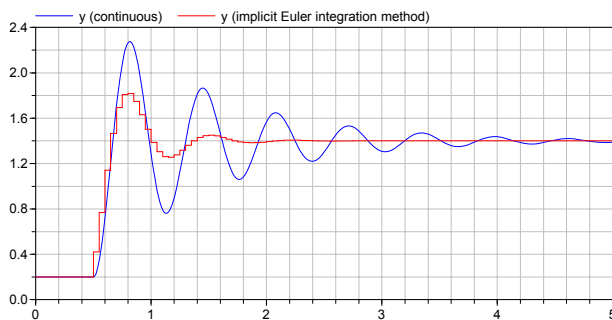
Via parameter *methodType* in the global block *sampleClock* or also individually for every block, the following discretization methods can be selected:

- explicit Euler integration method,
- implicit Euler integration method,
- trapezoidal integration method,

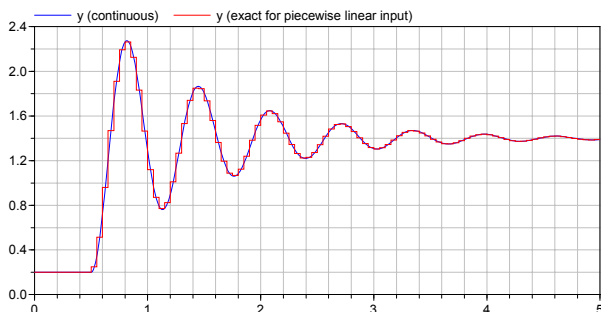
- exact solution under the assumption that the input signal is piecewise constant,
- exact solution under the assumption that the input signal is piecewise linear.

The last method (exact solution for piecewise linear input) gives nearly always the best approximation to the continuous solution without frequency or phase distortion. The effect of the discretization methods and of steady state initialization is demonstrated at hand of a simple PT2 block that is initialized in steady state, where the input signal is 0.2 at the beginning and jumps to 1.2 at 0.5 s

In the next figure, the blue curve is the solution of the continuous block and the red curve is the solution of the digital block using the **implicit Euler integration method** for the block discretization. As can be seen, even for this simple block the results of the continuous and digital representation are quite different. A much smaller sample time would be needed here, in order that the solutions of the discrete and the continuous representations would be closer together.



In the next figure, the red curve is the solution of the digital block using the **rampExact** method (i.e. the exact solution under the assumption that the input signal is piecewise linear). The solutions of the continuous and of the discrete blocks are practically identical without any phase shift. A better solution cannot be expected.



Continuous models in StateSpace form are transformed into discrete systems according to the sketched discretization methods. A TransferFunction system description is implemented as state space system in controller canonical form.

The transformation of a ZerosAndPoles system is more involved in order to reduce numerical difficulties, especially for filter implementations: The basic approach is to transform the transfer function

$$y = k \cdot \frac{\prod (s + n_{1i}) \cdot \prod (s^2 + n_{2j} \cdot s + n_{3j})}{\prod (s + d_{1k}) \cdot \prod (s^2 + d_{2l} \cdot s + d_{3l})} \cdot u$$

into a connected series of first and second order systems. All these systems are implemented as small state space systems in controller canonical form that are connected together in series. The output is scaled such that the gain of every block is one (if this is possible) in order that the inputs and outputs of the blocks are in the same order of magnitude. The output of the last block is multiplied with a factor so that the original gain of the transfer function is recovered.

A simpler, alternative implementation for both the TransferFunction and the ZerosAndPoles system would have been to compute the A,B,C,D matrices of the corresponding state space system with available function calls and then use the general StateSpace model for their implementations. The severe disadvantage of this approach is that the structure of the state space system is lost for the symbolic preprocessing. If, e.g., index reduction has to be applied (e.g. since a filter is used to realize a non-linear inverse model), then the tool cannot perform the index reduction anymore. Example:

Assume, a generic first order state space system is present

$$\dot{x} = a \cdot x + b \cdot u$$

$$y = c \cdot x + d \cdot u$$

and the values of the scalars a,b,c,d are parameters that might be changed before the simulation starts. If y has to be differentiated symbolically during code generation, then

$$\dot{y} = c \cdot \dot{x} + d \cdot \dot{u}$$

$$\dot{x} = a \cdot x + b \cdot u$$

As a result, the input u needs to be differentiated too, and this might not be possible and therefore translation might fail.

On the other hand, if the first order system is defined to be a low pass filter and the state space system is generated by keeping this structure, we have:

$$\dot{x} = -b \cdot x + u$$

$$y = x$$

Differentiating y symbolically leads to:

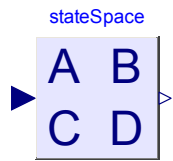
$$\dot{y} = \dot{x}$$

$$\dot{x} = -b \cdot x + u$$

Therefore, in this case, the derivative of u is not needed and the tool can continue with the symbolic processing.

4.4 Block Components

A short description of the input/output blocks of the Sampled sublibrary is given in Table 1. As an example block Sampled.StateSpace is shortly described.



Its icon is shown in the figure at the left. When double clicking on the block, the parameter menu of Figure 7 pops up. Here, either the name of a StateSpace record can be given, or when clicking on the “table” symbol at the right end of the input field another menu pops up in which the 4 matrices can be defined, as shown in Figure 8.

By clicking again on the “table” symbol a matrix editor appears that allows to conveniently define a matrix.

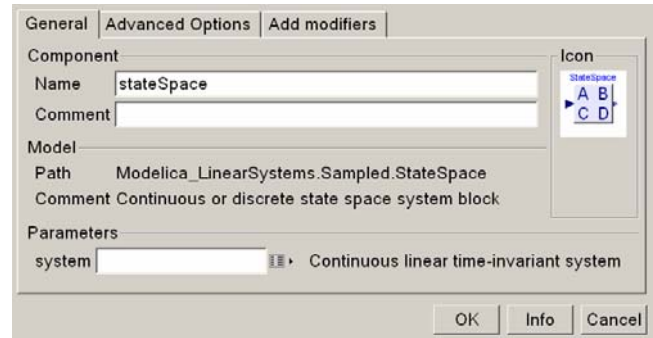


Figure 7: ParameterMenu of StateSpace block

Some of the blocks have only a pure discrete representation, such as the FilterFIR, UnitDelay, Sampler, ADconverter, DAconverter and Noise blocks. The continuous representation of these blocks is defined to be $y = u$, i.e., the output is identical to the input which means that these components are effectively removed.

















Name	Description
 SampleClock	Global options for blocks of Sampled library (e.g. base sample time)
 StateSpace	Continuous or discrete state space system block
 TransferFunction	Continuous or discrete, single input single output transfer function
 ZerosAndPoles	Continuous or discrete, single input single output block described by zeros and poles
 Filter	Analog low or high pass IIR-filters (CriticaDaming/Bessel/Butterworth/ Chebyshev)
 FilterFIR	Discrete finite impulse response (low or high pass) filters
 Integrator	Output the integral of the input signal (continuous or discrete block)
 Derivative	Approximate derivative (continuous or discrete block)
 FirstOrder	First order (continuous or discrete) transfer function block (= 1 pole)
 SecondOrder	Second order (continuous or discrete) transfer function block (= 2 poles)
 PI	Proportional-Integral controller (continuous or discrete block)
 UnitDelay	Delay the input by a multiple of the base sample time if discrete block or $y = u$ if continuous block
 Sampler	Sample the input signal if discrete block or $y = u$ if continuous block
 ADconverter	Analog to digital converter (including sampler)
 DAconverter	Digital to analog converter (including zero order hold)
 Noise	Generates uniformly distributed noise in a given band at sample instants if discrete and $y = 0$ if continuous

Table 1: Blocks of the Sampled sublibrary.

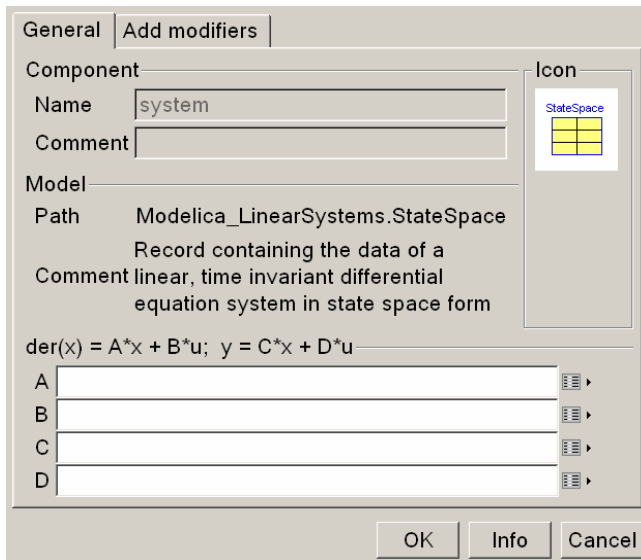


Figure 8: ParameterMenu of StateSpace record

5 Conclusion

The new, free LinearSystems library has been presented and important implementation details have been discussed. The library introduces a standard definition of the most important linear systems used for controller definitions and provides continuous and discrete block representations of every block. Some standard blocks, such as the Filter block for the most important IIR filters and the FilterFIR block for the most important FIR filters are provided.

The library is currently available in a Beta release. The available functionality is already very useful for controller simulations. For a first 1.0 release, some missing functions and components will be added and the documentation will be improved.

6 Acknowledgements

The first version of the LinearSystems.Sampled library was the library "Sampled" of Nico Walther as a result of his master thesis from the electrical engineering at the HTWK-Leipzig (supervised by Prof. Müller, HTWK, and Prof. Martin Otter, DLR). Based on the experience in using the Sampled library, new features in Modelica as well as in Dymola, the Sampled library was considerably restructured, and newly implemented. Not all blocks from the previous version are yet included.

Some functionality of LinearSystems (e.g., linearizing a Modelica model by StateSpace.fromModel) has been originally developed by Sven Erik Mattsson from Dynasim.

The Math.Complex, Math.Polynomial, and TransferFunction packages are based on proposals from Hilding Elmqvist, Dynasim, presented at the 33rd Modelica design meeting in Bielefeld (Nov. 2003) and the 37th Modelica design meeting in Lund (Jan. 2004).

Several functions of package Complex have been provided by Anton Haumer, who also performed a thorough test of the package.

The design of the records (such as Math.Complex and Math.Polynomial) has been inspired by the discussions about operator overloading at various Modelica design meetings.

Advice for implementation issues given by Hans Olsson from Dynasim, as well as advice for some numerical algorithms given by Andras Varga and Dieter Joos from DLR is appreciated.

Partial financial support of DLR for the development of this library within the European Network of Excellence HYCON (Hybrid Control: taming heterogeneity and complexity of networked embedded systems; contract number: 511368), and within the German BMBF Verbundprojekt PAPAS (Plug-And-Play Antriebs- und Steuerungskonzepte für die Produktion von Morgen; Förderkennzeichen: 02PH2060) is highly appreciated.

References

- Aström K.J., Wittenmark B. (1997): *Computer Controlled Systems: Theory and Design*. Prentice Hall. 3rd edition.
- Dynasim (2006). *Dymola Version 6.0*. Dynasim AB, Lund, Sweden. Homepage: <http://www.dynasim.se/>.
- Tietze U., and Schenk C. (2002): *Halbleiter-Schaltungstechnik*. Springer Verlag, 12. Auflage, pp. 815-852.
- Walther N. (2002): *Praxisgerechte Modelica-Bibliothek für Abtastregler*. Diplomarbeit, HTWK Leipzig, Fachbereich Elektro- und Informationstechnik, supervised by Prof. Müller (HTWK) and Prof. Martin Otter (DLR), 12 Nov. 2002.

ARENALib: A Modelica Library for Discrete-Event System Simulation

Victorino S. Prat Alfonso Urquia Sebastian Dormido

Departamento de Informática y Automática, ETS de Ingeniería Informática, UNED

Juan del Rosal 16, 28040 Madrid, Spain

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

Abstract

The design, implementation and use of ARENALib is discussed in this manuscript. ARENALib is a new Modelica library for modeling, simulation and analysis of discrete-event systems (DES). This new Modelica library tries to replicate the functionality and capabilities of Arena: a general-purpose simulation environment supporting the process approach to DES modeling [1]. ARENALib library will be released soon under the GNU General Public License (GPL). The library's general architecture, implementation and use are presented. Also the results of some model simulations, validated with Arena's results are discussed, as well as the future work and conclusions.

Keywords: discrete-event; DES; Arena

1 Introduction

ARENALib is a new Modelica library for Discrete-Event System (DES) modeling and simulation.

The main objective of this library is to provide a modeling and simulation environment for DES using the process approach, opposite to other contributions in Modelica that use Statecharts[2, 3] or Petri nets[4] approaches.

ARENALib has also to be considered as a general-purpose tool, instead of application-oriented libraries developed by other authors (for example [5]). DES models built using ARENALib are completely written in Modelica language and they can be simulated using Dymola, as opposed to other approaches that require the combined use of different software tools (for instance, [6]).

When finished, ARENALib will be freely distributed under the GNU-GPL license.

ARENALib has been designed and implemented replicating the functionality and capabilities of Arena [1],

which is a simulation environment for DES modeling and simulation using the process approach.

ARENALib validation is performed by comparison with Arena, when simulating the same systems.

In the next section, the capabilities and functionality of Arena will be discussed. It will be followed by a description of ARENALib general architecture, detailing the main components of the library. The connection between discrete and continuous systems using ARENALib will also be presented, as well as the random numbers and variates generation. After that, a simple case study will be explained in detail in order to show the use of the library, followed by the simulation and experiment setup. Finally an extended case study, future work notes and conclusions are provided.

2 Arena

The process approach describes the system from the entity perspective. The entity is the basic component of the simulation model.

To support this approach to DES simulation, Arena components are arranged into panels. The main panel is called Basic Process, and contains the main basic components for system simulation.

Panel components can be classified into two types: *flowchart modules* and *data modules*. Flowchart modules allow to describe the entity flow through the system (dynamic part of the system). They include Create, Process, Dispose, Decide, Batch, Separate, Assign and Record modules. Data modules describe the characteristics of system elements (static part), such as Entity, Queue, Resource, Variable, Schedule and Set modules. The procedure for building a model consists in drawing the flowchart diagram and configuring the required data modules. This procedure is analogous in ARENALib and will be detailed in Section 10.

At this moment, ARENALib implements the Create,

Process, Dispose and Decide flowchart modules and the Entity, Resource and Queue data modules. This implementation is described below.

3 *ARENALib* Architecture

ARENALib has been designed around the idea of implementing the Arena's Basic Process panel and being able to model and simulate the same kind of systems than Arena. As a consequence, its architecture has been divided in two parts (*ARENALib* general architecture is shown in Figure 1a)):

- *The user zone*, that contains the Basic Process panel modules (flowchart and data), the Project model draft which is the start package for new systems' models and the tutorial package that includes some examples to help the beginner user to get used with the library. The structure of this zone is shown in Figures 1c),1b) and 1d).
- *The developer zone*, stored in the "src" package, contains all the internal models and functions used by *ARENALib* simulations. Details about this package are shown in Figures 1e) and 1f).

The user zone will be detailed in Section 10, while the developer zone will be discussed in the next sections. The main problem that has been addressed in *ARENALib* is the management of the dynamical behavior of the components of the system. It includes the entity flow management, seizing and releasing of resources and the generation of the statistical indicators. This problem has been solved using dynamic-memory structures implemented as an static library (written in C language). The static library is connected to the Modelica code using Modelica's external-function interface. Packages Ext and DynMem contains the functions to access the static library (these packages are shown in Figures 1e) and 1f)).

4 Basic Process Panel

Analogously to Arena's process approach schema, *ARENALib* has been divided in two type of modules: flowchart modules and data modules. *Flowchart Modules*, that enables the modeler to describe the flow of entities in the system. It is composed by:

- Create module, represents the creation or arrival of entities to the system.

- Process module, simulates the point for entity processing.
- Dispose module, where the entities leave the system.
- Decide module, simulates a division in the flow of entities.

Data Modules, representing the components of every process in the system, such as:

- EntityType module, describes the characteristics of the entities.
- Queue module, represents the queue where the entities should wait for processing.
- Resource module, are the components of the defined processes.

All these modules will be discussed in Sections 6 and 7.

5 Interfaces

As previously mentioned the connection between modules is one of the main problems encountered during the development of the library.

One of the problems of module connection is that, for a given point in time, several and different entities could be arriving at the same module. This means that the same connector has to receive different entities from different modules of the system.

At the beginning, the approach taken to solve this problem was to implement an entity receival interface for each module. This interface was basically a text file in which the information of the received entity was written. The module that received the entity could read it from the file and process it. This solution was successful for small systems. However, the use of text files is very time consuming and the performance for large models was poor.

To solve that performance problem a dynamic memory interface was implemented for data exchange among modules. This interface consists in storing in an Integer variable in Modelica a number that corresponds to the pointer to a Dynamic Data Structure (DDS) previously created. These DDS's are stored in memory and correspond to the main data structures used in the system's simulation, such as queues and linked lists of elements. There is a DDS for managing queues of entities, lists of resources, lists of entity types, lists of statistical indicators and lists of attributes of an entity.

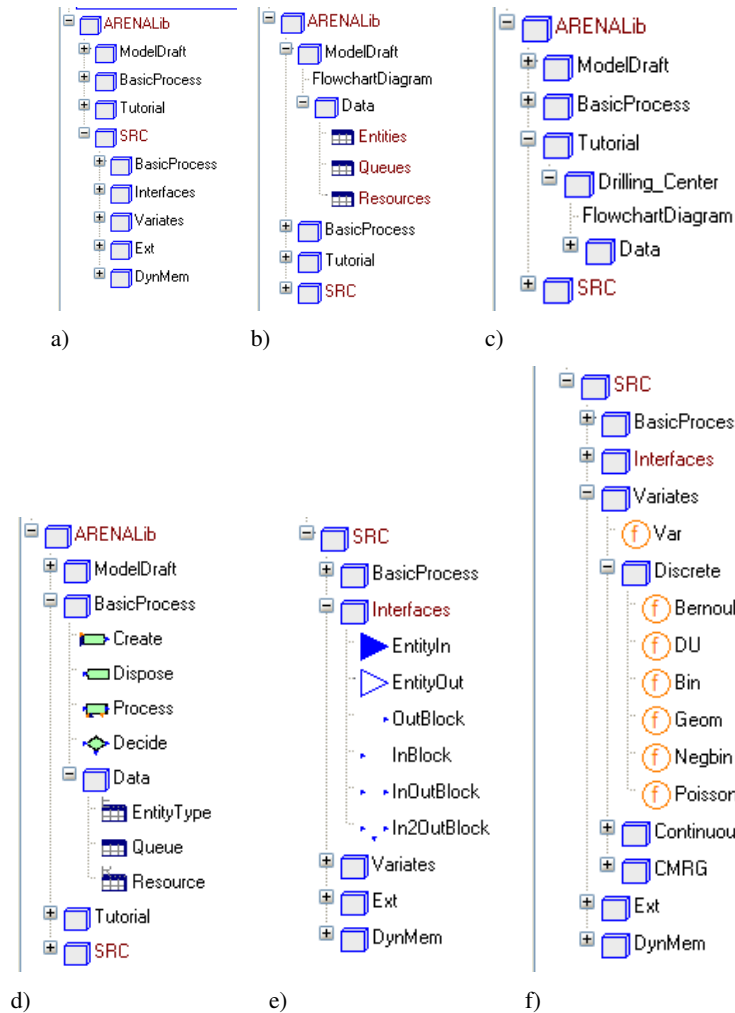


Figure 1: *ARENALib* architecture: a) General Architecture; b) Model Draft; c) Tutorial; d) BasicProcess; e) Interfaces; f) Random Variates

The performance was increased about 40 times from the text file's approach. (one input and two outputs).

So the interface between modules is composed by an integer number, which corresponds to the pointer to the entity receival queue of a given module, and indicates the memory address for accessing the queue. Also, every module has a queue of outgoing entities for managing the order in which the entities are transfered to the next module.

For the development of the flowchart modules, some general interfaces have been implemented. The basic ones are the connectors for input and output entities.

Based on these basic connectors several models have been created to cover all the possible interfaces of the flowchart modules. Depending on the number of inputs and outputs of each module they can be classified into: InBlock (one input), OutBlock (one output), InOutBlock (one input and output) and In2OutBlock

6 Flowchart Modules

Flowchart modules are used to model the flow of entities through the system. This entity flow begins in Create modules, and finish in Dispose modules. In this section, each of the currently implemented modules is detailed and it's use and configuration parameters are shown.

6.1 Create

As mentioned before, it allows to model the arrival of entities to the system. This arrival is expressed in form of an inter-arrival time between entities. The module extends the OutBlock interface module. The parameters of the module are the following:

- *EntityType*, is the type of the entities created in the module. This entity type has to be previously instantiated from an entity type data module.
- *Entity Generation Function*, represents the inter-arrival time. This time can be calculated from: (1) a probability distribution; (2) a constant expression and (3) a Modelica variable from any other model.
- *g1, g2 and g3*, are the parameters of the previous function. Depending on the distribution selected the meaning of each one changes.
- *TimeUnit*, is the local time unit. The inter-arrival time will be based on it.
- *Entities Per Arrival*, defines the number of entities generated in each arrival time.
- *Max Arrivals*, establishes the maximum number of entities generated in the module. If the number of entities reach this value, no more entities will be created.
- *First Creation*, sets the simulated time when the first entity is created.
- *Resources*, is the list of the resources associated to the process. Each resource has to be previously declared using a resource data module.
- *Resource Quantities*, defines the quantity of each resource that has to be seized by the entity. If any of the resources does not have the specified quantity available then the seize operation fails and the entity must wait in the queue. This parameter must have the same length than the previous one.
- *Queue*, is the queue associated to the process. It has to be previously defined with a queue data module.
- *Delay type*, represents the processing time for an entity. It can be a continuous value or a probability distribution.
- *g1, g2 and, g3*, are the parameters of the *Delay type* function.
- *TimeUnit*, also represents the local time unit. It is the base time for the delay value.
- *Allocation*, determines the kind of process the module is simulating. Possible values are “Value Added”, “Non Value Added”, “Wait”, “Transfer” and “Other”, and will influence the statistical results of the entities processed.

6.2 Process

It defines a process. It extends the InOutBlock interface module. The parameters of the module are the following:

- *Name*, the name of the module.
- *Type*, can be standard and submodel. The standard defines a simple process configured by the module parameters. On the other hand, the submodel indicates that this module is just a mask for a more complicated process, composed by a DES itself. The submodel type is not fully implemented, but it can be easily done by using the Modelica’s object oriented capabilities.
- *Action*, the possible values are Delay, Seize-Delay, Seize-Delay-Release and Delay-Release. Depending on the option chosen, the entity will seize a resource, be processed (delayed), and at the end release the resource.
- *Priority*, establishes the priority of entity selection from the waiting queue. This option is not implemented yet. The entity selected is the first in the queue (FIFO).

6.3 Dispose

It is the final point for the entities in the system: they are removed from the system and their statistical information is stored. For that reason, this module has no parameters.

6.4 Decide

It permits the modeler to simulate a division in the flow of the entities. Given a chance or a condition, the module decides the output connector the entity will leave the module through. It extends the In2OutBlock interface model. The parameters of the module are the following:

- *Type*, establishes the type of flow division, either by chance (percentage) or by condition. At this moment only the “by chance” option is implemented.
- *Percent True*, is the percentage of entities that will leave the module through it’s True output connector.

7 Data Modules

Data modules represent the static components of the system. These are the entities themselves and the rest of the components that will interact with them along its flow, such as queues and resources. At the present, three modules have been implemented and are detailed below.

7.1 EntityType

This data module defines the attributes of a type of entity. These attributes are the name, the picture of the entity (currently not implemented), and the costs associated to the entity.

7.2 Queue

It is used to describe process queues. The only type of queue currently implemented is the FIFO.

7.3 Resource

This module represents the resources used by the entities in the processes. An entity must seize (when needed) the resource in order to be processed. After processing, the entity can release the resource or not, depending on the kind of process performed.

8 Connection with other Modelica models

ARENALib modules can be connected to other Modelica models, so hybrid continuous-discrete systems can be easily modeled.

Arena provides the possibility of including continuous modules in the models, however this possibility is very limited. Analogously, *ARENALib* provides interfaces in each module to perform a connection with any Modelica model, which is much more powerful than Arena's capabilities.

This connection interfaces give the modeler the possibility of configuring *ARENALib* module parameters from external Modelica models.

We can separate the connection interfaces in two:

- Integer ones, that represent the transfer of entity related events between discrete and continuous systems. The input Integer connector is used to tell the system that an entity has arrived or that an entity has finished the processing (in the Create and Process modules). And the output connector

(in the Process module) is used to tell the continuous system which entity (with its serial number) has seized the resource and is ready to be processed.

- Real ones, are just values for the rest of the available parameters of the modules, such the time for the first arrival, the maximum arrivals, the priority, etc.

On the other hand, each discrete module provides information that can be accessed using the Modelica dot notation. The list of variables that can be accessed for each module is displayed in the information icon of the module's model.

9 Random Variables

A key point in DES simulation is the random number generation. This section has been divided into two parts to separate the random number generation (observations of the $U(0, 1)$ distribution) from the variate generation.

9.1 Random Number Generation

Due to the DES dependence on stochastic distributions and to ensure good simulation results, it is very important to have a good source for pseudo-random numbers, to build statistically good random variates[7].

Also, in order to be able to analyze and validate the results from *ARENALib* in comparison with the ones obtained from Arena, the same pseudo-random number generator has been implemented.

This random number generator is called CMRG (Combined Multiple Recursive Generator) [8] and has a period length of 2^{191} . This period length can be divided into disjoint streams, each of them of length 2^{127} .

ARENALib associates one of those streams with each Random Variable, so the random number stream independency can be ensured for variate generation.

The implementation of the CMRG has been done translating the implementation in C, done by Pierre L. Ecuyer (available on the web at [9]), into Modelica code. In this way, the generator is available for anyone outside the *ARENALib* environment. The only remarkable thing for its usage is the management of the seed, which is read from a text file and updated every time a new stream is generated.

9.2 Random Variates Generation

To provide the modeler enough functionality for modeling and simulating many kind of systems, some of the most commonly-used probability distributions have been implemented and included in the library.

These distributions are functions that use the stream created by the CMRG to generate random variates. The output variables of all the distribution functions are the variate's value and the updated random stream, that will be used to obtain more new variates. The input variables of the functions are the following:

9.2.1 Continuous Probability Distributions

- Uniform (min,max)
- Exponential (mean)
- Normal (mean,variance)
- LogNormal (mean,variance)
- Triangular (min,mode,max)

9.2.2 Discrete Probability Distributions

- Bernoulli (p)
- Discrete Uniform (min,max)
- Binomial (n,p)
- Geometric (p)
- Negative Binomial (n,p)
- Poisson (alpha)

10 ARENALib Use

In this section the use of ARENALib will be introduced by means of a simple example. The development of the model, the experiment setup, the analysis of the results and the validation with Arena are explained in detail.

10.1 Model Description

It is a very simple case of a processing system. The modeled system consists on a Drilling Center where the parts arrive, are drilled (processed) and leave.

The flowchart components of the system are a create module, that represents the parts arriving to the center, a process module, which is the drilling center itself and finally the dispose module where the parts leave the

system. The flowchart diagram is presented in Figure 2.

The data modules are: (1) the entity type that represents the processed parts; (2) the drilling press viewed as a resource; and (3) the queue associated to the press. For building this model in ARENALib we should use the package ModelDraft, duplicate and rename it to match our model requirements. This package includes all the basic components necessary to simulate the system.

Inside the ModelDraft there are two structures, the flowchart diagram that will be used to compose the system's structure and the data package which contains models for structuring all the needed data modules of our system.

For building the Drilling Center in ARENALib, an entity type, a resource and a queue modules have to be inserted in their corresponding data models, and then the flowchart diagram has to be drawn by drag a drop of a create, a process and a dispose module.

The last step is to configure the parameters of each data and flowchart module, as desired, to match the requirements of the system.

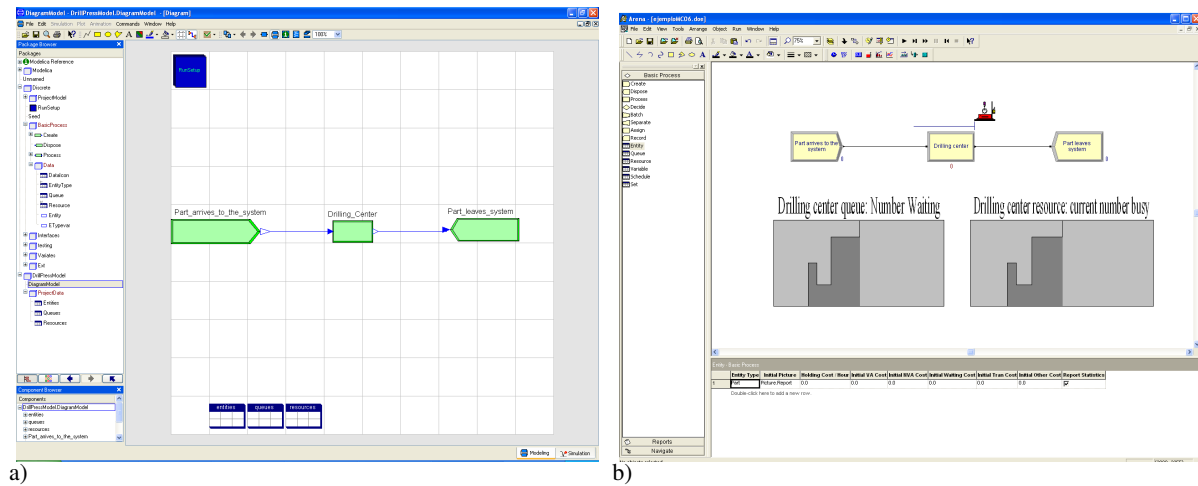
10.2 Simulation and Experiment Setup

Indicator	Arena		ARENALib
	Average	Half Width	Average
Part.NumberIn	20010		19887
Part.NumberOut	20006		19887
Part.VATime	3.3254	0.01574	3.3385
Part.WaitTime	3.4822	0.24270	3.4571
Part.WIP	1.3622	0.06354	1.3527
VATimePerEntity	3.3254	0.1574	3.3385
WaitTimePerEntity	3.4822	0.24270	3.4571
TotalTimePerEntity	6.8077	0.24771	6.7956
Queue.NumberInQueue	0.69695	0.05623	0.68754
Queue.WaitingTime	3.4827	0.2427	3.4571

Table 1: Results of the drilling center simulations

When having the system's model, and after having configured all the parameters of the modules, a simulation experiment can be run.

There are two basic parameters for each simulation. The simulation time and the global time unit. The first one establishes the duration of the simulation experiment, and the second one is the time unit (seconds, minutes, hours, etc.) in which the simulation time is established. These two parameters will influence the time unit parameter of the modules in the system, translating the values in each module to match the simulation time.


 Figure 2: Drilling center model composed using: a) *ARENALib*; and b) *Arena*

When the simulation time is finished, the statistical results of the run are written to a file named “SimResults.dsc”. After that, the only remaining variable in the system is the CMRG seed. This ensures that several experiments of the same model will, each of them, use different pseudo-random numbers.

Results from the Drilling Center simulation are shown in Table 1. These results have been obtained using Dymola.

Several experiments can be easily configured using Modelica’s scripting facilities.

Other result analysis capabilities are the plots. Every module has several variables that can be plotted at the end of the simulation and permits the analysis of the evolution in time of the experiment. This can be very useful to detect peak loads or compare several parameters in the system.

11 Case Study

As a case study, a Bottle Filling process is discussed. This process consists in a tank which fills bottles with liquid. Once a bottle is full, it is labeled and controlled in two quality control processes. The bottles that pass the first control are considered as first class bottles and the ones that pass the second control are considered as second class bottles. Any bottle that doesn’t pass the second quality control process is cleaned and relabeled again.

The tank has been modeled as a continuous system which fills the bottles at a constant rate. Every 400 time units the tank is refilled to it’s maximum level. The flowchart diagram of the process is displayed in Figure 3, modeled either in Modelica and Arena.

Indicator	Arena		ARENALib
	Average	Half Width	Average
Bottle.NumberIn	637		650
Bottle.NumberOut	623		632
Bottle.VATime	6.1618	0.35771	6.405
Bottle.WaitTime	35.894	5.2451	36.237
Bottle.WIP	5.3305	(Corr)	5.5132
Labeling.NumberIn	659		677
Labeling.NumberOut	645		660
Labeling.VATimePerEntity	5.3348	0.13555	5.3026
Labeling.WaitTimePerEntity	34.697	(Corr)	34.796
Labeling.TotalTimePerEntity	40.031	5.6018	40.098
Labeler.q.NumberInQueue	4.5534	(Corr)	4.7017
Labeler.q.WaitingTime	34.723	(Corr)	34.839
Cleaning.NumberIn	22		28
Cleaning.NumberOut	22		27
Cleaning.VATimePerEntity	19.202	(Insuf)	20.44
Cleaning.WaitTimePerEntity	1.0079	(Insuf)	0
Cleaning.TotalTimePerEntity	20.210	(Insuf)	20.44
Cleaner.q.NumberInQueue	0.0044	(Insuf)	0
Cleaner.q.WaitingTime	1.0079	(Insuf)	0

Table 2: Results of the Bottle Filling simulations

The results of the simulation and the validation data, from the comparison with the Arena’s results, are presented in Table 2. Also some plots from Dymola’s simulation results are shown in Figure 4.

12 Future Work

The main task for the future work will be the Basic Process panel development completion, including the modules still not implemented and the rest functionalities for the existing ones.

Other problems, that actually delay the development of the whole library, are the management of different data types for attributes and variables and the implementation of variable size matrices, for example, when introducing entity attributes that can be modified dynami-

cally during the simulation by any module, or the use of system variables whose value and size can change similarly to the attribute ones.

The solution for these problems is still in progress but will be solved soon, enabling the further development of the whole library.

The tutorial package will also be completed with more examples as well as the information pages for every module, either in the user zone and the development zone.

Visual representation of the simulations will also be studied.

13 Conclusions

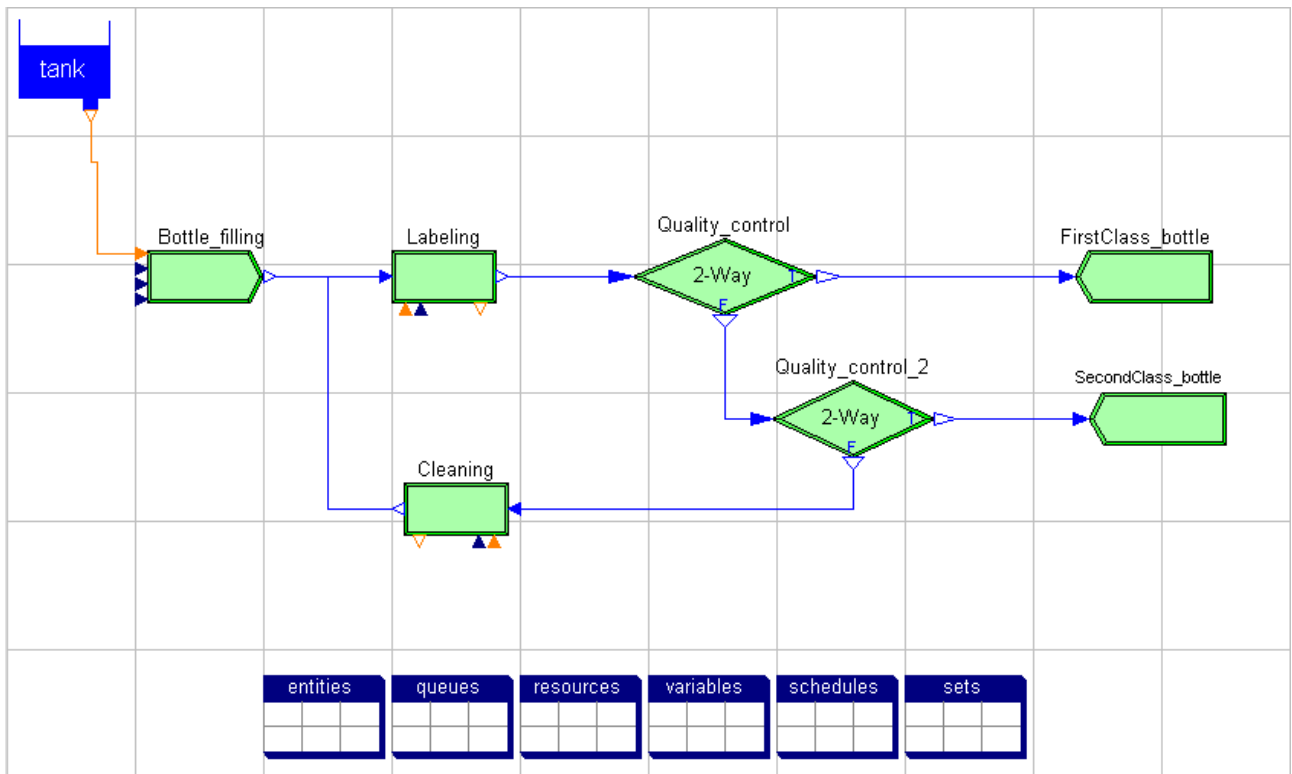
A new Modelica library has been designed and implemented, offering the possibility of modeling DES and hybrid continuous-discrete systems in a simple way. This new library is based on Arena, a simulation environment for DES.

This new library is completely compatible with the rest of the Modelica's components.

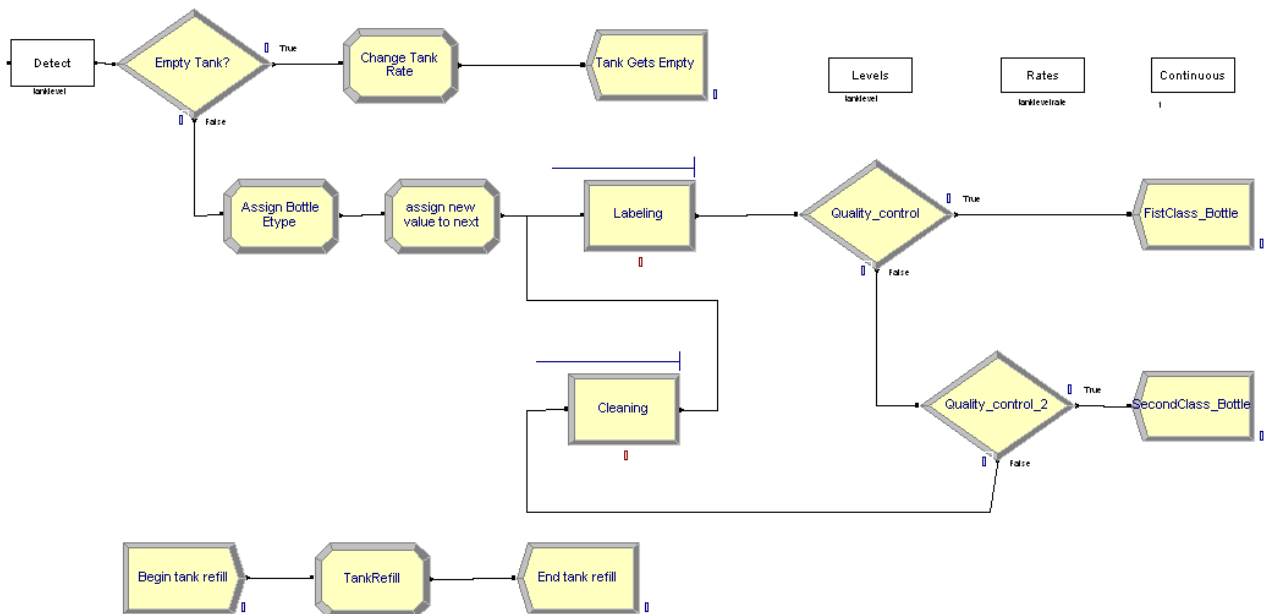
Several experiments have been performed obtaining successful results. Validation has been done using Arena for comparing results.

References

- [1] Kelton W.D, Sadowski R.P, Sturrock D.T. Simulation with Arena (Third Edition). McGraw-Hill, 2004.
- [2] Otter M, Årzén K.-E, Dressler I. StateGraph - A Modelica Library for Hierarchical State Machines. In: Proc. of the 4th Int. Modelica Conference, 2005, pp. 569–578.
- [3] Ferreira J. A, Estima de Oliveira J.P. Modelling Hybrid Systems using Statecharts and Modelica. In: Proc. of the 7th IEEE Int. Conference on Emerging Technologies and Factory Automation, 1999.
- [4] Mosterman P.J, Otter M, Elmqvist H. Modelling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica. In: Proc. of the Summer Computer Simulation Conference, 1998, pp. 314–319.
- [5] Färnqvist D, Strandemar K, Johansson K. H, Hespanha J.P. Hybrid Modeling of Communication Networks Using Modelica. In: Proc. of the 2nd Int. Modelica Conference, 2002, pp. 209–213.
- [6] Remelhe M.A.P. Combining Discrete Event Models and Modelica - General Thoughts and a Special Modeling Environment. In: Proc. of the 2nd Int. Modelica Conference, 2002, pp. 203–207.
- [7] L'Ecuyer P. Software for uniform random number generation: distinguishing the good and the bad. In: Proc of the 33rd conference on Winter simulation, 2001, pp. 95–105.
- [8] L'Ecuyer P, Simard R, Chen E. J, Kelton W. D. An Object-Oriented Random-Number Package with Many Long Streams and Substreams. Operations research, vol. 50, 2002, pp. 1073–1075.
- [9] L'Ecuyer P. CMRG source code web page. <http://www.iro.umontreal.ca/lecuyer/myftp/streams00/> July, 2006.



a)



b)

 Figure 3: Bottle Filling model composed using: a) *ARENALib*; and b) *Arena*

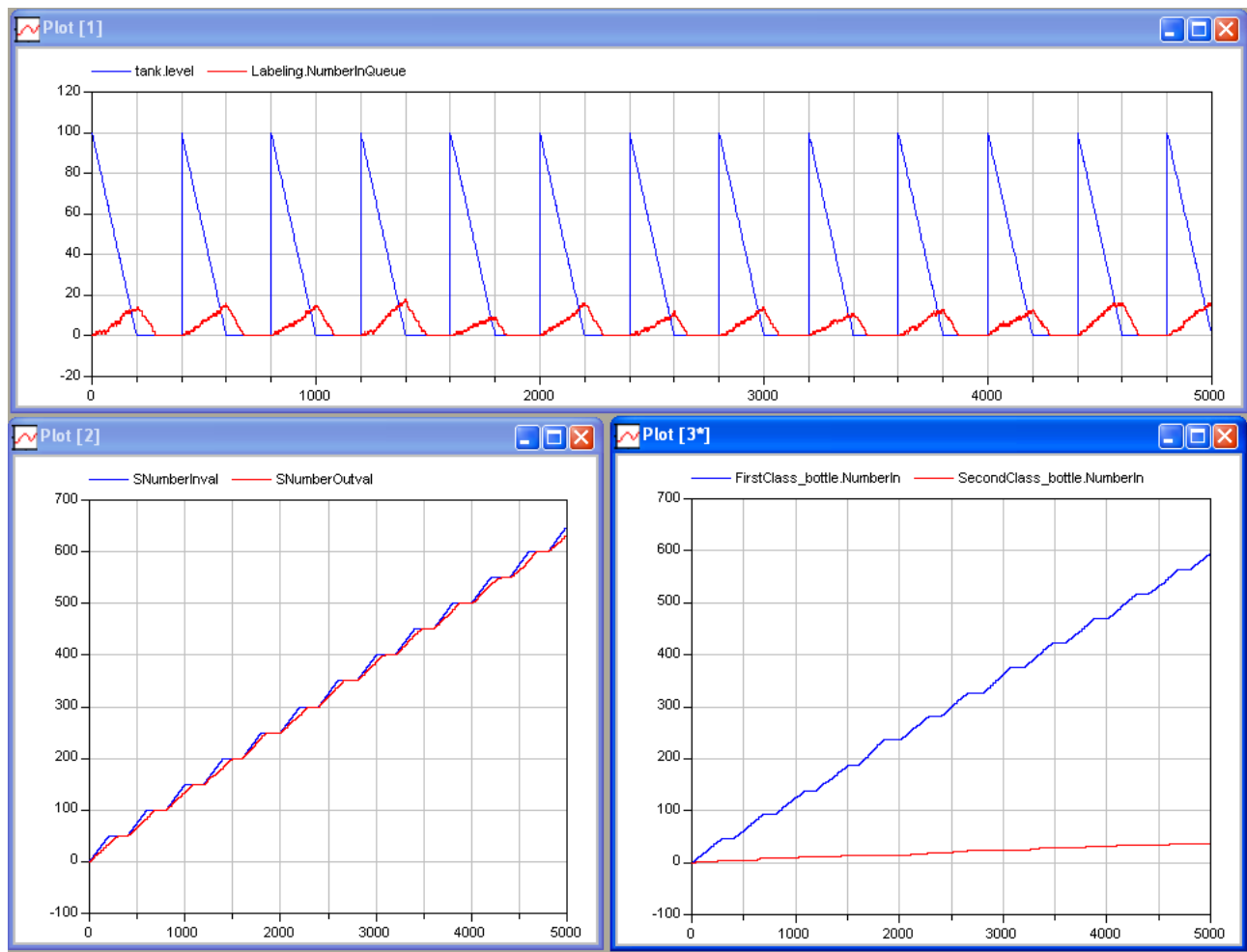


Figure 4: Bottle Filling model result plots using Dymola

Neural Network Library in Modelica

Fabio Codecà Francesco Casella
 Politecnico di Milano, Italy
 Piazza Leonardo da Vinci 32, 20133 Milano

Abstract

The aim of this work is to present a library, developed in Modelica, which provides the neural network mathematical model. This library is developed to be used to simulate a non-linear system, previously identified through a specific neural network training system. The `NeuralNetwork` library is developed in Modelica 2.2 and it offers all the required capabilities to create and use different kinds of neural networks.

Currently, only the *feed-forward*, the *elman*[6] and the *radial basis* neural network can be modeled and simulated, but it is possible to develop other kinds of neural network models using the basic elements provided by the library.

Keywords: neural network, library, simulate, model

1 Introduction

The work described in this paper is motivated by the lack of publicly available Modelica libraries for neural networks. A neural network is a mathematical model, which is normally used to identify a non-linear system. Its benefit is the capability to identify a system also when its model structure is not defined. For such a characteristic, sometimes it is used to model complex non-linear system.

There are different kinds of neural networks in literature and all of them are characterized by a specific architecture or some other specific features. This library takes into consideration only three types of neural networks:

- the *feed-forward* neural network,
- the *elman*[6] neural network, which is a recurrent neural network,
- the *radial basis* neural network,

but the basic elements of the library make possible the construction of any other different neural network.

There are, in literature, different kinds of neural networks, many different algorithms to train them and many different softwares to do this task. For this reason, the library purposefully lacks any function to train a neural network; the training process has to be made by an external program. The MatLab[8] Neural Network toolbox was chosen, during the development and the tests, because it is commonly used and it is extremely powerful; however any other training software can be used.

The library was already used to develop and simulate the neural network model of an electro-hydraulic semi-active damper.

The paper is organized as follows. Section 2 presents the neural network mathematical model: a briefly description about the characteristics of each kind of network, implemented in the library, is provided. Section 3 describes the chosen library architecture and the reasons which guide its implementation. Section 4 shows an example of library use: the entire work process will be explained, from the neural network identification, with an external training software, through the network parameters exchange (from the training software environment to the Modelica one), to the validation of the Modelica model. Last section (5) shows some possibilities of future work, and draws some conclusions.

2 Neural Network model

The neural network mathematical model was born in the Artificial Intelligence (AI) research sector, in particular in the 'structural' one: the main idea is to reproduce the intelligence and the capability to learn from examples, simulating the brain neuronal structure on an calculator.

The first result was achieved by McCulloch and Pitts in 1943[1], when the first neural model was born. In 1962 Rosenblatt[2] proposed a new neuron model, called *perceptron*, which could be trained through examples. A *perceptron* makes the weighted sum of the inputs and, if the sum is greater then a bias value, it

sets its output as '1'. The training is the process used to tune the value of the bias and of the parameters which weight the inputs.

Some studies[3] underline the *perceptron* training limits. Next studies[4], otherwise, show that different basic neuron models, complex neuron networks architecture as suitable learning algorithms, ensure to go beyond the theoretical *perceptron* limits.

Three kinds of neural networks, which are described in the following paragraphs, were taken into consideration in the library: they differ in neuron model and network architecture.

2.1 Feed-forward neural network

The *feed-forward* neural network is the most used neural network architecture: it is based on the series connection of neuron layers, each one composed by a set of neurons connected in parallel. Examine the i -th layer: the layer inputs, u_1, u_2, \dots, u_n , are used by r neurons, which produce r output signals, y_1, y_2, \dots, y_r . These signals are the inputs of the next layer, the $i+1$ -th layer.

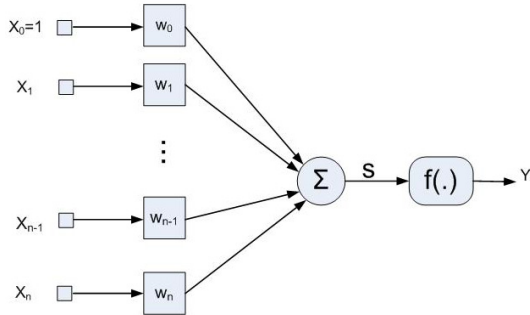


Figure 1: *Standard* neuron model

The neuron used in the *feed-forward* neural network is called *standard* neuron (figure 1). A *standard* neuron maps \mathbb{R}^q into \mathbb{R} ; it is characterized by n inputs, u_1, u_2, \dots, u_n , and one output, y . The first step, which is taken by the neuron, is to compute a weighted sum of the inputs: the result is called *activation signal*

$$s = w_0 u_0 + w_1 u_1 + w_2 u_2 + \dots + w_n u_n,$$

where $w_0, w_1, w_2, \dots, w_n$ are real parameters. w_0 is the neuron *bias* and w_1, w_2, \dots, w_n are the neuron *weights*. The second step is to perform a non-linear elaboration of s , obtaining y . This is computed using a function $\sigma(\bullet)$ ($\mathbb{R} \rightarrow \mathbb{R}$), called *activation function*; it is usually a real function of real variable, monotonically increasing, with a lower and an upper asymptote:

$$s = \lim_{s \rightarrow -\infty} \sigma(\bullet) = \sigma_{inf},$$

$$s = \lim_{s \rightarrow \infty} \sigma(\bullet) = \sigma_{sup}.$$

For this reasons, it is usually called *sigmoid*. Different functions can be used; the most used are:

- $\sigma(s) = \tanh(s)$ (called in MatLab `tansig`);
- $\sigma(s) = \frac{1}{1+\exp^{-s}}$ (called in MatLab `logsig`);

A linear function is also used as *activation function*: it is normally used for the neurons which compose the output layer ($\sigma(s) = s$ (called in MatLab `purelin`)). The *feed-forward* architecture allows to build very complex neural networks: the only constraint is to connect the layer in series and the neurons of a layer in parallel, each of them with the same *activation function*. The first section of the network, which takes the inputs and passes them to the first layer without doing anything, is usually called *Input* layer. The last layer is called *Output* layer and the others are called *Hidden* layer¹.

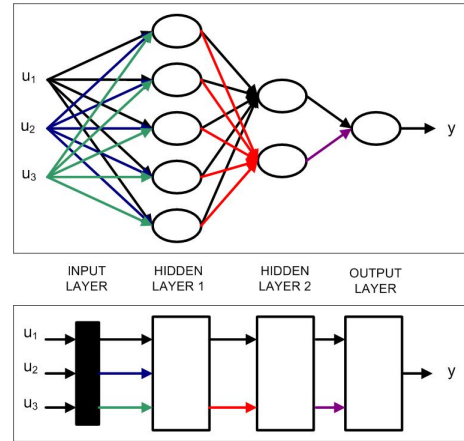


Figure 2: A *feed-forward* neural network structure

An important theoretical result, related to the *feed-forward* neural network, ensures to specify which is the non-linear function class that can be evaluated by a specific neural network. The result is applicable to different kinds of networks: in particular, it involves the *standard* neural network. This network is composed by only two layers: an *Hidden* layer, composed by m neurons², which processes n inputs, u_1, u_2, \dots, u_n , and an *Output* layer, composed by one neuron with a linear *activation function*.

¹this is not an univocal nomenclature; for example, in MatLab, the first neuron layer is called *Input* layer and the others are simply called *layer*.

²all having the same *activation functions*

Theorem 1 (Universal Approximator[4]) Take a standard neural network where $\sigma(\bullet)$ satisfies the following conditions:

1. $\lim_{s \rightarrow \infty} \sigma(s) = 1$,
2. $\lim_{s \rightarrow -\infty} \sigma(s) = 0$,
3. $\sigma(\bullet)$ is continuous.

Taking a function $g(u) : \mathbb{R}^q \rightarrow \mathbb{R}$, continuous on a set I_u compact in \mathbb{R}^q , and an $\varepsilon > 0$, a standard neural network exists which achieves the transformation $y = f(u)$ so that

$$|g(u) - f(u)| < \varepsilon, \forall u \in I_u.$$

2.2 Recurrent neural network (Elman)

A particular type of neural network is the *recurrent* neural network. This network is a dynamical system, in which the output depends on the inputs and the internal state, which evolves with the network inputs. If the internal state is $Z(t)$, the network then agrees to the following relations:

$$\begin{cases} Z(t+1) &= F(Z(t), U(t)) \\ Y(t) &= G(Z(t), U(t)) \end{cases}$$

Recurrent networks are usually based on a feedback loop in the network architecture, but this is not the only way.

In the library, the *Elman*[6] neural network is considered: in this network, the feedback loop is between the output of the *Hidden* layer and the input of the layer itself. This allows the network to learn, recognize and create temporal and spatial models.

An *Elman* neural network is usually composed by two layer connected as shown in figure 3: there is an *Hidden* layer, which is the *recurrent* layer, composed by neurons with an hyperbolic tangent *activation function* ($\sigma(\bullet) = \tanh(\bullet)$), and an output layer, characterized by a linear *activation function*.

As for the *feed-forward* neural network, the *universal approximator* theorem ensures that the *Elman* neural network is an universal approximator of a non-linear function. The only requirement is that the more the function to be estimated becomes complex, the more the number of the neurons, which compose the *Hidden* layer, increases.

The only difference between a *feed-forward* neural network and an *Elman* neural network is the recurrence: this allows the network to learn spatial and temporal models.

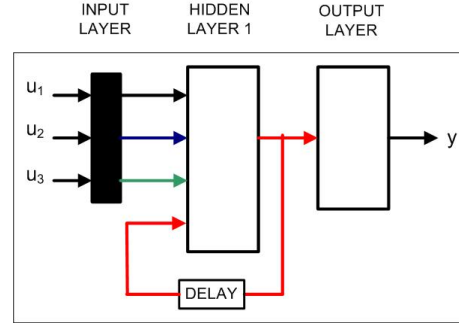


Figure 3: An *Elman* neural network

2.3 Radial basis neural network

The *Radial* basis neural network is used as an alternative to the *feed-forward* neural network. Like this one, it is based on the series connection of layers, each of them composed by a set of neurons connected in parallel. Two are the main differences:

- the number of layers is commonly fixed, with one *Hidden* layer and one *Output* layer;
- the basic neuron is not the *standard* neuron but it is called *radial* neuron.

A *radial* neuron maps \mathbb{R}^q into \mathbb{R} ; it is characterized by n inputs, u_1, u_2, \dots, u_n , and one output, y . The first step took by the radial neuron is to compute an *activation signal*: it differs from the *standard* one because it is not a weighted sum of inputs but it is equal to:

$$s = \text{dist}(\{u_1, u_2, \dots, u_n\}, \{\alpha_1, \alpha_2, \dots, \alpha_n\})b,,$$

where $\alpha_1, \alpha_2, \dots, \alpha_n$ are real parameters, regarding which distances of the inputs are calculated (they are called *centers* of the neuron), b is called neuron *amplitude* and the function $\text{dist}(\{x_1, x_2\}, \{a_1, a_2\})$ computes the euclidean distance between $\{x_1, x_2\}$ and $\{a_1, a_2\}$ ³. The following step is to perform a non-linear elaboration of s , obtaining y . This is made using the function $\sigma(\bullet) = \exp^{-(\bullet^2)}$ ($\mathbb{R} \rightarrow \mathbb{R}$) which is the *radial* neuron *activation function*; it is not a *sigmoid* function but a *bell-shaped* function (figure 4).

As previously remarked, the *radial basis* neural network architecture is commonly fixed: there is an *Hidden* layer, composed by *radial* neurons, and one *Output* layer, composed by a *standard* neuron with a linear activation function (*purelin*). Although the structure of this neural network is more limited, compared to the *feed-forward* one, this is not a limit for its approximator capability.

³ $\text{dist}(\{x_1, x_2\}, \{a_1, a_2\}) = \sqrt{(x_1 - a_1)^2 + (x_2 - a_2)^2}$

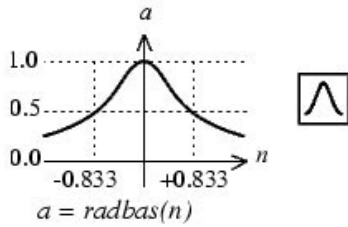


Figure 4: Radial neuron activation function

As for the *feed-forward* and the *Elman* neural networks, the universal approximator theorem ensures that this kind of neural network is an universal approximator of a non-linear function. The only requirement is that, the more the function to approximate becomes complex, the more the number of the neurons which compose the *Hidden* layer increases.

3 NeuralNetwork library

The reason of this library is the lack of an suitable Modelica library, able to simulate a neural network. The aim was to develop a library with the capabilities to create and to simulate such a mathematical model. There are already many different algorithms to train a neural network and many different softwares to do this task so no training algorithm was given. This requires that the training process must be performed by an external software. The MatLab[8] Neural Network toolbox was chosen, during the development and the tests, because it is used commonly and it is extremely powerful; however any other training software can be used. These elements affect some library architectural choices.

The first aim was to give to the users all the elements to create the previously presented neural networks: no constraints were put in for the user, who can create any kind of network architecture without limits. The user himself is directly responsible to use the basic blocks correctly and no checks are performed by the library blocks.

The basic element of the *NeuralNetwork* library was chosen to be a network layer. A layer in a neural network (*NeuralNetworkLayer*) is a set of neurons which are connected in parallel[5]. It is characterized by the following parameters:

- *numNeurons*: it is the number of neurons which compose the layer;
- *numInputs*: it is the number of inputs of the layer;

- *weightTable*: it is a matrix which collects the *weight* parameters (or the *centers* of neurons) used by every neuron of the layer to weight the inputs; its dimension is $[numNeurons \times numInputs]$;
- *biasTable*: it is a vector which collects the *biases* of neurons that compose the layer; its dimension is $[numNeurons \times 1]$;
- *NeuronActivationFunction*: it is the *activation function* used to compute the output by each neuron of the layer. The neurons, which compose a layer, can only have the same *activation function*.

Using a network layer as the basic element has the only limit that the *activation function* of each neuron in a layer must to be the same, but the neural network architectures previously presented don't need this property. Moreover this choice ensures to have an easier data exchange between the neural network training environment and the Modelica one.

This is particularly true when the MatLab Neural Network toolbox is used to train a neural network. As reported in the section 4, in the object used by MatLab to store a neural network, the *weights* (or the *centers* of neuron) and the *bias* of layer are collected in a matrix with the same property of the matrix used to initialize a *NeuralNetworkLayer*.

The library is organized in a tree-based fashion (Figure 5), and it is composed by five sub-packages:

- the package *BaseClasses*: it contains only one element, the *NeuralNetworkLayer*;
- the package *Networks*: it contains some neural networks based on the connection of many *NeuralNetworkLayer*;
- the package *Utilities*: it contains different functions and models used to define some library elements or used itself in the library;
- the package *Types*: it contains the constants used to specify the *activation functions* which characterize a *NeuralNetworkLayer*;
- the package *Examples*: it contains some examples which allow the user to explore the library capabilities.

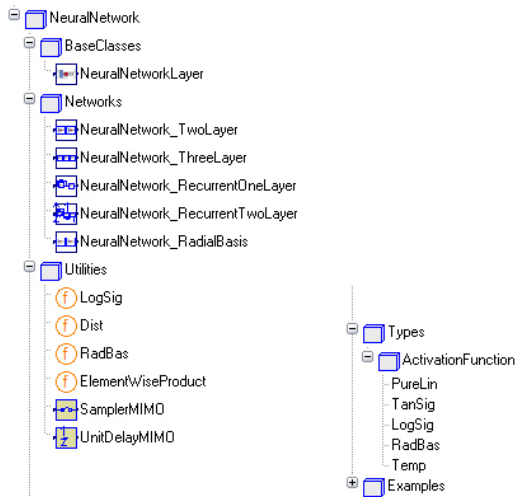


Figure 5: Library structure

3.1 BaseClasses - NeuralNetworkLayer

As previously described, there is only one element in the `BaseClasses` package, the `NeuralNetworkLayer`. This is a block with a MIMO interface, in which the number of inputs is specified through a parameter and the outputs number is the same to the neurons one.

The parameters of the `NeuralNetworkLayer` are:

- *numNeurons*: it is the number of neurons which compose the layer
- *numInputs*: it is the number of inputs to the layer
- *weightTable*: it is the table of the *weights*, if the layer is composed by *standard* neurons, or the table of the *centers* of the neuron, if the layer is composed by *radial* neurons
- *biasTable*: it is the *bias* matrix of the neurons which compose the layer
- *NeuronActivationFunction*: it is the *activation function* of the layer neurons

The *NeuronActivationFunction* characterizes the behavior of the neuron network layer. The parameter can be selected in the set, defined by `NeuralNetwork.Types.ActivationFunction`; the possible choices and behaviors are:

- *PureLin*: the block acts as a layer composed by *standard* linear neurons; the output is equal to the *activation signal* s , which is equal to

$$y = s = \text{weightTable} * u + \text{biasTable}[:,1]$$

- *TanSig*: the block acts as a layer composed by *standard* non linear neurons; the output is equal to the hyperbolic tangent of the *activation signal*

$$y = \text{Modelica.Math.tanh}(s);$$

- *LogSig*: the block acts as a layer composed by *standard* non linear neurons; the output is equal to the value returned by the *LogSig* function:

$$y = \text{NeuralNetwork.Utilities.LogSig}(s);$$

- *RadBas*: the block acts as a layer composed by *radial* non linear neurons; the output is computed with the following steps:

- the euclidean distance between the *centers* of layer neurons and the inputs is evaluated using the function `NeuralNetwork.Utilities.Dist()` with the following parameters: *weightTable*, *matrix(u)*
- the element-wise product between the previous function output and the *bias* matrix is calculated using the `NeuralNetwork.Utilities.ElementWiseProduct` function: this value is the activation signal s
- the output is then evaluated using the specific radial neuron *activation function* (`NeuralNetwork.Utilities.RadBas`);

3.2 Networks

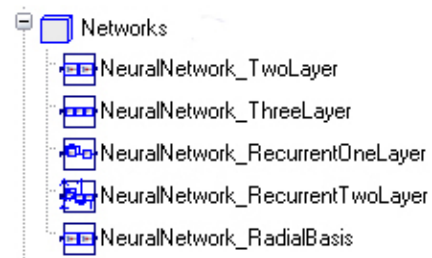


Figure 6: Networks package structure

This package (shown in figure 6) is composed by five blocks: each one represents a neural network. The *feed-forward* neural network and the *radial basis* neural network are easily composed using the `NeuralNetworkLayer` block.

The case of the *Elman* neural network (figure 7 shows the model in Dymola[7]), which in the library is called

NeuralNetwork_RecurrentOne(Two)Layer⁴, is different. In figure 7 the NeuralNetwork_RecurrentOneLayer is shown: the delay block has been introduced to create the recurrence. The parameters of every layer and the parameter of the delay block, which is the *samplePeriod* of the recurrent layer, can be tuned. The *samplePeriod* has to be equal to the input signal sample rate, so that the network can work correctly.

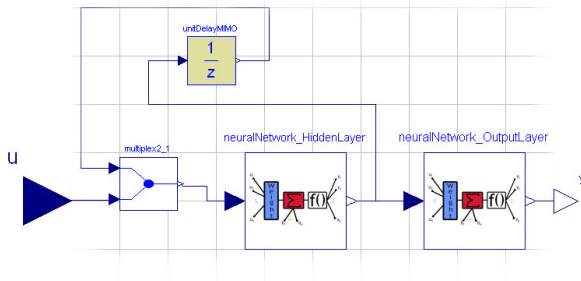


Figure 7: Elman neural network in Dymola

The NeuralNetwork.Utilities.UnitDelayMIMO was introduced to realize the layer feedback: it behaves as the Modelica.Blocks.Discrete.UnitDelay but it has a MIMO interface in place of the SISO one.

3.3 Utilities

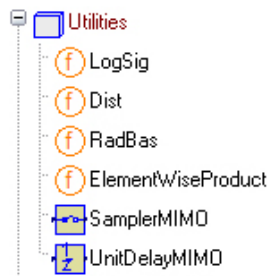


Figure 8: Utilities package

The Utilities package (shown in figure 8) is composed by some mathematical functions and blocks needed to the library to work. In the blocks there are the NeuralNetwork.Utilities.UnitDelayMIMO block, used to model an *Elman* neural network, and the NeuralNetwork.Utilities.SamplerMIMO, used to sample more signals at the same time and used to build the *Elman* neural network example. The mathematical functions instead are used to model a specific *activation function* (*LogSig*

and *RadBas*) or to elaborate signals which are used by neurons to compute the *activation signal* (*Dist* and *ElementWiseProduct* are used by a layer composed by *radial*).

4 An application example

The package Examples contains some instances which allow the user to explore the library capabilities. In this section, an example of how to use the NeuralNetwork library is shown: the entire work process will be explained, from the neural network identification, with MatLab Neural Network toolbox, through the network parameters exchange, to the validation on the model implementation in Modelica.

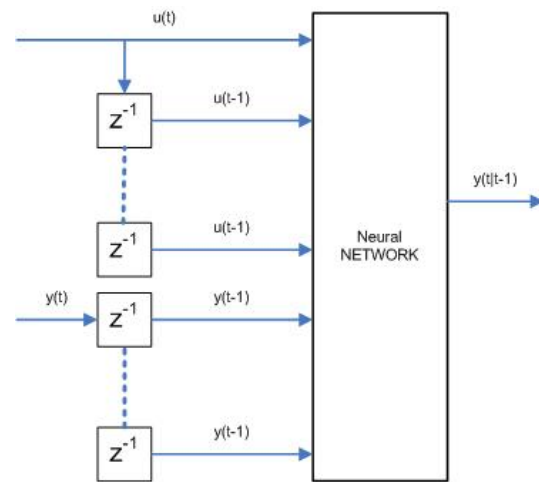


Figure 9: NARX: neural network with external dynamics

The example shown here (which is the model FeedForwardNeuralNetwork placed in Examples package) is about a *feed-forward* neural network with external dynamics. This neural network, shown in figure 9, is a *feed-forward* neural network in which the signals used as inputs are previously delayed. The *feed-forward* neural network with external dynamic, which is normally called *NARX*, performs the following function

$$y(t) = f(u(t) \dots u(t - n_a), y(t) \dots y(t - n_b)).$$

This example shows how to use the elements of the NeuralNetwork library to create a *feed-forward* neural network with external dynamic, where u is a vector composed by two elements, $n_a = 2$ and $n_b = 0$. First of all we have to create the model of the process which has to be identified by the network. We assume that the process is driven by the non-linear function

$$F(t) = (3x(t)x(t-1)x(t-2)) + (y(t)y(t-2)),$$

⁴they differ for the number of recurrent layer

where x and y are the inputs of the system. Note that the process is dynamic because $F(t)$ uses the input values at t time, $t-1$ time and $t-2$. For this reason we choose to use a dynamical *feed-forward* network with 6 inputs:

$$y(t) = f(x(t), y(t), x(t-1), y(t-1), x(t-2), y(t-2)).$$

To train the network is mandatory to have some input signals and the correspondent outputs. The Matlab environment can be used: define the input signals with the following commands⁵

```
t=0:0.01:10;
x=sin(2*pi*t);
y=cos(5*pi*t);
```

and calculate the output signal of the process from the inputs previously defined.

```
for k=3:length(t)
    f(k)=(3*x(k)*x(k-1)*x(k-2));
    f(k)=f(k)+(y(k)*y(k-2));
end
```

After the input and output signals are created, the network has to be built. To construct a *feed-forward* neural network, the command `newff` has to be used. As parameters, the command requires the variances of the inputs, the dimension of the network and the layer *activation functions*. To do this use the following commands

```
var_x = [min(x) max(x)];
var_X = [var_x; var_x; var_x];
var_y = [min(y) max(y)];
var_Y = [var_y; var_y; var_y];
net = newff([var_X ; var_Y], [4 1],
            {'tansig', 'purelin'});
```

Note that `var_X` and `var_Y` are a 3×2 matrix, with one line for $x(t)$, one for $x(t-1)$ and one for $x(t-2)$. To train the network, the input signal matrices have to be created (they are `in_X` and `in_Y`). Some parameters, like the train method and the train epochs number, has to be set and then the function `train` can be used. This is done with the following commands:

```
in_X = [ x ; [0 x(1:end-1)] ;
         [0 0 x(1:end-2)] ];
in_Y = [ y ; [0 y(1:end-1)] ;
         [0 0 y(1:end-2)] ];
net.trainFcn = 'trainlm';
```

⁵when dealing with dynamic *feed-forward* networks it is very important that the sampling time during the simulation be the same as the one used for the network training, otherwise the model will not behave correctly.

```
net.trainParam.epochs = 100;
[net, tr]=train(net, [in_X; in_Y], f);
```

To see how the network has learned the non-linear system the command `sim` can be used

```
f_SIM = sim(net, [in_X; in_Y]);
```

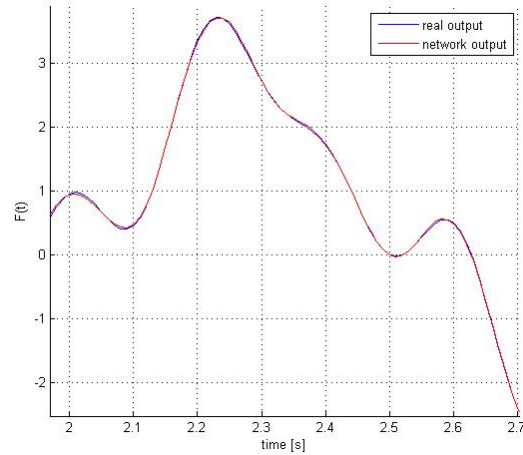


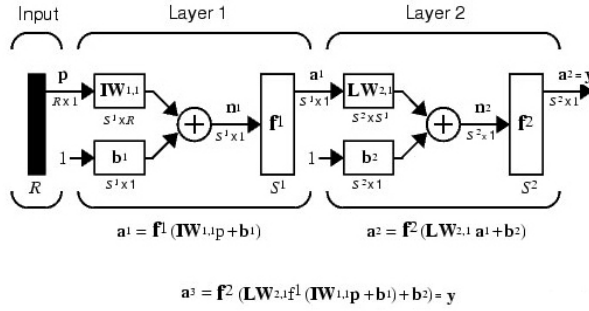
Figure 10: Real process and neural network output comparison

Plotting the real output and the network simulated output (figure 10), we can see that the network has identified the non-linear system very well. Two ways were taken into consideration in order to use the parameters coming from the MatLab environment:

- create a specific script for MatLab (called *extract-Data.m*) which collects the parameters from the environment and creates a text file containing all the information as the Modelica notation and the library requests;
- use the `DataFiles` library which provides some functions to read/write parameters from/into *.mat* files (saved using the `-V4` option).

The `DataFiles` library is a particular implementation supplied by Dymola to manage *.mat* files: this approach was used in absence of a general solution in Modelica.

In this particular example the first way was used. At first, it has to be understood how the MatLab saves the *feed-forward* neural network parameters. Watching the figure 11, which shows how MatLab maps the *weights* and *bias* of the layer on the network object matrices and keeping in mind that the first hidden layer is called *InputLayer* and the others only *Layer*, can be asserted that:

Figure 11: MatLab *weights* and *bias* matrices

- to access to the *weights* matrix of a layer has to be used the command `net.X{1,1}`⁶, where $X=IW$ for the first layer and $X=LW$ for the others; the *weights* matrix is a $[S \times R]$, where S is the neuron number and R the layer inputs number
- to access to the *bias* matrix has to be used the command `net.b{1}`, $[S \times 1]$.

Using this information and the *extractData.m* script, two files, which contain the Modelica definition of the network layers that compose the neural network, were created:

```
extractData('LW.txt', 'OutputLayer',
            net.LW{2,1}, net.b{2}, 'lin')
extractData('IW.txt', 'HiddenLayer',
            net.IW{1}, net.b{1}, 'tan')
```

where 'LW.txt' and 'IW.txt' are the names of the file where the definition of the Modelica neuralNetwork.TwoLayer *OutputLayer* and *HiddenLayer* are stored. The other parameters of the command are the *weights* and the *bias* matrices and the layer *activation function*.

Now it's possible to create this neural network using the Modelica language⁷. At first take a neuralNetwork.TwoLayer block and change its parameters using the results of the previous steps (located in 'IW.txt' and 'LW.txt'). Then, since the neural network expects 6 inputs which have to be externally built, some unit delay blocks (with sample time sets to 0.01, which is the input signals sample time) and a multiplexer must be used.

As last step, build a *.mat* file enclosing the input signals used in MatLab to simulate the neural network. To compare the Modelica output to the MatLab one, enclose the output signals too.

⁶For the index selection please use the MatLab Neural Network toolbox help.

⁷The example model (figure 12) was created in Dymola[7]

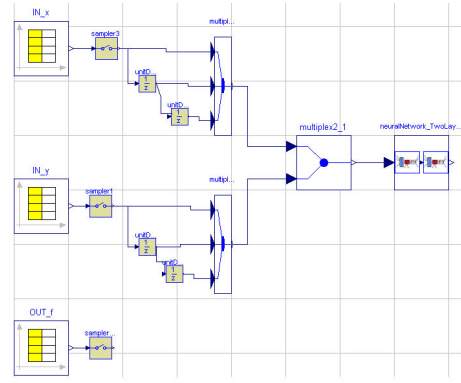


Figure 12: FeedForwardNeuralNetwork example

```
IN_x=[t' , x'];
IN_y=[t' , y'];
OUT_f=[t' , f_SIM'];
save testData_FeedForwardNN.mat -V4
IN_x IN_y OUT_f
```

the figure 13 shows the output of the Modelica simulation and the output of MatLab: see that there is no difference between them.

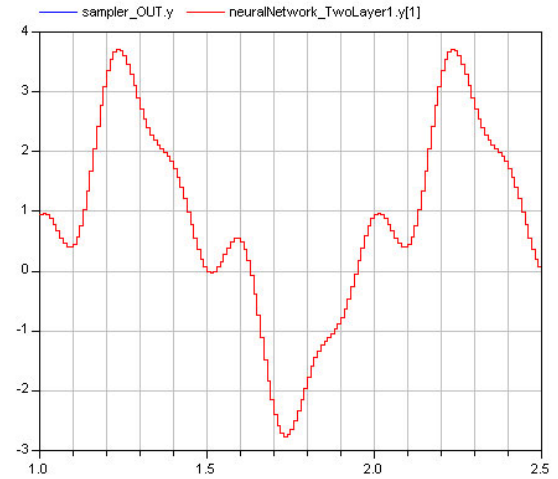


Figure 13: Matlab and Modelica simulation output comparison

Similar examples have been built for the other kinds of networks in the library. They are available in the Examples package, to check their results against the Matlab implementation.

5 Conclusion

A Modelica library, providing the neural network mathematical model is presented. This library is developed to be used to simulate a non-linear system,

previously identified through a specific neural network training system. The `NeuralNetwork` library is developed in Modelica 2.2 and it offers all the required capabilities to create and use different kinds of neural networks.

Currently, only the *feed-forward*, the *elman*[6] and the *radial basis* neural network can be modeled and simulated, but it is possible to build different network structures, by using the basic elements provided by the library. In section 4, a library extension example is shown: a dynamical neural network model is created using the library blocks. The entire work process is explained, from the neural network identification, with an external training software, through the network parameters exchange (from the training software environment to the Modelica one), to the validation of the Modelica model. This lead us to show that there is no difference between the Modelica simulation output and the MatLab one.

The library is publicly available under the Modelica License from the www.modelica.org website.

References

- [1] McCulloch, W. S. and Pitts, W., A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115–133, 1943.
- [2] Rosenblatt, F., The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 1958, 65, 386-408.
- [3] Minsky, M.L. and Papert, S., *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.
- [4] Hornik, K., Stinchcombe, M. and White, H., Multilayer feedforward neural networks are universal approximators, *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [5] Bittanti, S., *Identificazione dei modelli e sistemi adattativi*, Pitagora Editrice, 2002.
- [6] Elman, J. L., Finding structure in time, *Cognitive Science*, 15, 1990, 179-211.
- [7] Dymola, Dynamic Modeling Laboratory, Dynasim AB, Lund, Sweden.
- [8] The Math Works Inc., MATLAB®- The language of Technical Computing, 1997.

The Modelica Multi-bond Graph Library

Dirk Zimmer and François E. Cellier
Institute of Computational Science, ETH Zürich
CH-8092 Zürich, Switzerland
{dzimmer, fcellier}@inf.ethz.ch

Abstract

Bond graphs have established themselves as a reliable tool for modeling physical systems. Multi-bonds are a bondgraphic extension that provides a general approach to modeling all kinds of multi-dimensional processes in continuous physical systems. This paper presents a Modelica library for modeling multi-bond graphs and their application to three-dimensional mechanical systems. A set of bondgraphic models for ideal mechanical components is provided that enables a fully object-oriented modeling of mechanical systems. The wrapping of the bondgraphic models and their representation by meaningful icons gives the mechanical models an intuitive appeal and makes them easy to use. The resulting mechanical systems can be efficiently simulated. Additionally, the continuous mechanical models were extended to hybrid models that allow discrete changes to be modeled that occur in mechanical system as a result of hard impacts.

Keywords: Bond Graph, Multi-bond Graph, Multi-body System, Model Wrapping

1 Introduction

1.1 Introduction to Bond Graphs

If a physical system is subdivided into small components, we observe that these components all exhibit specific behavior with respect to power and energy: Certain components *store* energy like a thermal capacitance; other elements *dissipate* energy like a mechanical damper. An electric battery can be considered a *source* of energy. The power that is flowing between components is distributed along different types of junctions. This perspective offers a general modeling approach for physical systems: bond graphs: [3], [8].

Bond graphs are a domain-neutral modeling tool for continuous system modeling in the field of physics. The actual graph represents the power flows be-

tween the elements of a physical system. The edges of the graph are the bonds themselves. A bond is represented by a “harpoon” and carries two variables: the flow, f , written on the plain side of the bond, and the effort, e , denoted on the other side of the bond [3].

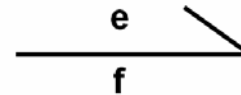


Figure 1: Representation of a bond

The product of effort and flow is defined to be power. Hence a bond is denoting a power flow from one vertex element to another.

The assignment of effort and flow to a pair of physical variables determines the modeling domain. Table 1 below lists the effort/flow pairs for the most important physical domains.

Table 1: Domain-specific effort/flow pairs

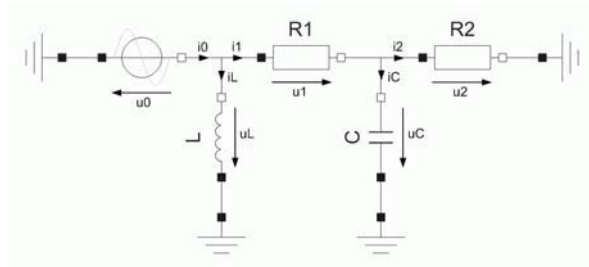
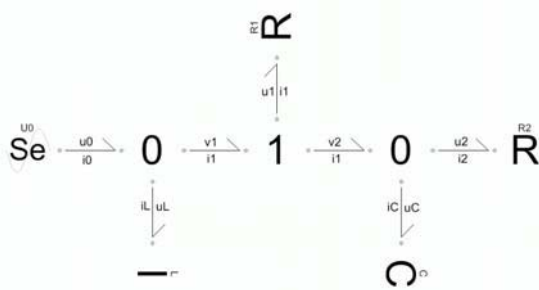
Domain	Effort	Flow
electrical	voltage (u)	current (i)
translational mechanics	force (f)	velocity (v)
rotational mechanics	torque (t)	angular velocity (ω)
acoustics / hydraulics	pressure (p)	volumetric flow (Φ)
thermodynamics	temperature (T)	entropy flow (\dot{S})
chemical	chemical potential (μ)	molar flow (ν)

The vertex elements are denoted by a mnemonic code corresponding to their behavior with respect to energy and power. Table 2 lists the most important bondgraphic elements. The mnemonic code is borrowed from the electrical domain.

Table 2: Mnemonic code of bondgraphic elements

Name	Code	Equation
resistance	R	$e = R \cdot f$
source of effort	Se	$e = e_0$
source of flow	Sf	$f = f_0$
capacitance	C	$f = C \cdot \text{der}(e)$
inductance	I	$e = I \cdot \text{der}(f)$
0-junction	0	all efforts equal $\text{sum}(f) = 0$
1-junction	1	all flows equal $\text{sum}(e) = 0$

Using the bondgraphic methodology, one can e.g. model the electric circuit of Figure 2 by the corresponding bond graph of Figure 3:

**Figure 2:** Schematic of electric circuit**Figure 3:** Bond graph of electric circuit

1.2 The Modelica BondLib

To conveniently model with bond graphs using Dymola, a Modelica library called BondLib [4] has been developed by F.E. Cellier and his students. Using this library, bond graphs can be created in an object-oriented fashion. The library provides a complete set of basic bondgraphic elements as atomic (equation) models. These can be composed graphically to form more complex composite models.

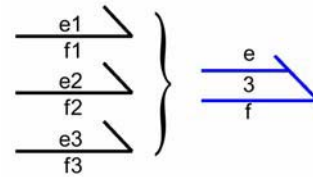
The basic bondgraphic elements and the bonds themselves can be placed on the screen by drag and drop. They can then easily be connected with each

other. Further specifications can be added by means of parameter menus.

Although BondLib is a general Modelica library, its usage is strongly linked to the graphical modeling environment of Dymola [5]. Since bond graphs are a graphical modeling tool, it may be much less desirable to use this library in a purely alphanumerical modeling environment.

1.3 Introduction to Multi-bond Graphs

Multi-bond graphs (sometimes also called vector-bond graphs) are a vectorial extension of the regular bond graphs [2]. They are especially well suited for modeling multi-dimensional processes.

**Figure 4:** Composition of a multi-bond

A multi-bond is composed of a certain number of bonds of either the same domain or at least closely related domains. It is represented by a (blue) double half-arrow as shown in Figure 4. The number shown at the center of the multi-bond denotes its cardinality, i.e., the number of individual bonds included.

This paper presents multi-bond graphs of mechanical systems, which is their primary application. However, multi-bond graphs can be used to model all kinds of multi-dimensional processes. Diffusion processes like heat distribution in a planar electric circuit may be another meaningful application. Yet another interesting field of application could be the modeling of chemical reaction dynamics [2]. Multi-bonds may also be applicable in the field of general relativity [7].

1.4 Advantages of Bondgraphic Modeling

Concerning the modeling of complex physical systems, bond graphs offer a suitable balance between specificity and generality. The interdisciplinary concept of energy and power flows creates a semantic level for bond graphs that is independent of the modeling domain. Thus, basic concepts of physics such as the first rule of thermodynamics can always be verified in a bond graph, independent of its application. This is particularly helpful for intra-domain models that operate in multiple energy domains. In addition, the semantic level helps the modeler avoid many types of modeling errors and find an adequate

solution for his or her task. Modeling by equations is far more flexible and therefore leaves more room for mistakes.

Another advantage of bond graphs is their graphical approach to modeling. Relations can be expressed more naturally by two-dimensional drawings than in a one-dimensional equation code. Also the limitations of the screen (or drawing area) force the modeler to split his model into simple, easily understandable elements.

To us, bond graphs offer also a perfect approach to gaining a profound understanding of the basic principles covering all of physics and to organizing the knowledge concerning specific models. This makes bond graphs extremely valuable and useful for teaching purposes. However, bond graphs are a modeling tool like any other. Everything that can be modeled by bond graphs can also be modeled by other modeling paradigms. Some researchers will find bond graphs a convenient means to organizing their knowledge, whereas other researchers won't.

2 The MultiBondLib

The original BondLib only contains models for regular bond graphs. An additional library is needed to conveniently create models of multi-bond graphs. This paper presents a Modelica library for multi-bondgraphic modeling. It is called MultiBondLib and was designed to bear a strong resemblance to the existing BondLib in structure and composition. All users already familiar with BondLib should therefore be able to quickly acquaint themselves with the new MultiBondLib.

The MultiBondLib features also domain-specific sub-libraries for mechanical systems. These are introduced further on in Section 4. Although mechanics are the major field of application, the basic multi-bondgraphic elements provide a general solution to modeling all kinds of physical processes. These basic elements shall be briefly discussed in the following paragraphs.

Each vertex element of a regular bond graph has its multi-bondgraphic counterpart. The elementary equations of the basic bondgraphic elements remain the same. A transformation to the multi-bondgraphic terminology simply extends the scalar equations to vectorial form. The MultiBondLib provides these multi-bondgraphic counterparts and much more.

Just like the regular BondLib, the MultiBondLib features also causal bonds that are helpful for determining the computational causality of a model and for analyzing computational problems. A stroke de-

notes the side of the bond at which the flow vector is being computed. However, there is no need to ever use causal bonds in Modelica since the computational causality is determined automatically otherwise. Causal multi-bonds are also less convenient than their single-bondgraphic counterparts because the two causal multi-bonds do not cover all possible cases: mixed causality is possible as well, but a standard notation for mixed causality is missing and would probably be more confusing than helpful.



Figure 5: The two causal multi-bonds

In addition, MultiBondLib provides certain elements that are specific to multi-bond graphs. These elements handle the composition, decomposition and permutation of multi-bonds. Also special converter elements have been designed to allow a combination with the classic BondLib.

The cardinality of a multi-bond and its connected elements can be defined for each individual element separately. Consequently, it is possible to create models containing multi-bonds of different cardinality without any additional effort. Yet in many cases, the same cardinality is being used for the entire model. To afford a good usability in such cases, a default model has been developed. The default model is an outer model for all multi-bondgraphic components on the same level of the modeling hierarchy or below that defines the default cardinality of all underlying bondgraphic elements.

3 Mechanical MultiBond Graphs

In the mechanical domain, the bondgraphic effort is identified with the force, f , and the bondgraphic flow is identified with the velocity, v . The corresponding effort/flow pair of the rotational domain is torque, t , and angular velocity, ω . These assignments define the semantic meaning of the bondgraphic elements.

The inductance implements the fundamental equation $f = m \cdot dv/dt$ and represents a storage of kinetic energy. The capacitance represents the storage of potential energy. It can be used to model a spring. A linear bondgraphic resistor models an ideal damper. All of these elements can be rigidly connected using a 1-junction, whereas a force interaction between two neighboring elements is modeled by a 0-junction.

Based upon this standard, various tools for the multi-bondgraphic modeling of mechanical systems have been developed since their invention by Breedveld in 1984 [2]. Unfortunately, most of these tools are incomplete, e.g. only suitable for the description of linear systems; and many are outdated.

To understand the specific difficulties that arise when using a bondgraphic approach to modeling a mechanical system, it is important to note that such systems often exhibit holonomic constraints, i.e., constraints resulting from the topology of the system. No two mechanical bodies can occupy the same space at the same time, and frequently, different mechanical bodies are related to each other by a distance constraint, e.g. they may be connected to each other by a mass-less bar. Holonomic constraints are constraints based on location. However, bond graphs operate on velocities and forces only. Hence mechanical systems cannot be modeled by bond graphs alone. There is a need for additional graphical modeling tools for expressing holonomic constraints between bodies.

The MultiBondLib offers such a link to other modeling tools through sensor elements. Sensors elements “measure” certain bondgraphic variables like effort, flow, momentum (integrated effort) or position (integrated flow). In the MultiBondLib, the signal emitted by the sensor elements is a-causal and simply represents an equality equation (not an assignment). Hence sensor elements are not limited to a single usage. They may serve a number of different purposes:

- to measure bondgraphic variables,
- to convert bondgraphic variables to non-bondgraphic signals,
- to establish algebraic relationships between bondgraphic elements.

It is this latter form of usage that enables us to state holonomic constraint equations: The positional state is derived through sensor elements and influences the dynamical behavior through modulations. It is important to note that sensor elements as well as modulations are neutral with respect to power and energy. Hence the entire energy flow is described by the bond graph alone.

The following paragraphs present two examples that include the applications of these elements in the field of planar mechanical systems. In such a system, the world is restricted to two dimensions. Models for planar mechanic systems are therefore a lot simpler than three-dimensional models. They may serve as a good introduction to multi-dimensional mechanics since such models are much more easily understand-

able. One major simplification in planar mechanics is the fact that the rotational inertance, J , is a constant scalar for all possible orientations, because the axis of rotation is fixed. Thus everything can be comfortably computed using the coordinate vectors of the inertial system. There is no need for transformations between different coordinate systems as is the case in 3D-mechanics. All effort and flow variables of a planar mechanical bond graph can be conveniently resolved in the inertial system.

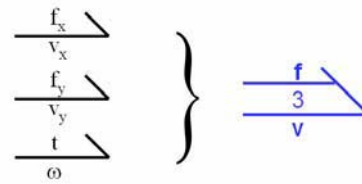


Figure 6: Composition of a planar multi-bond

Hence planar mechanical systems can be described by multi-bond of cardinality three. Figure 6 depicts the planar mechanical multi-bond. The first two bonds belong to the translational domain, whereas the third bond belongs to the rotational domain. The effort vector of a planar multi-bond is then $[f_x, f_y, t]^T$, whereas the corresponding flow vector is $[v_x, v_y, \omega]^T$.

Figure 7 presents the bondgraphic model of a simple planar pendulum.

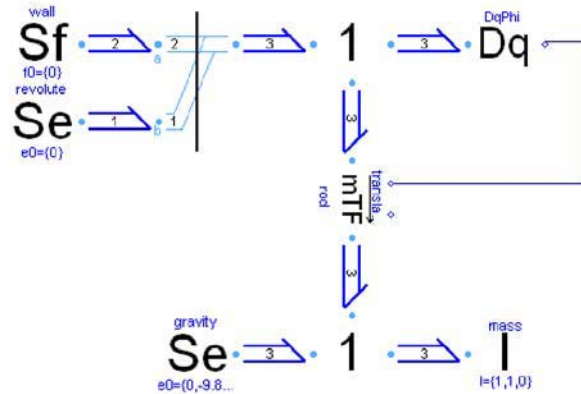


Figure 7: Bond graph of a planar pendulum

The translational position is fixed at the revolute joint by a source of zero flow, Sf , of cardinality two. The revolute joint is free to rotate, i.e., it does not experience any torque. Hence it is modeled by a source of zero effort, Se , of cardinality one. The two multi-bonds are amalgamated to form a general multi-bond of planar mechanical systems of cardinality three.

The mass itself is represented by the 1-junction shown at the bottom of Figure 7. In a 1-junction, the flow variables (velocities) are equal, whereas the effort variables (forces) add up to zero. Hence the 1-junction represents the d'Alembert principle applied to the mass. The forces acting on the mass are the inertial force, I , and the gravitational force, which can be represented by another source of effort, Se , pulling in the negative y -direction.

The mass element is connected to the revolute joint by a mass-less rod describing a positional translation between the body element and the revolute joint. This rod is modeled by a modulated transformer element, mTF , that transforms the angular velocity into a translational velocity, and vice-versa.

This transformation is dependent on the current angle of the revolute joint. Therefore the transformer is modulated by the orientation angle of the revolute joint that is measured by the sensor element, Dq . The a-causal signal connecting the sensor element in combination with the transformer implements the holonomic constraint.

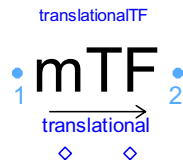


Figure 8: Icon of the modulated transformer. It provides two bondgraphic connectors (denoted by light blue circles) and two signal connectors (denoted by blue diamonds)

Let us take a closer look at the mTF element presented in Figure 8 to get a more profound understanding. The element inherits the bondgraphic effort and flow vectors¹, $(\mathbf{e}_1, \mathbf{f}_1)$ and $(\mathbf{e}_2, \mathbf{f}_2)$, from its two bondgraphic connectors. These two connectors represent the hinges of a mass-less rod. Hence, the transformation between the variable pairs is specified by the parameter vector, \mathbf{d} , that represents the distance vector between the two hinges. A modulation of the transformation is determined by two signals: the current orientation φ and an optional elongation factor $ampl$. The actual model then defines an additional auxiliary variable, \mathbf{r} , and implements the balance equations of a lever:

$$\mathbf{r} = \begin{bmatrix} -\sin(\varphi) & \cos(\varphi) \\ -\cos(\varphi) & -\sin(\varphi) \end{bmatrix} \cdot \mathbf{d} \cdot ampl$$

¹ Please remember the vector composition presented in Figure 6.

$$\mathbf{f}_2[1:2] = \mathbf{f}_1[1:2] + \mathbf{r} \cdot \mathbf{f}_1[3]$$

$$\mathbf{f}_2[3] = \mathbf{f}_1[3]$$

$$\mathbf{e}_1[1:2] = \mathbf{e}_2[1:2]$$

$$\mathbf{e}_1[3] = \mathbf{e}_2[3] + \mathbf{r}^T \cdot \mathbf{e}_2[1:2]$$

This modulated transformer, mTF , is a highly specialized element that hardly makes any sense outside the planar mechanical domain. Such specialized elements are thus provided in corresponding domain-specific sub-libraries, rather than listing them among the basic multi-bond graph elements of the Multi-BondLib.

The large bond graph of Figure 10 represents the model of a second example: a simple model of a crane crab. The corresponding schematic diagram is presented in Figure 9.

Let us refrain from offering a detailed explanation of the model. Instead we shall take a look at the overall structure. The reader may observe the a-causal signals in Figure 10 that flow alongside the actual bond graph. These signals contain the variables for the current position and orientation. Whereas the multi-bond graph models the dynamics of the system, the signals handle the system's positional state.

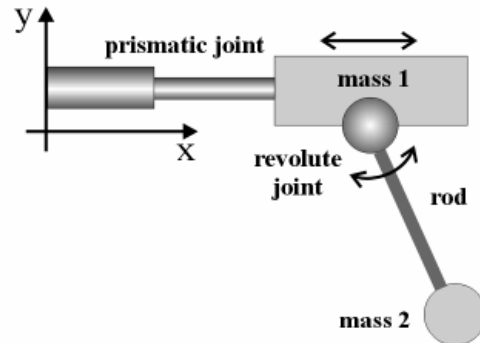


Figure 9: Schematic diagram of a crane crab

Furthermore, multi-bond graphs of mechanical systems tend to become very large, since the sheer generality of the bondgraphic approach does not allow a more specific representation of larger mechanical elements. This makes the resulting bond graph hard to read and understand. It is therefore helpful to separate the bond graph into specific mechanically meaningful subparts as indicated by the gray rectangular frames of Figure 10. These subparts can then be represented by composite models that contain a wrapped version of the underlying subsystem multi-bond graph. The resulting model is presented in Figure 11 and is now easily understandable.

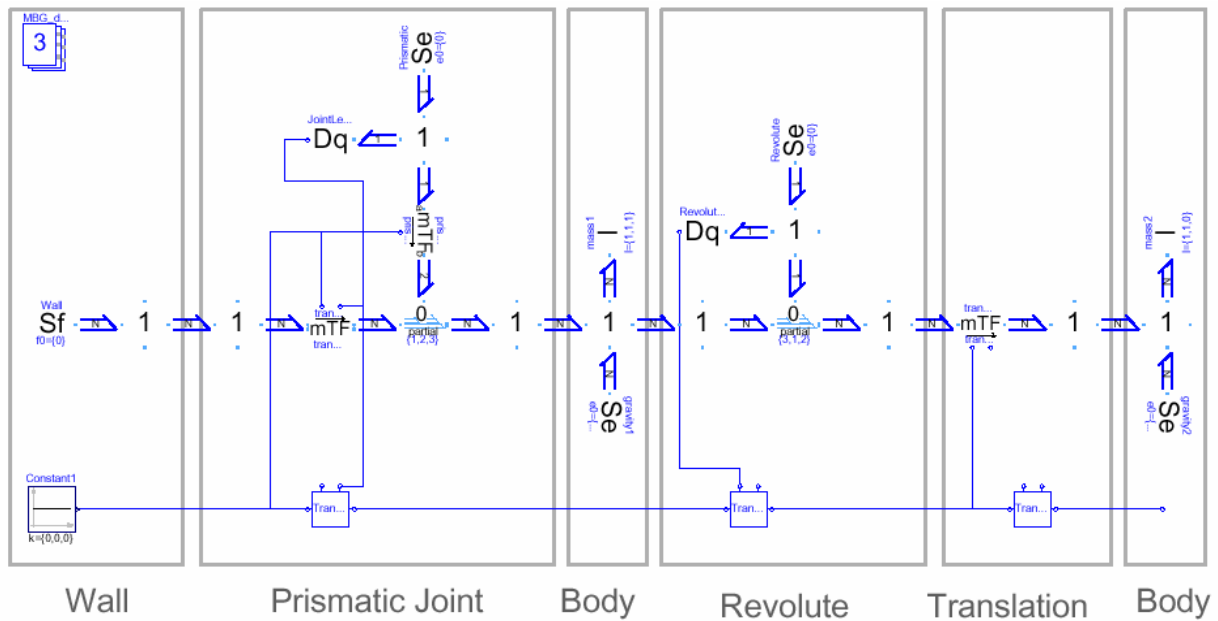


Figure 10: Multi-bond graph of a crane crab

The wrapping technique offers the better of two worlds: It provides an easy-to-use model at the top level, whereas a look inside the model reveals a familiar bondgraphic explanation.

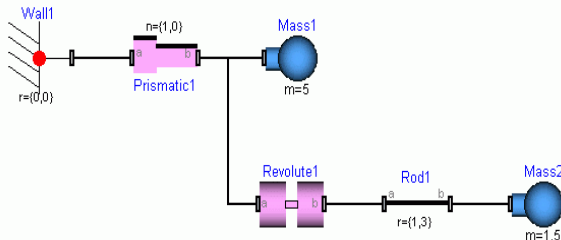


Figure 11: Multi-body diagram of a crane crab

4 The Mechanical Sub-libraries

Two libraries for the modeling of mechanical systems have been developed, one for planar mechanics: “PlanarMechanics,” and the other for 3D-Mechanics: “Mechanics3D”. Both libraries are included as sub-packages within the MultiBondLib. The following discussion will now focus on the components of the Mechanics3D library. However, most of the subsequent descriptions hold for the planar mechanical library as well.

Similar to the Multi-body systems library of M. Otter [9], the Mechanics3D library offers models for an extensive set of mechanical components. It contains models for rigid parts such as bodies and rods,

as well as models for different kinds of joints such as revolute joints or prismatic joints. A spring and a damper are typical examples of force elements for which models are offered in the library as well. In addition, models of ideal rolling objects such as wheels or marbles are provided. All of these elements can be further specified by parameter menus and feature a suitable animation.

Due to the similarity in structure and design, users of the standard Modelica MultiBody library will find the Mechanics3D library very easy to use. The Mechanics3D library represents both a subset and a superset of the MultiBody library. Yet in contrast to the standard MultiBody library, all mechanical models of the new Mechanics3D library are based upon wrapped bondgraphic models. A look inside the models reveals a bondgraphic explanation.

4.1 Connector Types

In this section, we discuss the selection of connector variables for the mechanical components in Mechanics3D. This selection is defined by the process of wrapping. Therefore all bondgraphic variables have to be part of the connector. These are:

- the force vector, \mathbf{f} , (flow variables),
- the torque vector, \mathbf{t} , (flow variables),
- the velocity vector, \mathbf{v} , (potential variables),
- the angular velocity vector, $\boldsymbol{\omega}$, (potential variables);

where the variables of the rotational domain are resolved with respect to their corresponding body system. All other variables are resolved in the inertial system. Also the variables of the positional signals are part of the connector:

- the position vector, \mathbf{x} , (potential variables),
- the orientation matrix, \mathbf{R} , (potential variables).

Please note that the connector contains redundant information. The variables \mathbf{v} and $\boldsymbol{\omega}$ are derivatives of \mathbf{x} and \mathbf{R} . Also \mathbf{R} itself is 3×3 orientation matrix and thus a redundant way of expressing the current orientation.

Although the Mechanics3D library and the MultiBody library are very similar, the Mechanics3D library defines its own slightly different connectors. Hence it is not possible to combine the components of these two libraries without additional tools.

4.2 Kinematic Loops

The redundancy of the connector causes problems when components are connected in a circular fashion, as this is often the case in kinematic loops. A standard connection statement leads then to a singularity in the model. To overcome these difficulties, a second alternative connection statement is needed that contains only non-redundant information. For each kinematic loop, one standard connection has then to be replaced by its non-redundant counterpart.

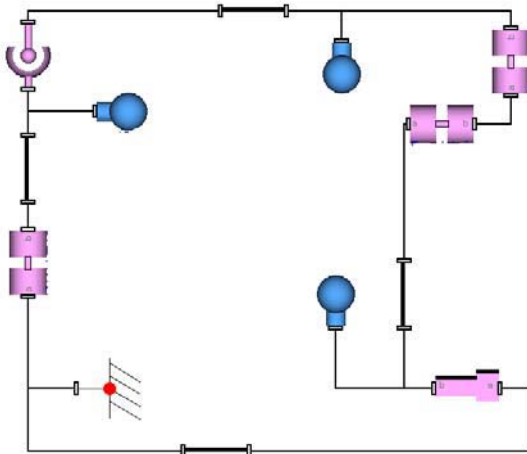


Figure 12: Multi-body diagram of a 3D kinematic loop

However, the user of the library does not need to worry about these details, because the replacement is achieved automatically. To implement the necessary preprocessing of the model, a set of special Modelica

functions² has been used. These are the same methods that have also been applied in the case of the standard MultiBody library (see [9]).

Figure 12 shows the Mechanics3D model of a 3D kinematic loop. No special cut joints have been used, and the elimination of redundant equations and redundant states takes place automatically. Whereas the automatic elimination algorithm works in most cases, planar loops within a 3D environment still require special treatment. A special cut joint has been developed for this purpose.

4.3 An Example: The Bicycle

This example was created by components of Mechanics3D and presents the model of an uncontrolled ideal rolling bicycle. Such a bicycle has seven degrees of freedom on the level of position and three degrees of freedom on the level of velocity. It is a partially stable system for a specific range of driving velocities. A nice description of the dynamic behavior of a bicycle can be found in a paper by K. Åström et al. [1]. The relevant parameter values for mass and geometry of this specific model are taken out of a paper by Schwab et al. [11]. The described bicycle is self-stabilizing for driving velocities between $4.3ms^{-1}$ and $6.1ms^{-1}$.

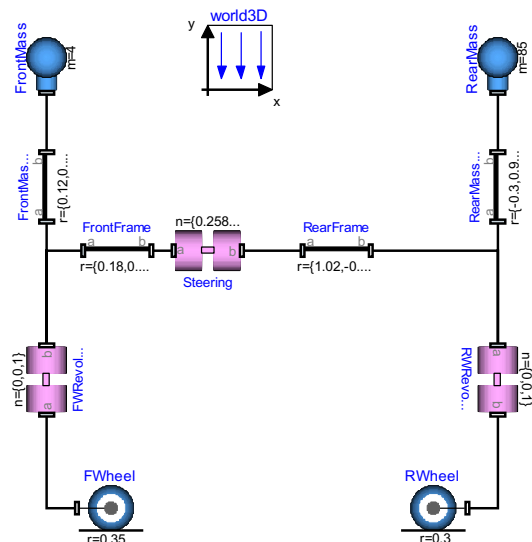


Figure 13: Multi-body diagram of a bicycle

Figure 13 depicts the multi-body diagram of the bicycle model. Although no closed loop is visible in

² These are: `defineRoot()`, `definePotentialRoot()`, `defineBranch()`, `isRoot()`, and `rooted()`.

the multi-body diagram, the model contains a closed kinematic loop, since both wheels are connected to the road. The Mechanics3D library offers an outer world3D model that is used in exactly the same fashion as the corresponding model of the standard MultiBody library.

The selected state variables are the cardan angles of the rear wheel and their derivatives. These three cardan angles represent the orientation on the plane, the lean of the rear frame, and the roll angle of the rear wheel. Additional state variables are the position of the rear wheel on the plane, the angle of the steering joint, and the angle of the front revolute joint. Each of the two wheels defines one holonomic constraint equation that prevents the bicycle from sinking into the road.



Figure 14: Animation model of the bicycle

Figures 14 and 15 show the results of the simulation. The lean angle can be examined in a plot window, and the bicycle is nicely animated within Dymola.

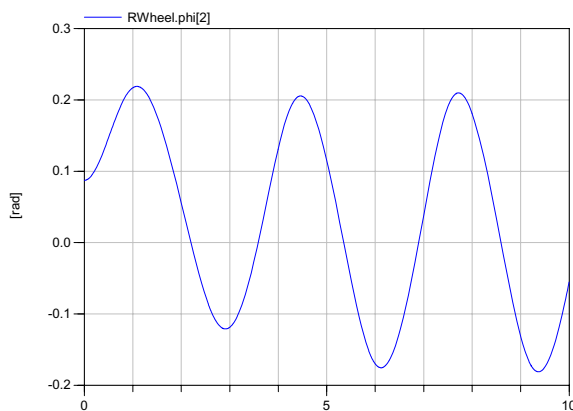


Figure 15: Plot of the lean angle

4.4 Run-time Efficiency

The selection of the state variables is of major importance for the efficiency of the resulting simulation. Usually this selection is automatically achieved by Dymola. Nevertheless variables can be suggested to the simulation program via the advanced Modelica attribute: “StateSelect”. These variables are the states of a joint’s relative position and velocity. Each joint is declaring state variables, unless there is a kinematic loop.

In the latter case, there are often many equivalent sets of state variables. Dymola uses the feature of dynamic state selection, where the state variables are chosen at run time. This offers the most flexible solution but leads to a slower simulation and to many additional equations that are often unnecessary. Hence it is strongly recommended to determine a static set of state variables manually in such a case whenever this is feasible.

All mechanical components that contain potential state variables have the option of a manual enforcement of their state variables via the parameter menu. To this end, the Boolean parameter “enforce-States” needs to be activated.

The Mechanics3D library and the standard MultiBody library contain elements for the same purpose that can be used in very similar ways. It is therefore easily possible to translate the model for a mechanical system from one library to the other. Both libraries are examined with respect to their run-time efficiency. The complexity of the resulting systems of equation and the computational effort of the simulation are compared for a given set of examples. The results of this examination are presented in Table 3. There is hardly any difference. In fact, the generated equations of both libraries are very similar. There is only one remarkable difference: In contrast to the Mechanics3D library, the translational velocity is not part of the connector variables of the standard MultiBody library. The elements are only connected by the translational position. The equations for the translational velocities are then derived (when necessary) by differentiation. In the bondgraphic models of the Mechanics3D library, these equations are explicitly stated.

Table 3 lists the sets of non-linear equations for a given set of experiments. The number of integration steps is counted for a simulation period of 10 seconds. The simulation method was Dassl with a tolerance of 10^{-4} . A * indicates that the parameters of the experiment setup differ slightly between the two libraries.

Table 3: Comparison of the two mechanical libraries

Experiment	MultiBody		Mechanics3D	
	Non-linear equations	Integr. steps	Non-linear equations	Integr. steps
Double Pendulum	0	549	0	549
Crane Crab	0	205	0	205
Gyroscopic Exp. with Quaternions	0	24438	0	25574
Planar Loop	2	372	2	372
Centrifugal	{2,2}	70	{2,2}	70
FourBar	5	446	5	625
Loop*				
Bicycle*	1	97	1	84

5 Conclusions and Further Work

A Modelica library for convenient multi-bond-graphic modeling has been developed. It provides a general solution to modeling all kinds of multi-dimensional processes in continuous physical systems. Multi-bond graphs offer a good framework for modeling mechanical systems. The bondgraphic methodology proved to be powerful enough for modeling all important ideal subparts of mechanical systems accurately and efficiently.

The resulting libraries for mechanical systems PlanarMechanics and Mechanics3D provide an extensive set of component models. These domain-specific models have an intuitive appeal and are easy to use. They consist in wrapped bondgraphic models, and a closer look reveals a bondgraphic explanation of their behavior. Also quality and efficiency of the resulting solution are not impaired by the bondgraphic modeling technique. The selected state variables are chosen wisely, and the resulting systems of equations can be solved fast and accurately.

Furthermore, a third library for mechanics has been developed and included in the MultiBondLib. It is called “Mechanics3DwithImpulses” and represents an extension of the Mechanics3D library. The existing purely continuous mechanical models were extended by their corresponding impulse equations to hybrid models that contain an additional discrete event part.

This library was designed to model the behavior of mechanical systems in situations of hard ideal impacts. The type of mechanical system is thereby not limited at all. Impacts can be modeled between single objects, as well as between kinematic loops (e.g. a car suspension). The provided parameter set for

the impact characteristics includes also specifications for elasticity and friction.

Sadly, the Modelica support for discrete event modeling [10] is not yet fully sufficient. Hence the resulting solution leads partially to models that are unnecessarily complicated in execution and design. Nevertheless the solution is fine and workable for small scale models. Figure 16 shows a simple example of a mechanical system modeled using the extended 3D library.

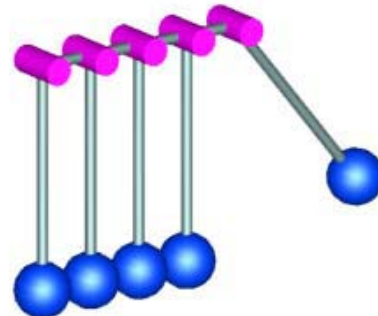


Figure 16: An implementation of Newton's cradle

References

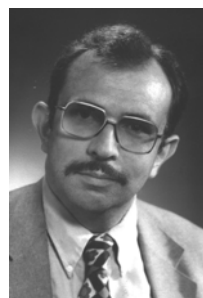
- [1] Åström, K.J., R.E. Klein and A. Lenartsson, “Bicycle Dynamics and Control: Adapted Bicycles for Education and Research,” *IEEE Control Systems Magazine*, **25**(4), pp.26-47, 2005.
- [2] Breedveld, P.C., *Physical Systems Theory in Terms of Bond Graphs*, Ph.D. dissertation, University of Twente, The Netherlands, 1984.
- [3] Cellier, F.E., *Continuous System Modeling*, Springer-Verlag, New York, 755p, 1991.
- [4] Cellier, F.E. and A. Nebot, “The Modelica Bond Graph Library,” *Proc. 4th International Modelica Conference*, Hamburg, Germany, Vol.1, pp.57-65, 2005.
- [5] Dynasim AB, *Dymola Users' Manual*, Version 6.0, Lund, Sweden, 2006.

- [6] Elmqvist, H., M. Otter and J. Díaz López, "Collision Handling for the Modelica MultiBody Library." *Proc. 4th International Modelica Conference*, Hamburg, Germany, Vol.1, pp.45-53, 2005.
- [7] Gussn, N.M.N, *On the Bondgraphic Power Postulate and its Role in Interpreting the 1905 and 1915 Relativity Theories*, MS Thesis, Dept. of Electr. & Comp. Engr., University of Arizona, Tucson, AZ, 1994.
- [8] Karnopp, D.C., D.L. Margolis and R.C. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems*, 4th edition, John Wiley & Sons, New York, 576p, 2006.
- [9] Otter, M., H. Elmqvist and S.E. Mattsson, "The New Modelica MultiBody Library," *Proc. 3rd International Modelica Conference*, Linköping, Sweden, pp.311-330, 2003.
- [10] Otter, M., H. Elmqvist and S.E. Mattsson, "Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle," *Proc. IEEE International Symposium on Computer Aided Control System Design*, Hawaii, pp.151-157, 1999.
- [11] Schwab, A.L., J.P. Meijaard and J.M. Papadopoulos, "Benchmark Results on the Linearized Equation of Motion of an Uncontrolled Bicycle" *KSME Journal of Mechanical Science and Technology*, 19(1), pp.292-304, 2005.
- [12] Zimmer, D., *A Modelica Library for MultiBond Graphs and its Application in 3D-Mechanics*. MS Thesis, ETH Zurich, Switzerland, 2006.

Biographies



Dirk Zimmer received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. He gained additional experience in Modelica and in the field of modeling mechanical systems during an internship at the German Aerospace Center DLR 2005. Dirk Zimmer is currently pursuing a PhD degree with a dissertation related to computer simulation and modeling under the guidance of Profs. François E. Cellier and Walter Gander. His current research interests focus on the simulation and modeling of physical systems with a dynamically changing structure.



François E. Cellier received his BS degree in electrical engineering in 1972, his MS degree in automatic control in 1973, and his PhD degree in technical sciences in 1979, all from the Swiss Federal Institute of Technology (ETH) Zurich. Dr. Cellier worked at the University of Arizona as professor of Electrical and Computer Engineering from 1984 until 2005. He recently returned to his home country of Switzerland. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 and a second textbook on Continuous System Simulation in 2006, both with Springer-Verlag, New York. 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.

Session 5d

Electric Systems and Applications 2

The SmartElectricDrives Library – Powerful Models for Fast Simulations of Electric Drives

Johannes Vinzenz Gragger Harald Giuliani Christian Kral
 Thomas Bäuml Hansjörg Kapeller Franz Pirker
 Arsenal Research

Giefinggasse 2, 1210 Vienna, Austria

phone +43-50550-6210, fax +43-50550-6595, e-mail: johannes.gragger@arsenal.ac.at

Abstract

In this work the SmartElectricDrives (SED) library Release 1.0.1 is presented. The SED library is a tool for simulating and tuning of electric drives in complex electromechanical systems. This library is specifically designed for simulations of electric drive systems with a *Modelica* development platform. Besides a variety of elementary components for simulating modern electric drives the SED library also contains powerful 'ready-to-use' drive models. These 'ready-to-use' drive models include all features and characteristics of full modern electric drives in only one single component. Furthermore, the SED library facilitates simulations on different levels of abstraction. By choosing the right level of abstraction the user can save processing power and therefore computing time. This work outlines all the specific features and design considerations of the SED library. Three simulation examples are presented. Based on the results of these examples the performance of the applied SED models is assessed and analysed.

Keywords: electric drives; electromechanical systems; quasi stationary simulation; signal bus systems; parameter estimation functions;

1 Electric drives applications – the scope of the SED library

The SED library facilitates simulations of any electric drive application. Drives such as machine tools, robotics, mining drives, traction drives and auxiliary drives in vehicles and hybrid electric vehicles (HEV) are good examples for typical applications [1], [2], [3]. These applications require high static control preci-

sion, fast dynamic response, overload capability and sometimes speed ranges significantly above nominal speed. Conventional asynchronous induction machines used for such high performance applications are usually controlled by exploiting the principle of field oriented control (FOC) [4]. FOC can also be applied to synchronous induction machines and is widely used in respective drive applications. Since FOC is the most popular method to control ac machines it is implemented in all 'ready-to-use' ac drive models in the current version of the SED library.

2 Structure of the SED library

All components needed for modern electric drives can be found in the SED library. In Figure 1 the arrangement of the packages is shown.

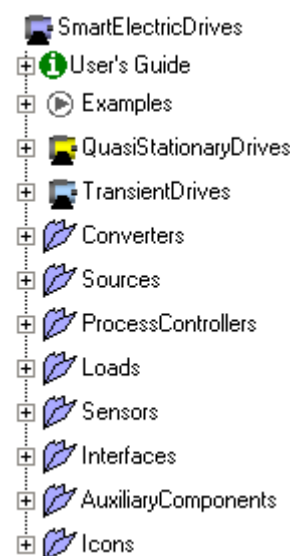


Figure 1: Screenshot of the SED library in the *package browser*.

2.1 The QuasiStationaryDrives package

In the QuasiStationaryDrives package, shown in Figure 2, only models of full torque controlled drives with a dc supplied converter are included. These models are designed such way that all electrical transients in the machines are neglected. Mechanical transients due to the inertia of the rotor are considered, however. Consequently these models cannot be used for the simulation of electric transient effects such as current spikes due to converter switching, for instance. However, as long as only the energy consumption or the efficiency of the drive is of interest it is very advantageous to use QuasiStationaryDrives models because they are remarkably faster than models considering electrical transient effects in machines. Another benefit of QuasiStationaryDrives models compared to conventional drive models is that current controllers, flux controllers and torque controllers do not have to be parameterized since these components do not appear in the quasi stationary equations of electrical drive systems. It can be shown that if electrical transient effects get neglected in electrical machines then simple feed forward control can achieve a system performance that can only be reached by feed back control in systems, which take the electrical transients into account. This fact leads to shorter simulation times of the QuasiStationaryDrives models compared to their counterparts in the TransientDrives package.

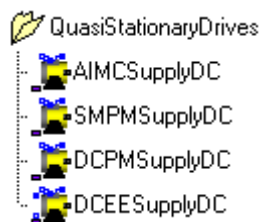


Figure 2: Screenshot of the QuasiStationaryDrives package in the *package browser*.

2.2 The TransientDrives package

The TransientDrives package, shown in Figure 3, contains all drive components that are needed to build a simulation of an electric drive considering transient electric effects in machines. This package is split up into four machine type specific packages containing 'ready-to-use' models of full torque controlled drives as well as the respective elementary

control components of these models. Each 'ready-to-use' drive model consists of a dc/dc or dc/ac converter considering power balance, a machine model and the elementary drive components, such as controllers, a measurement device, a flux model, bus connectors, etc. Figure 4 shows the internal set-up of the 'ready-to-use' asynchronous induction machine drive model. In order to facilitate a very easy switching between QuasiStationaryDrives models and TransientDrives models the connector naming and coding is standardized.

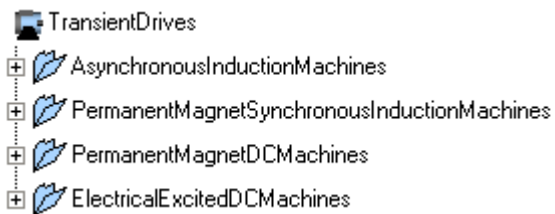


Figure 3: Screenshot of the TransientDrives package in the *package browser*.

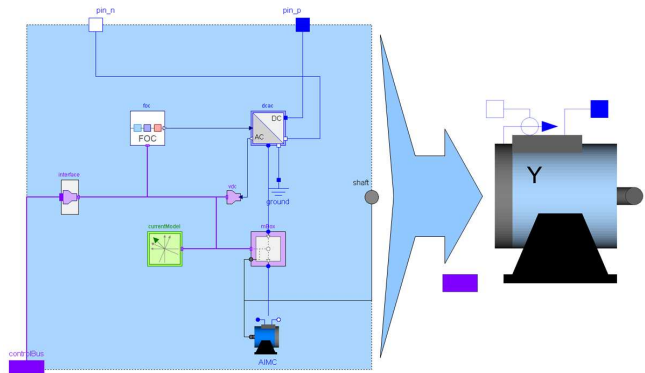


Figure 4: Screenshot of the TransientDrives model of a dc supplied asynchronous induction machine.

2.3 The AuxiliaryComponents package

In the AuxiliaryComponents package a number of primary transformations such as space phasor transformations and line/phase transformations are available. Moreover, in this package the user has some functions for controller parameter estimation at ones disposal. The opened sub-package containing these functions is shown in Figure 5.

These controller parameter estimation functions help the user to adjust the whole set of controller parameters in speed controlled drive simulations for any machine type. Since drive control systems have a

cascaded structure deploying many elementary controllers it can be quite time consuming to find proper values for this big set of controller parameters. By using the provided parameter estimation functions the process of controller adjustment can be accelerated considerably. With regard to specific controller optimization criteria the estimation functions generate controller values based on the parameters that define the machine model such as nominal current, nominal speed, stator resistance etc. Furthermore, the control system behaviour can be tuned by the choice of specific dynamic gains in the estimation functions.

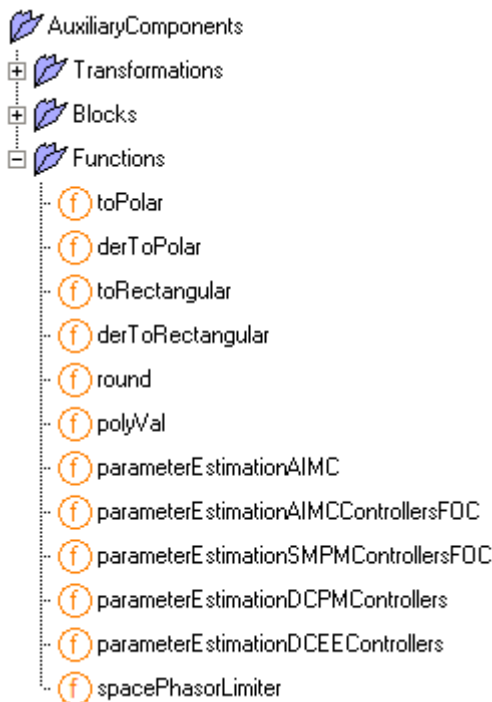


Figure 5: Screenshot of the AuxiliaryComponents.Functions package in the *package browser*.

2.4 The Examples package

A large selection of examples for applications of the SED library components is given in the Examples package shown in Figure 6. In order to ease starting up with the SED library eight specific tutorial examples are included in the Examples package. Step by step, these tutorial examples explain the most important models and their correct application in drive simulations. More sophisticated examples can be found in the Examples.AutomotiveApplications package. In this package some possibilities for the use of electric drives in vehicular applications are presented. Further examples are included in order to give

details concerning the correct use and application of major SED library components such as controllers, converters, sources and loads.

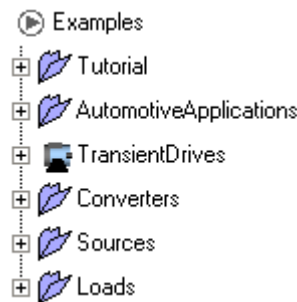


Figure 6: Screenshot of the Examples package in the *package browser*.

2.5 Further important packages

Apart from models representing full torque controlled drive setups the SED library also offers some different models of dc energy sources. There are two battery models, a supercap model and a proton exchange membrane (PEM) fuel cell model available. These models can be found in the Sources package.

The Converters package contains converters on two different levels of abstraction. On one hand the user can choose the so called *power balance converters* and on the other hand there are converters modeled with ideal switches available. *Power balance converters* are designed for simulations in which switching effects do not have to be considered. Their big plus is that simulations work much faster with these models since the calculation effort for the power balance equation is much smaller compared to processing a large number of switching events.

In the Sensors package there are different meters for the generation of specific measuring values available. One important component in this package is the RMS model, which transforms an instantaneous signal to the respective RMS value within a certain measuring period.

The Load package contains different models that can be used for electric power dissipation. The components of this package allow the simulation of dc power loads and the modeling of constant or variable efficiencies in dc circuits.

3 The bus concept

In order to group control signals and measuring signals most SED components are featured with a bus connector. For the internal use of each type of machine control system in the *TransientDrives* package there is a specific internal bus system available since each type of machine has a particular set of control parameters and variables. There is also a general *ControlBus* connector in the *Interfaces* package available that is used to build an external bus system by connecting torque controlled drive systems with further controllers, such as speed controllers, position controllers or drive strategy controllers. Figure 7 illustrates that the external bus system and the internal bus system must be connected via an *Interface* that is also available in the SED library.

A big advantage of bus concepts in *Modelica* is that when programming in a *diagram layer* the model stays manageable because the number of connections gets minimized. Another plus is that during the simulation of the model all variables that are on the bus get grouped together in the *variable browser* of the *simulation tab*. The buses used in the SED library are designed such way that the most important variables for basic analyses of the drive system can be selected on first sight in the variable browser. Some of these very important variables are the shaft speed, the shaft angle, the stator current, the stator voltage and the dc-link voltage.

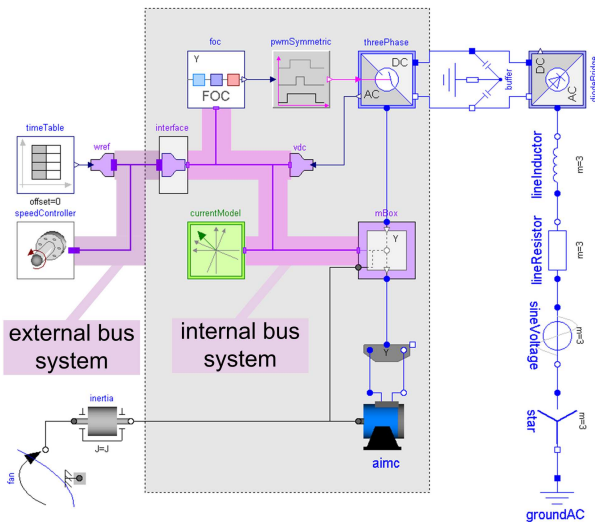


Figure 7: An SED simulation of a speed controlled asynchronous induction machine with an internal and an external bus system.

4 Examples

In order to point out the basic advantages of using standardized 'ready-to-use' models and models on different levels of abstraction three simulations get compared. These three simulations can be found in the *Examples.Tutorial* package of the SED library. All three simulations represent the same drive system with identical component parameters. The drive system simulated, is a speed controlled asynchronous induction machine driving an industrial fan. The machine is sourced via a six pulse diode bridge that converts three phase ac voltages to a rippled dc voltage in the dc-link circuit. A capacitive grounding component smoothens the ripple of the dc voltage and assures system stability when calculating results with *Dymola*. FOC is used to control the asynchronous induction machine. The fan modelled with a quadratic speed dependent torque and an inertia component is directly connected to the shaft of the machine. The parameters of the simulations are as follows:

- Three phase ac supply:
 - Supply Voltage $\hat{V}_{AC} = 660 \text{ V}$
 - Line Resistor $R_{line} = 1 \text{ m}\Omega$
 - Line Inductance $L_{line} = 0.1 \text{ mH}$
- dc-Link:
 - Buffer resistance $R_{buffer} = 0.01 \text{ m}\Omega$
 - Buffer capacitance $C_{buffer} = 0.01 \text{ F}$
- Asynchronous induction machine:
 - Number of pole pairs $p = 4$
 - Nominal frequency $f = 50 \text{ Hz}$
 - Nominal phase voltage $V_N = 400 \text{ V}$
 - Nominal phase current $I_N = 416 \text{ A}$
 - Rotor's moment of inertia $J_r = 35 \text{ kg m}^2$
 - Stator resistance $R_s = 8.086 \Omega$
 - Stator stray inductance $L_{s\sigma} = 300.1 \mu\text{H}$
 - Main field inductance $L_m = 8.231 \text{ mH}$
 - Rotor stray inductance $L_{r\sigma} = 502 \mu\text{H}$
 - Rotor resistance $R_r = 4.934 \Omega$
 - Stator phases star-connected
- Fan:
 - Nominal speed $\omega_N = 78 \frac{\text{rad}}{\text{s}}$
 - Nominal torque $\tau_N = 5227 \text{ Nm}$
 - Inertia $J = 50 \text{ kg m}^2$

4.1 Simulations with 'ready-to-use' models

In Figure 8 a quasi stationary simulation, designated as case A, is presented. The 'ready-to-use' model `aimcqs` contains a torque controlled asynchronous induction machine drive considering only quasi stationary effects. Also the dc/ac converter is included in this component. The reference torque is generated by a speed controller and fed to the FOC in the 'ready-to-use' model via the external bus system and an interface model `wRef`. The reference speed curve for the drive is defined by a time table. This time table contains a `RealOutput` connector, which doesn't match the `ControlBus` connectors of the drive model and the speed controller. That is why a bus adaptor must be used to connect the reference speed signal to the external bus system. This bus adaptor also provides the correct physical unit accordingly. In Figure 9 another transient simulation setup, designated as case B, is shown. The difference to case A is that in case B the electric transients are considered by using a 'ready-to-use' `TransientDrives` model of a torque controlled asynchronous induction machine, called `aimcfoc`.

Figure 10, a simulation result of case B, illustrates that the reference speed, w_{Ref} , matches the real shaft speed, $w_{Mechanical}$, in small speed regions whereas at the instant $t = 3$ s, when the speed is higher, the torque limit of the drive is reached and therefore the acceleration of the inertia is limited. The speed curves in case A are very similar to the ones displayed in Figure 10.

In Figure 11, Figure 12 and Figure 13 case A and case B are compared. From the beginning of the simulations until $t = 1.2$ s the stator phase current in case B, $i_{Machine, caseB}$, deviates significantly from the stator phase current in case A, $i_{Machine, caseA}$. This is because the flux controller in the `TransientDrives` model creates a very high flux by generating a reference current to magnetize the machine very fast. In case A this magnetizing effect is neglected, because transient electric effects are not considered in the quasi stationary model. At $t = 3.45$ s the current $i_{Machine, caseB}$ deviates from $i_{Machine, caseA}$ because the flux weakening function in the FOC triggers a demagnetization of the iron core in the machine model in case B whereas in case A this effect is not considered. Furthermore there are larger overshoots of current, torque and voltage in case B whenever the reference speed, w_{Ref} , triggers a significant torque change.

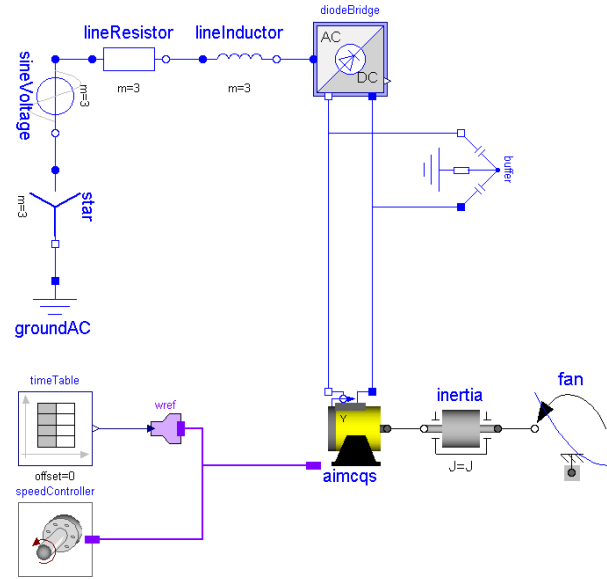


Figure 8: Case A; quasi stationary simulation of the speed controlled fan drive with a 'ready-to-use' SED model.

4.2 Simulating ideal switching effects

Figure 14 shows a simulation model considering switching effects, which is designated as case C. In case A and case B the dc/ac converter is modelled by considering the power balance between supply circuit and load circuit. However, in case C a converter modeled with ideal switches and controlled by a symmetric pulse width modulation (PWM) algorithm is used to simulate the effects of pulsed stator voltages in the drive. The basic differences between case C and the simulations using power balance converters are shown in Figure 15 and in Figure 16. Due to the converter switching the current and the electric torque of the machine have a significant ripple.

4.3 Performance comparison of the simulations

The three simulations are developed and simulated on a conventional 1 GHz, 512 MB PC with a 5400 rpm hard drive with Dymola 6.0a using the DASSL solver. Processing the quasi stationary simulation, case A, is the fastest since it contains the smallest set of differential equations and algebraic equations among the three investigated cases. After translation, case A has only 16 differentiated variables and 333 equations. Processing the presented results takes less than 1 s processing time. Case B shows a simulation considering also electrical transient effects in the machine. Consequently, the model of the control system applied in

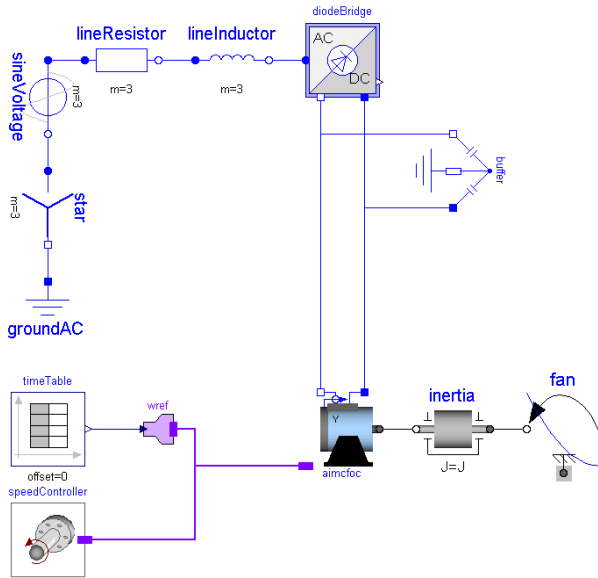


Figure 9: Case B; electrical transients simulation of the speed controlled fan drive with a 'ready-to-use' SED model.

case B is more complex. Case B contains 42 differentiated variables and 867 equations, which is the reason for a processing time increase of 50% for the presented results. However, processing case B takes still a much smaller effort than processing case C. Calculating the presented results of case C takes around 5 min 30 s. The reason for this tremendous computation time is that the solver iterates each event in order to find the precise switching instant. Since the converter in case C works with a 2 kHz PWM the processing effort is considerably. Case C contains 41 differentiated variables and 849 equations, which indicates a smaller system than case B. Still, case B can be solved much faster because the dc/ac converter in the 'ready-to-use' drive model is modeled by only considering the power balance between supply circuit and load circuit.

5 Conclusions

The scope, the structure and the most important packages of the SED library are described. Three simulation setups with similar parameter settings on different levels of abstraction are presented and the results are compared. The comparison shows that the 'ready-to-use' *QuasiStationaryDrives* models of the SED library have the best performance. However, they don't show all the physical effects that can be analysed with a simulation considering electric transient effects and converter switching effects. Typical applications for *QuasiStationaryDrives* models are energy

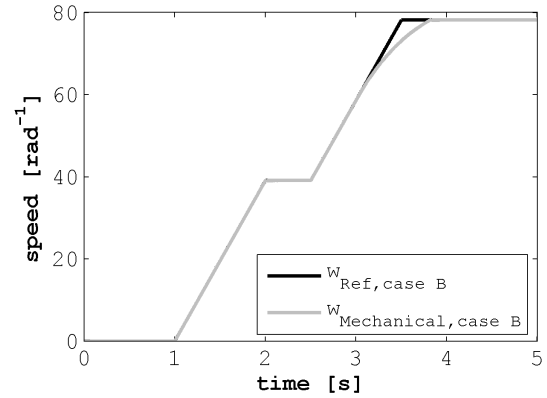


Figure 10: Reference speed and real fan speed in case B.

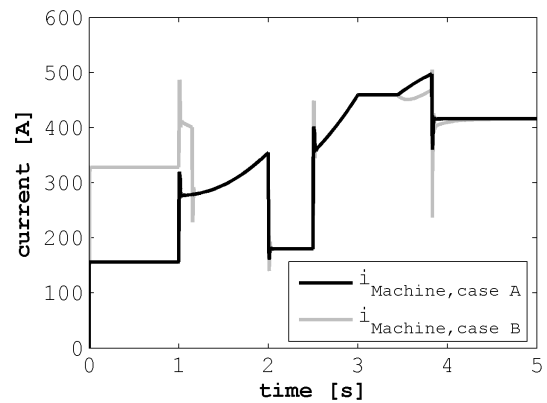


Figure 11: RMS values of the stator phase currents of the machine in case A and case B.

balance analyses of electromechanical systems, such as HEV or electric vehicle concepts. In such applications switching effects and electrical transient effects can be neglected. The models in the *Transient-Drives* package can be used when system responses on fast transient events have to be investigated or if controller optimization is the focus of investigation. Furthermore, it is shown that the application of power balance converter models helps saving a considerable amount of computer resources when simulating electric drives with *Modelica*.

References

- [1] D. Simic, H. Giuliani, C. Kral, and F. Pirker, "Simulation of conventional and hybrid vehicle including auxiliaries with respect to fuel consump-



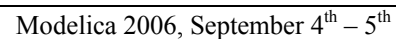
Modelica 2006, September 4th – 5th

Figure 14: Case C; electrical transients and switching effects simulation of the speed controlled fan drive with elementary SED models.

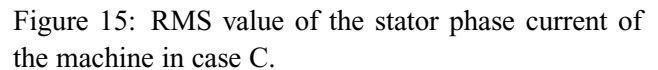


Figure 16: Electrical torque of the machine in case C.

Quasi-stationary AC Analysis Using Phasor Description With Modelica

Olaf Enge¹⁾, Christoph Clauß¹⁾, Peter Schneider¹⁾, Peter Schwarz¹⁾,
Matthias Vetter²⁾, Simon Schwunk²⁾

1) Fraunhofer Institute Integrated Circuits, Branch Lab Design Automation,
Zeunerstraße 38, 01069 Dresden, Germany
olaf.enge@eas.iis.fraunhofer.de

2) Fraunhofer Institute Solar Energy Systems, Heidenhofstraße 2, 79110 Freiburg, Germany

Abstract

The investigation of large technical systems by simulating long time periods requires very effective methods. If only sinusoidal quantities occur in the electrical domain, phasor analysis can be used to describe the steady-state behaviour of this part of the physical system. In this paper, modelling of AC circuits and electromechanical drives with electrical AC subsystems is presented using the well-known phasor method within Modelica. To this end, some fundamentals concerning phasor description are repeated before a possible implementation of AC circuits within Modelica is proposed. This implementation uses a new Modelica-library which is still under construction. The main content of this library is introduced. Furthermore, some statements are given concerning the library's usage when coupling with other domains or with transient submodels, or when switching between transient and phasor analysis, respectively. Finally, three examples are presented.

Keywords: sinusoidal quantity, steady-state analysis, phasor domain models, model coupling, variable model structure

1 Introduction

Many linear electrical circuits use alternating current (AC circuits). Most of them operate with a nominal frequency and nearly ideal sinusoidal electrical quantities. Distortions occurring due to circuits' nonlinearities can often be neglected. This way, we can define so called *idealized linear AC circuits*. They shall be characterized by ideal sinusoidal quantities and one single (nominal) frequency.

Three operating modes can be distinguished in idealized linear AC circuits: the steady-state mode, the dynamic mode, and the so-called *quasi-stationary* mode. The first mode is characterized by constant amplitudes and phases of all sinusoidal quantities. The

system yields the sinusoidal steady-state response. During the second mode, „fast“ dynamic changes of sinusoidal quantities occur. „Fast“ means that the appearing dynamic processes shall have a low dominant time constant compared to the nominal frequency. Usually, it is a good choice if this time constant is less than $10T$ ($T = 1/f$, f – nominal frequency). In this case, the full transient (or complete) response of a system has to be considered (using e.g. the electrical Modelica standard library [14] or Haumer's libraries [6]). Such „fast“ dynamic processes appear with switching operations or (stepwise or „fast“ continuous) changes of parameters. Because consisting only of decreasing shares in most cases, they fade away with advancing time and can be neglected after a finite time period. The third mode – the so-called *quasi-stationary* mode – shall be understood as a sequence of steady states on the following condition: parameters (which would be constant at steady-state analysis) may vary extraordinary slowly compared to the nominal frequency. It is signaled by „slow“ alterations of amplitudes and phases of the sinusoidal quantities. Usually, it is a good choice if the dynamic processes have a dominant time constant higher than $10T$.

All three operating modes can be investigated by implementing behavioural models (differential-algebraic equations) within appropriate numerical simulation systems. Because the instantaneous values of the sinusoidal quantities are changing perpetually, the performance of dynamic simulations depends on the nominal frequency and, hence, is limited. Especially, the study of such systems for a long time period (hours, days, years) is hardly possible.

In this paper, a very efficient method for modelling AC circuits and its implementation within Modelica is presented. This method can naturally be used for steady-state analysis of such systems. The method's principal idea is the substitution of the sinusoidal (time-depending) physical quantities in the transient model by constant complex quantities – so-called

phasors – in the steady-state model. Hence, the system's behaviour is not described by differential but by algebraic equations. The models are called phasor-domain models. This method of steady-state analysis was introduced by Steinmetz in the late 19th century ([10], [11]). Nowadays it is well-known and can be found in many elementary textbooks (e.g. [1], [3], [4], [5], [7], [8], [9]). Wiesmann also uses such an approach in parts of his power-systems library for Modelica ([12]).

Sometimes it is necessary to study physical systems containing both AC circuits and subsystems from other domains for long time periods. Then the quasi-stationary operating mode of AC circuits is of special interest. Considering this mode, a combination of phasor-domain and transient models is possible.

2 Phasor description of AC circuits

2.1 Fundamentals

A sinusoidal signal of the form

$$x(t) = \hat{x} \sin(\omega t + \varphi) \quad (1)$$

(\hat{x} – amplitude, ω – angular frequency, φ – phase) can be understood as a well-defined part of a time-dependent complex quantity

$$\begin{aligned} \underline{x}(t) &= \hat{x} e^{j(\omega t + \varphi)} \\ &= \hat{x} [\cos(\omega t + \varphi) + j \sin(\omega t + \varphi)] . \end{aligned} \quad (2)$$

In the complex plane, signal $\underline{x}(t)$ from (2) describes a rotating vector having a constant length \hat{x} and forming an angle of $(\omega t + \varphi)$ with the real axis at time instant t (see **Fig. 1**). The original signal $x(t)$ can then be gained at any time instant by projecting $\underline{x}(t)$ to the imaginary axis according to

$$\begin{aligned} x(t) &= \text{Im}[\underline{x}(t)] = \text{Im}[\hat{x} e^{j(\omega t + \varphi)}] \\ &= \hat{x} \sin(\omega t + \varphi) . \end{aligned} \quad (3)$$

If the angular frequency ω is constant then the angular velocity of a rotating vector is also constant and, therefore, is not of special interest. The complex quantity $\underline{x}(t)$ and, hence, the original signal $x(t)$ are adequately determined by two values: amplitude \hat{x} and phase φ . Using the rms value $X = \hat{x}/(\sqrt{2})$ of the signal instead of its amplitude, $x(t)$ can be represented by the following phasor

$$\underline{X} = X e^{j\varphi} . \quad (4)$$

This phasor has the constant length X and forms a constant angle φ with the real axis (see **Fig. 2**).

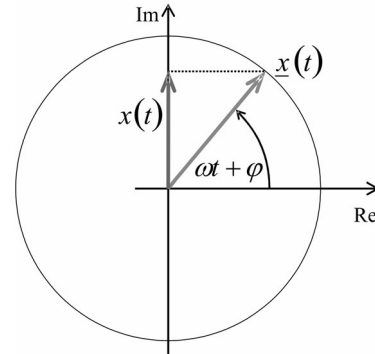


Figure 1: Rotating vector in complex plane

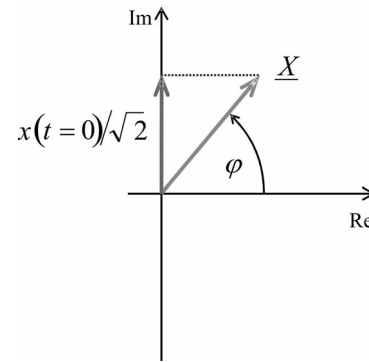


Figure 2: Phasor in complex plane

Please note, that

$$\frac{d}{dt} \{ \text{Im}[\underline{X} e^{j\omega t}] \} = \text{Im}[j\omega \underline{X} e^{j\omega t}] \quad (5)$$

which means that a derivation of $x(t)$ in time domain has to be replaced by multiplication with $j\omega$ in phasor domain.

2.2 Ohm's Law in generalized form

Ohm's Law is well-known in DC analysis. Using phasor description, this law can be written in an analogous form for AC analysis. Ohm's Law reads in its generalized (or complex) form

$$\underline{V} = \underline{Z} \underline{I} , \quad (6)$$

where \underline{V} and \underline{I} are the voltage and current phasors, respectively, and \underline{Z} denotes the impedance or „complex resistance“. It holds for linear electric components like resistors, inductors, and capacitors. Consideration of the relations between current and voltage at each type of component yields the corresponding impedances. Assuming a sinusoidal current $i(t) = \hat{i} \sin(\omega t + \varphi_i)$ and a sinusoidal voltage

$v(t) = \hat{v} \sin(\omega t + \varphi_v)$ and replacing them by corresponding phasors \underline{V} and \underline{I} , the impedances of each type of component can finally be represented with (5) as:

$$\underline{Z}_R = R, \quad (7)$$

$$\underline{Z}_L = j\omega L, \quad (8)$$

$$\underline{Z}_C = \frac{1}{j\omega C} = -j\frac{1}{\omega C} \quad (9)$$

(indices R , L , and C stand for resistor, inductor, and capacitor, respectively).

In AC circuits, inductors coupled by magnetic fields often occur. The behaviour of such a coupling can easily be included into the phasor description. Let M denote the mutual inductance between two coils. Then the EMF in coil 1 caused by a current through coil 2 reads

$$e_1(t) = \mp M \frac{di_2(t)}{dt} \quad (10)$$

(the upper sign holds if the coils are oriented likewise). This yields the following voltage drop

$$v_1(t) = \pm M \frac{di_2(t)}{dt}. \quad (11)$$

With (5), the relations between the current phasor in one branch and the voltage phasor in the other branch read finally

$$\begin{aligned} \underline{V}_1 &= \pm j\omega M \underline{I}_2, \\ \underline{V}_2 &= \pm j\omega M \underline{I}_1. \end{aligned} \quad (12)$$

2.3 Kirchhoff's Laws

According to Kirchhoff's Current Law, the sum of the instantaneous values of all currents in a node must vanish at each point in time

$$\sum_k i_k(t) = 0, \quad (13)$$

where k represents each branch being incident with the considered node. In case of sinusoidal quantities, each current $i_k(t) = \hat{i}_k \sin(\omega t + \varphi_{ik})$ can be determined by projecting a rotating vector

$$\underline{i}_k = \hat{i}_k e^{j(\omega t + \varphi_{ik})} = \sqrt{2} \underline{I}_k e^{j\omega t} \quad (14)$$

to the imaginary axis. That's why it follows from (13)

$$\sum_k \text{Im}[\sqrt{2} \underline{I}_k e^{j\omega t}] = 0 \quad (15)$$

for each time instant t which means finally

$$\sum_k \text{Im}[\underline{I}_k] = \text{Im}\left[\sum_k \underline{I}_k\right] = 0. \quad (16)$$

Displacing the time axis by $\pi/2$, the currents read $i_k(t) = \hat{i}_k \cos(\omega t + \varphi_{ik})$. Those can be represented by projecting the rotating vector \underline{i}_k to the real axis. Therefore, it holds

$$\text{Re}\left[\sum_k \underline{I}_k\right] = 0. \quad (17)$$

Equations (16) and (17) yield finally the generalized form of Kirchhoff's Current Law

$$\text{Re}\left[\sum_k \underline{I}_k\right] + j\text{Im}\left[\sum_k \underline{I}_k\right] = \sum_k \underline{I}_k = 0. \quad (18)$$

Kirchhoff's Voltage Law says that the sum of the instantaneous values of all voltage drops in one mesh must vanish at each point in time

$$\sum_k v_k(t) = 0. \quad (19)$$

The generalized form of this law can be derived in the very same way as shown above. It reads

$$\sum_k \underline{V}_k = 0. \quad (20)$$

2.4 Coupling phasor domain models and transient models – electric machines

Sometimes, investigations of models consisting of a fast part and a comparatively slow part shall be carried out. If only sinusoidal quantities occur in the fast part of such a model then it can be of interest to use a phasor domain description for this submodel. In these cases, it is necessary to couple at least two submodels: one submodel in phasor-domain description and one transient submodel.

For convenience in the following, phasor-domain models are referred to as *P-models* while transient models are called *T-models*. To couple a P- and a T-submodel within one mathematical description (see **Fig. 3**), some assumptions must be fulfilled:

- the connection between the two submodels consists of one-directional signals only (signals computed within the P-submodel and needed to be known in the T-submodel are referred to as I-signals – input to the phasor domain; signals resulting from the T-submodel and influencing the P-submodel are referred to as O-signals –

- output from phasor domain),
- all I-signals are only allowed to alter very slowly compared with the P-submodel's nominal frequency.

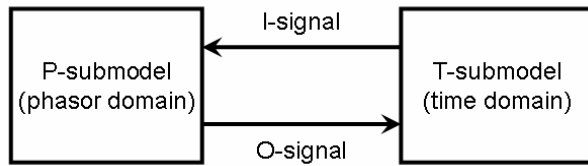


Figure 3: Coupling a P- and a T-submodel

The second point in the list requires either some a priori knowledge about the T-submodel or a permanent check of the first time derivatives of all I-signals. The consideration of all assumptions ensures that the P-submodel is always in steady-state or quasi-stationary mode.

Electric machines consist of a mechanical and an electrical subsystem. The complete response of the mechanical part is very often slow compared to that of the electrical part. This is particularly true with AC machines because of the nominal frequencies mostly used. Under certain assumptions (perfect symmetry, no saturation etc.), the steady state of an AC machine is characterized by ideal sinusoidal voltages and currents with constant amplitudes and phases. The steady state is furthermore signalized by a constant angular velocity of the rotor – with an induction machine – or by a constant torque angle – with a synchronous machine – as well as a constant torque produced electrically with both machines (see e.g. [13]). If the mechanical angular velocity or the torque angle alter very slowly compared to the electrical nominal frequency then the electrical subsystem is in quasi-stationary mode. This results in a slow variance of amplitude and phase of the electric sinusoidal quantities. To study electric machines in steady-state or quasi-stationary mode of the electrical subsystem, a combination of two submodels – a phasor-domain model of the electrical subsystem and a transient model of the mechanical subsystem – is possible.

2.5 Switching between phasor-domain and time-domain models

An electrical model which is to be switched between time domain and phasor domain may be part of a heterogeneous system containing also subsystems of other domains. Therefore, this model is referred to as AC submodel in the following.

If the transient response of an AC subsystem has been

faded away then a changeover from time domain to phasor domain is possible because the system is in a quasi-stationary mode or in a steady state. Both states are characterized by constant or at least by nearly constant amplitudes and phases of voltages and currents. During a numeric simulation, such a changeover causes a transition of the formerly differential-algebraic equation system (DAE system describing the AC submodel in time domain) into a linear system of algebraic equations (AE system describing the AC submodel in phasor domain)

$$Ax = b. \quad (21)$$

In (21), matrix A may depend on time t or on state variables of other domains (see e.g. [2]) and, hence, is known at switching time. The elements of vector b either are known (if depending on time or on state variables of other domains) or have to be determined during the transient simulation by “scanning” amplitude and phase continuously. This way, a consistent changeover can be carried out.

A changeover from phasor domain to time domain is always possible. It is actually necessary, if the transient response of an AC subsystem is of interest for a time interval (e.g. caused by a step-wise change of a parameter). In this case, the quasi-stationary or steady state is finished at switching time. During a numeric simulation, the AE system has to be replaced by the corresponding DAE system. Amplitudes and phases of all “transient” source components can be determined from the corresponding phasors. Additionally, initial values for all state variables of the DAE system are needed. These values can be calculated from the voltage phasors across capacitors and from the current phasors through inductors.

3 Implementation in Modelica

3.1 Basic partial models

A new library called *Complex* has been created for the implementation of phasor-domain models in Modelica. This library is designed such that it can be used like the Modelica standard library `Modelica.Electrical.Analog`. An important difference is the definition of a so-called *complex pin* instead of the standard pin to be used for connecting components. Without any annotations, the definition of the complex pin reads

```
connector ComplexPin
  Real vRe, vIm;
  flow Real iRe, iIm;
end ComplexPin;
```


containing real and imaginary part (v_{Re} , v_{Im}) of a voltage phasor \underline{V} as well as real and imaginary part (i_{Re} , i_{Im}) of a current phasor \underline{I} . Using this connector, important partial models like `OnePort` were derived which, first, realize Kirchhoff's relations in generalized form between the component's pins and, second, compute real and reactive (idle) power out of internal quantities and make them available via output signals. The main part of the code for `OnePort` reads

```
partial model OnePort
  PosComplexPin p;
  NegComplexPin n;
  Real vRe, vIm, iRe, iIm;
  Real phi_v, phi_i, phi;
  Real activePwr, idlePwr;
  Complex.Interfaces.ComplexOutput
    complexPwr;
equation
  vRe = p.vRe - n.vRe;
  vIm = p.vIm - n.vIm;
  iRe = p.iRe;
  iIm = p.iIm;
  0 = p.iRe + n.iRe;
  0 = p.iIm + n.iIm;
  phi_v = Complex.Math.atan2(vIm, vRe);
  phi_i = Complex.Math.atan2(iIm, iRe);
  phi = phi_v - phi_i;
  activePwr = v*i*cos(phi);
  idlePwr = v*i*sin(phi);
  complexPwr.real = activePwr;
  complexPwr.im = idlePwr;
end OnePort;
```

Based on `OnePort` and other partial models, source components (different voltage sources, some current generators) as well as many linear RLC components for single-phase grids have been created.

Many AC circuits are three-phase systems. To simplify modelling of such systems using the library `Complex`, the *complex plug* – a special “multi-phase complex pin” – was implemented:

```
connector ComplexPlug
  parameter Integer m(final min=1) = 3;
  Complex.SinglePhase.Interfaces.ComplexPin
    complexpin[m];
end ComplexPlug;
```

Using this connector, some partial models were derived which are suitable to be extended to source components or RLC components of symmetric multi-phase systems. An example of such a partial model is `TwoPlug`:

```
partial model TwoPlug
  PosComplexPlug plug_p(final m=m);
  NegComplexPlug plug_n(final m=m);
  Real vRe[m], vIm[m], iRe[m], iIm[m];
  Real phi_v[m], phi_i[m], phi[m];
```

```
Real activePwr[m], idlePwr[m];
Complex.Interfaces.ComplexOutput
  complexPwr[m];
equation
  vRe = plug_p.complexpin.vRe -
    plug_n.complexpin.vRe;
  vIm = plug_p.complexpin.vIm -
    plug_n.complexpin.vIm;
  iRe = plug_p.complexpin.iRe;
  iIm = plug_p.complexpin.iIm;
  for j in 1:m loop
    phi_v[j] =
      Complex.Math.atan2(vIm[j], vRe[j]);
    phi_i[j] =
      Complex.Math.atan2(iIm[j], iRe[j]);
    activePwr[j] = v[j]*i[j]*cos(phi[j]);
    idlePwr[j] = v[j]*i[j]*sin(phi[j]);
  end for;
  phi = phi_v - phi_i;
  complexPwr.real = activePwr;
  complexPwr.im = idlePwr;
end TwoPlug;
```

This model mainly realizes Kirchhoff's Voltage Law between the two plugs and computes real and reactive power for all three phases.

3.2 Basic one-phase components

The basic one-phase RLC components are assorted in the package `Complex.SinglePhase.Basics`. The inductor's model (see **Fig. 4**) reads e.g. without any annotations and comments:

```
model Inductor
  extends
    Complex.SinglePhase.Interfaces.OnePort;
  parameter Real L=1;
equation
  vRe = -omega*L*iIm;
  vIm = omega*L*iRe;
end Inductor;
```

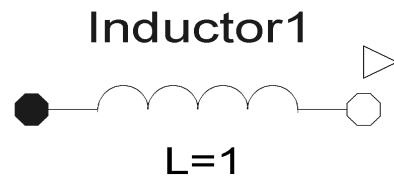


Figure 4: Icon for complex inductor

If the inductance is not constant but depends on time or some other physical quantity like a mechanical coordinate then the model `VariableInductor` has to be used. In this model, the inductance is governed via an input signal.

```
model VariableInductor
  extends
    Complex.SinglePhase.Interfaces.OnePort;
```

```

Modelica.Blocks.Interfaces.RealInput L;
equation
  vRe = -omega*L*iIm;
  vIm = omega*L*iRe;
end VariableInductor;

```

A complete list of the Basics-package reads:

- Ground
- Resistor
- Inductor
- Capacitor
- Conductor
- Transformer
- IdealTransformer
- VariableResistor
- VariableInductor
- VariableCapacitor
- VariableConductor
- VariableTransformer

Some one-phase source components can be found in the package `Complex.SinglePhase.Sources`. Presently, voltage sources and current generators are implemented with

- constant amplitude and phase,
- amplitude and phase governed by input signal.

Additionally, power sources with given power (constant or governed by input signal) are available. Furthermore, the package `Complex.SinglePhase` contains some sensors for “measuring” the phasors (rms value, phase) of voltage or current. Finally, the power quality sensor (`PQ_sensor`) can be used to determine all interesting values concerning the power and its quality.

3.3 Basic multi-phase components

The basic multi-phase RLC components are assorted in the package `Complex.MultiPhase.Basics`. These definitions can be used in symmetric multi-phase grids of arbitrary number of phases. The package contains models for resistors, inductors, capacitors, and conductors each with constant constitutive parameters or with parameters governed by input signals. For comparability, the inductor’s model with constant inductance is given here:

```

model Inductor
  extends
    Complex.MultiPhase.Interfaces.TwoPlug;
  parameter Real L[m]=fill(1,m);
  Complex.SinglePhase.Basics.Inductor
    inductor[m] (final L=L);
equation
  connect(inductor.p, plug_p.complexpin);
  connect(inductor.n, plug_n.complexpin);
end Inductor;

```

Moreover, elements for realizing star connections or delta connections as well as for connecting multi-phase and single-phase components together are included in the package. Finally, some sources and sensors are available, too.

3.4 Electric induction machine

In the presented library, a model of an electric induction machine has been implemented, too. To this end, the package `Complex.Machines` was created. An induction machine can be regarded as consisting of a mechanical and an electrical subsystem. Specific interactions take effect between the two subsystems. The model of the induction machine combines a transient (time-domain) part of the mechanical subsystem with a phasor-domain part of the electrical subsystem. The model is only valid for analysing the machine within steady-state or quasi-stationary mode of the electrical subsystem. **Fig. 5** shows the well-known steady-state equivalent circuit for one phase of an induction machine. In this diagram, \underline{V}_N is the voltage

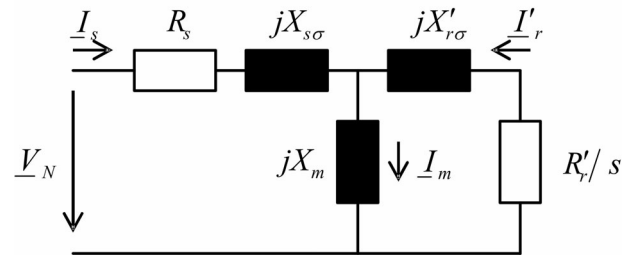


Figure 5: Induction machine’s equivalent circuit

phasor of the net, \underline{I} denotes a current phasor (subindex s stands for the stator, subindex r stands for the rotor, m denotes the main reactance, σ refers to the stray reactances). R and $X = \omega L$ are ohmic resistance and inductive reactance, respectively. The slip between electrical angular frequency ω and rotor’s angular velocity ω_m is denoted by s , while $s = (\omega - \omega_m)/\omega$. (Please note that this model is a time phasor description not using any space phasors.) Furthermore, it shall hold $X_s = X_m + X_{s\sigma}$ and $X_r = X_m + X_{r\sigma}$. Finally, the stray coefficient can be determined from the reactances according to $\sigma = 1 - \sqrt{X_m^2/(X_s X_r)}$. Using some simplifying assumptions (e.g. R_s sufficient small) and denoting the number of phases by n , the torque produced electrically reads (see e.g. [13])

$$T = n \frac{V_N^2}{\omega} \frac{(X_h/X_s)^2}{(R_r'/s) + (s/R_r')(\sigma X_r')^2}. \quad (22)$$

Slip s and torque T are the signals which connect the mechanical and the electrical subsystem. Considering the definitions of coupling signals in chapter 2.4 (see **Fig. 3**), the slip is the I-signal and the torque is the O-signal. Hence, the slip must not vary too fast, because one must ensure at each point in time that the electrical subsystem works within the quasi-stationary mode. (An automatic monitoring of this demand is not implemented in the library at the moment but would be a nice feature.) Under this assumption, the torque is always calculated correctly.

4 Examples

4.1 Electric circuit with varying resistance

Fig. 6 shows a simple example circuit containing a VariableResistor-component. The voltage source V works with constant amplitude and zero phase. The resistance $R3$ is a ramp function of time ($10^{-6} \dots 5\Omega$). **Fig. 7** shows the rms values of the voltage drops across resistor $R2$ and the inductor as well as the source voltage. The voltages across $R2$ and L

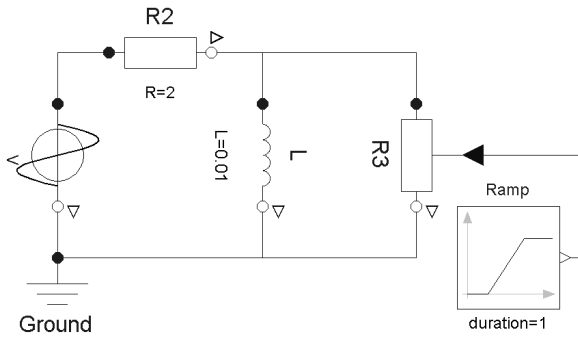


Figure 6: Example circuit

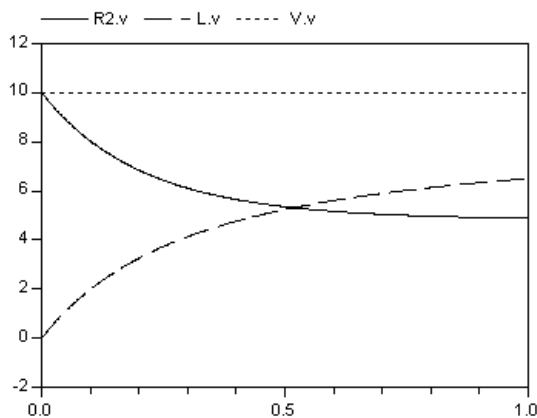


Figure 7: Voltages drops across $R2$ and L , source voltage

depends on time. But this curve is not a transient response. It is rather a sequence of quasi-stationary states spread over time axis. To prove this statement, one shall have a look at the time constant of the curves. This constant is about 0.5 s which is more than $10T$ because a nominal frequency of 50 Hz is used. The real and imaginary part of the voltage drop across the inductor is plotted in **Fig. 8**. This diagram shows the variation of phase within the sequence of quasi-stationary states.

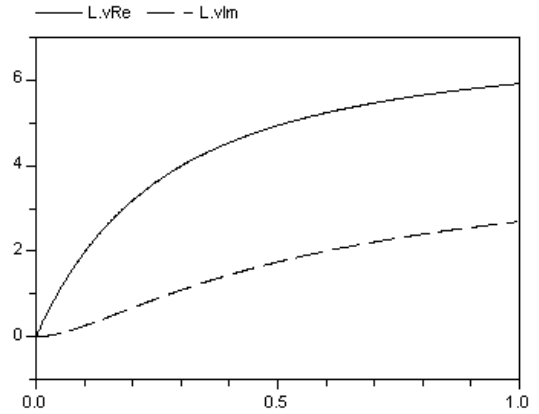


Figure 8: Real and imaginary part of voltage across L

4.2 Induction machine in quasi-stationary mode

This example deals with a three-phase induction machine. The test setup is shown in the schematic diagram of **Fig. 9**. The machine's electrical subsystem

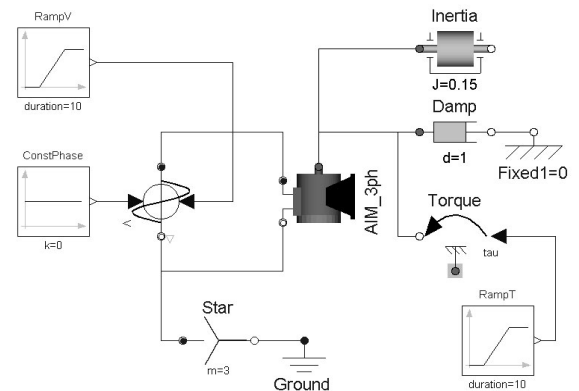


Figure 9: Schematic diagram

is connected to a “three-phase voltage source” which is the same as being connected to three single source components. The three-phase source works with variable amplitude and phase each governed by one input signal. During the simulation, the amplitude increases along a ramp function (see **Fig. 10**, solid line) while the phase remains at zero. On the mechanical side, the shaft of the induction machine is connected to an ad-

ditional inertia, a damper component, and an applied torque. Inertia and damping constant have fixed values. The torque is governed by a ramp function which starts at a simulation time of 20 s (see **Fig. 10**, dashed line).

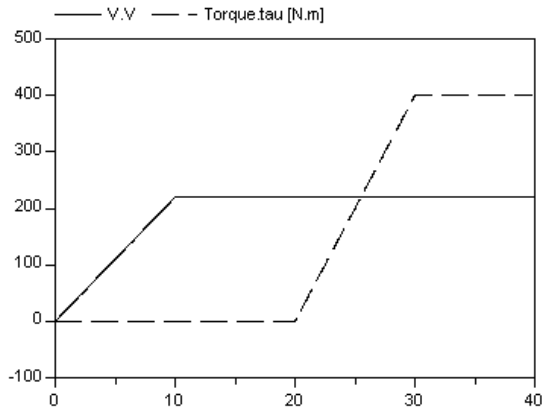


Figure 10: Inputs: source voltage and applied torque

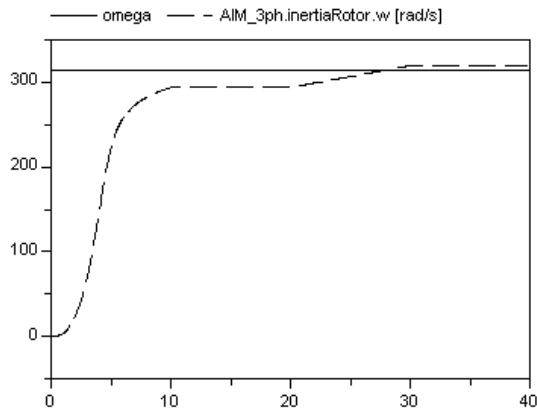


Figure 11: Electrical and mechanical frequencies

In the first time interval (between 0 and 20 s), the induction machine works as a motor. The increasing stator voltage causes an increasing acceleration of the shaft. Because of the mechanical damping, the angular velocity – i.e. the shaft velocity – (see **Fig. 11**, dashed line) finally reaches a value which is less smaller than the electrical angular frequency (see **Fig. 11**, solid line). Hence, the slip has only positive values in the first time interval. **Fig. 11** additionally shows that the simulation produces a transient response of the shaft's angular velocity. The time constant during the highest acceleration is about 0.2 s which equals $10T$. Therefore, the electrical subsystem is still in the quasi-stationary mode.

In the second time interval (between 20 and 40 s), the shaft is accelerated by an increasing torque which is additionally applied to it. At one moment in time, the shaft's angular velocity becomes higher than the electrical angular frequency. Beginning with this time

instant (approximately 28 s), the induction machine is working in generator mode. Hence, the slip goes below zero. The transition of the working modes is characterized by a change of the sign of the difference between both frequencies or – which is the same – by a change of slip's sign. The transient process in the second time intervall is much slower than in the first interval. Therefore, the electrical subsystem is in the steady state or in the quasi-stationary mode during the complete simulation.

Finally, **Fig. 12** shows the electrical active power of one phase of the voltage source component and the mechanical power of the complete induction machine.

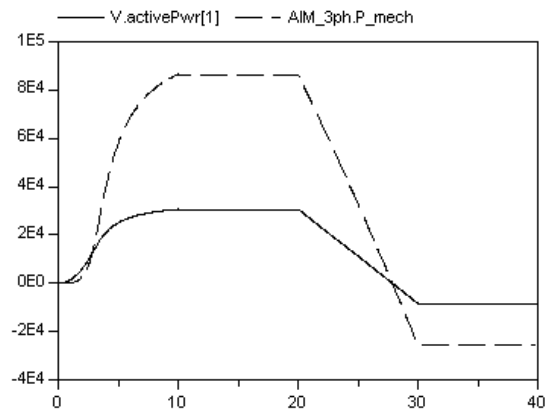


Figure 12: Electrical active power

Both curves have analogous shapes. After the transient response is faded away, the mechanical power is three times higher than the electrical power (because a three-phase machine is under consideration). In the first part of the simulation, the power curves have positive values which means that the source component produces energy while the motor consumes it. After a simulation time of about 28 s, the curves go below zero. This fact indicates that now the induction machine produces energy which is fed back into the electric net via the source component.

4.3 Example for domain switching

The switching between time domain and phasor domain and vice versa will be explained on a simple circuit containing a sinusoidal voltage source, an ohmic resistor, and a capacitor. **Fig. 13** shows the time-domain model (on the left hand side) and the phasor-domain model (on the right hand side). The left circuit remains in time domain during the complete simulation. The right circuit is modelled in time domain at the beginning. At an arbitrary point in time ($t_1 = 0.177$), the modelling domain of the right subsystem is changed to phasor domain (T-P

changeover). Now, the phasor-domain description is used until $t_2 = 0.377$ (again chosen arbitrarily). At this time instant, the modelling domain is changed back to time domain (P-T changeover). From t_2 to the end of simulation, the time-domain model is used. The component's parameters are identical in both circuits ($R = 10\Omega$, $C = 1\text{mF}$). Both sources are feeding a voltage of 1 V at $t = 0$ increasing at a rate of 0.5V/s . Because of the low increasing rate, the circuits are always in a quasi-stationary mode.

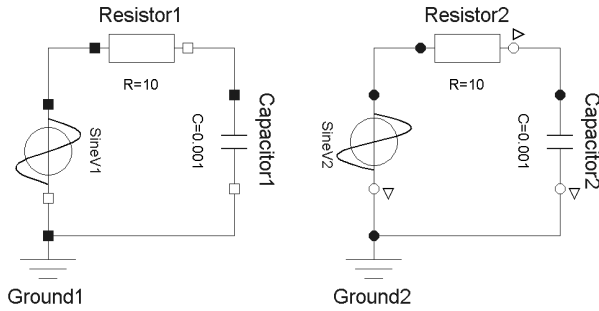


Figure 13: Schematic of domain-switching example; left: always within time domain, right: switching from time domain to phasor domain (T-P) and reverse (P-T)

Some simulation results are shown in the following figures. The first three diagrams contain the voltage drop V_{cref} across the capacitor of the permanent time-domain system and, from the switching system, the sinusoidal voltage drop (first and third interval) or the amplitude of the voltage phasor (second interval), respectively (both denoted by V_c). **Fig. 14** shows the complete simulation progress, while **Fig. 15** and

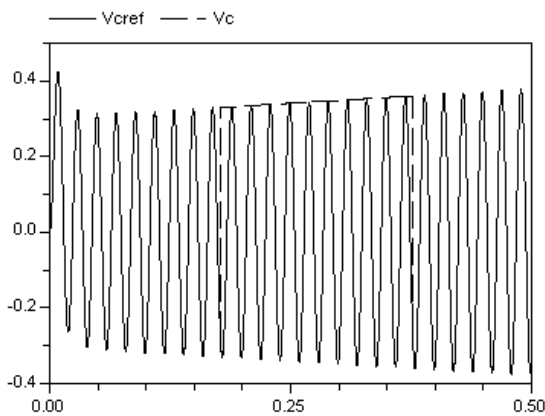


Figure 14: Voltages; V_{cref} : always in time domain, V_c : switching T-P and P-T (showing amplitude in P-domain)

Fig. 16 represent the details around the switching instants. Using the time-domain model for the switching circuit, both curves (V_{cref} and V_c) are identical. If this circuit is described within the phasor-domain then the dashed curve is nearly constant showing only the

amplitude of V_c .

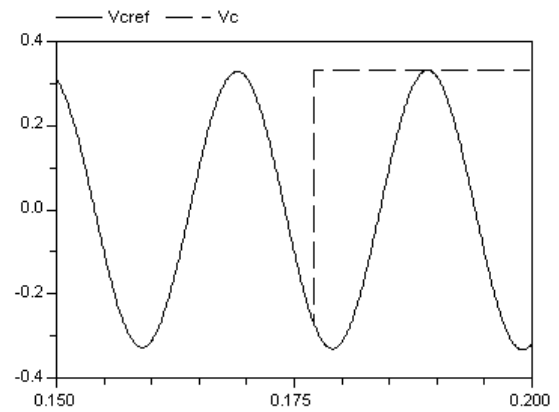


Figure 15: Zoomed voltages; T-P changeover

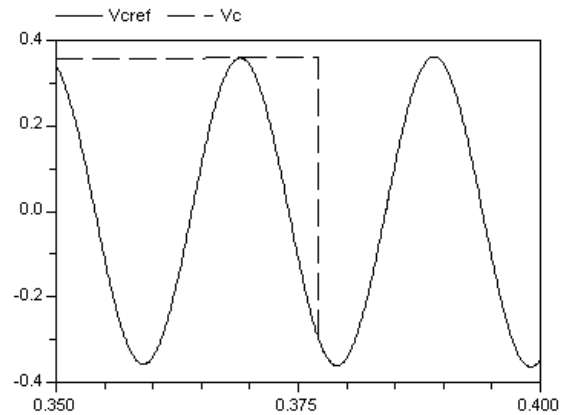


Figure 16: Zoomed voltages; P-T changeover

The last three diagrams show the current flowing through the permanent time-domain circuit (I_{ref}) and, from the switching system, the sinusoidal current (first and third interval) or the amplitude of the current phasor (second interval), respectively (both denoted by I). **Fig. 17** shows the complete simulation progress, while **Fig. 18** and **Fig. 19** represent the details around

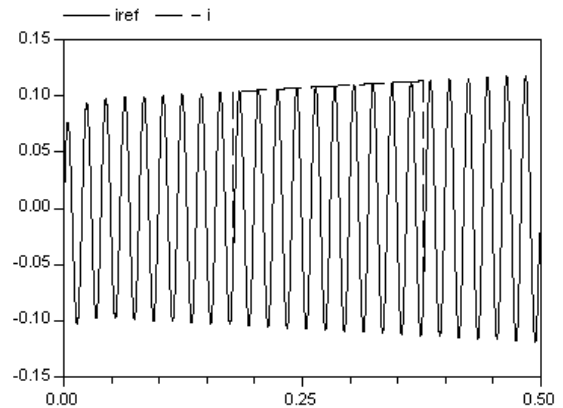


Figure 17: Currents; I_{ref} : always in time domain, I : switching T-P and P-T (showing amplitude in P-domain)

the switching instants. In these diagrams, both currents are identical, too, if the time-domain model is applied to the switching circuit. If the phasor-domain model is valid then the dashed curve is nearly constant showing only the amplitude of I .

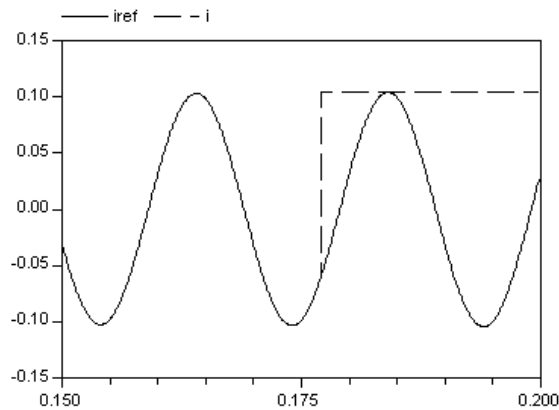


Figure 18: Zoomed currents; T-P changeover

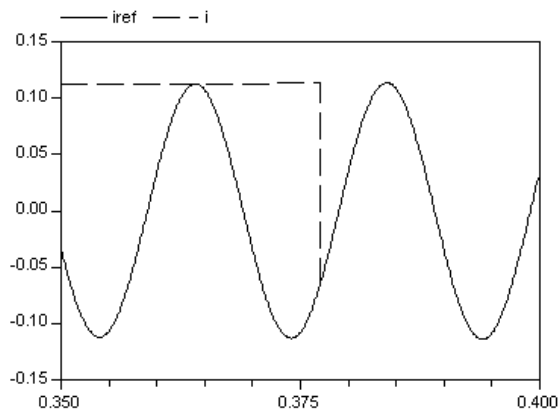


Figure 19: Zoomed currents; P-T changeover

5 Summary

This paper deals with the quasi-stationary AC analysis using phasor-domain models within Modelica. First, the term “quasi-stationary mode” is declared. Hereafter, some fundamentals concerning phasor description are repeated. Using such phasors, a possible implementation of AC circuits within Modelica is proposed. The main content of a corresponding new Modelica-library is introduced. This library can be used for modelling AC circuits in a very efficient way. Furthermore, some investigations of coupling time-domain models and phasor-domain models are presented in the paper. Hence, the introduced Modelica-library can also be used for modelling electromechanical drives with electrical AC subsystems under some assumptions. This fact is exemplified by modelling an induction machine.

Studying AC systems for long time periods within the time domain is hardly possible. Therefore, it may be of interest to switch between a time-domain model and a phasor-domain model and vice versa in an appropriate manner. This scenario is shortly concerned in the paper.

Finally, simulation results of three examples are given proving the principal capabilities of the library.

References

- [1] Desoer CA, Kuh ES. *Basic Circuit Theory*. McGraw-Hill, 1966.
- [2] Enge O. *Analyse und Synthese elektromechanischer Systeme*. Shaker, 2005.
- [3] Irwin JD. *Basic Engineering Circuit Analysis*. John Wiley & Sons, 2005.
- [4] Johnson DE. *Basic Electric Circuit Analysis*. John Wiley & Sons, 1995.
- [5] Johnson DE, Johnson JR, Hilburn JL, Scott PD. *Electric Circuit Analysis*. John Wiley & Sons, 1997.
- [6] Kral C, Haumer A. Modelica libraries for dc machines, three phase and polyphase machines. In Schmitz G (Ed.): *4th Int. Modelica Conference*, Hamburg, Germany, March 7-8, 2005, Proc., pages 549–558, The Modelica Association, 2005.
- [7] Paul CR. *Fundamentals of Electric Circuit Analysis*. John Wiley & Sons, 2000.
- [8] Phillipow E. *Grundlagen der Elektrotechnik*. Verlag Technik, 1992.
- [9] Smith RJ, Dorf RC. *Circuits, Devices and Systems – A First Course in Electrical Engineering*. John Wiley & Sons, 1992.
- [10] Steinmetz CP. Die Anwendung komplexer Größen in der Elektrotechnik. *Elektrotechnische Zeitschrift*, 4(1893)42: 597–599, 44: 631–635, 45: 641–643, 46: 653–654.
- [11] Steinmetz CP. *Theory and Calculations of Alternating Current Phenomena*. 1e, W.J. Johnston Comp., 1897 or 4e, McGraw, 1908.
- [12] Wiesmann H. Personal notifications to The Modelica Design Group.
- [13] Woodson HH, Melcher JR. *Electromechanical Dynamics – Part I: Discrete Systems*. John Wiley & Sons, 1968.
- [14] <http://www.modelica.org>

Identification and Controls of Electrically Excited Synchronous Machines

Hansjörg Kapeller Anton Haumer Christian Kral Franz Pirker Gert Pascoli
Arsenal Research
Giefinggasse 2, 1210 Vienna, Austria

phone +43-50550-6606, fax +43-50550-6595, e-mail: hansjoerg.kapeller@arsenal.ac.at

Abstract

Three-phase generators for traction applications are exposed to extreme mechanic and electric stress. Yet, the operating life has to be ensured for a period of 30 years. Therefore, technical optimization as well as focusing on today's and future aspects of environmental protection is required. Based on that, the development processes focuses on the minimization of losses over the entire operational range (usual speed range 600 and 2100 rev/min) and the reduction of masses. Both results in a considerably energy saving of the drive train operation. In many cases electrically excited synchronous machines feed diode rectifiers. The effects of this operation mode, particularly with regard to losses due to the harmonics, should be simulated just in the beginning of the design process for optimization.

Keywords: traction applications; three-phase generators; electrically excited synchronous machines; diode rectifiers

1 Introduction

The main objective of the paper is the development of an electromagnetic simulation tool in order to optimize operational characteristics. The particular focus of the investigation is the estimation of the losses caused by harmonics due to the rectifier. The innovative simulation tool is based on the object oriented language Modelica. For performing the Modelica simulations the simulation tool Dymola is used. The electromagnetic behavior of the traction generator can be modeled for typical operation modes using the simulation tool Dymola and the comprehensive Modelica Standard Library.

2 Model of the Controlled Synchronous Machine

For the Modelica Standard Library a model of an electrically excited synchronous machine with constant reactances is developed using space phasor theory ([1], [2]). In the presented paper the standard model of the electrically excited synchronous machine is used. Yet, for a more detailed investigation, the standard model can be extended such way that magnetic saturation in direct axis (d-axis) is taken into account [3].

The main synchronous generator is excited by an auxiliary synchronous generator (with rotating three-phase winding and stationary excitation) via a rotating rectifier bridge. The model of the integrated excitation equipment consists of a simplified model of a synchronous machine and a three phase diode rectifier bridge. A PID controller provides excitation for the auxiliary synchronous excitation machine.

2.1 Controller of the Excitation Equipment and Design Rules

In order to achieve appropriate terminal voltages of the traction generator under different operating conditions, it is necessary to implement an excitation voltage controller for the auxiliary excitation generator, which in turn determines the excitation voltage and current of the main generator, respectively.

The choice and the design of the controller (PI or PID controller) has to be performed after identifying the controlled system. The controller design significantly depends on this identification.

Identification of the Controlled System If a step-function is applied to the input of the system (i.e. excitation voltage of the auxiliary excitation generator), the response (i.e. the terminal voltage of the main

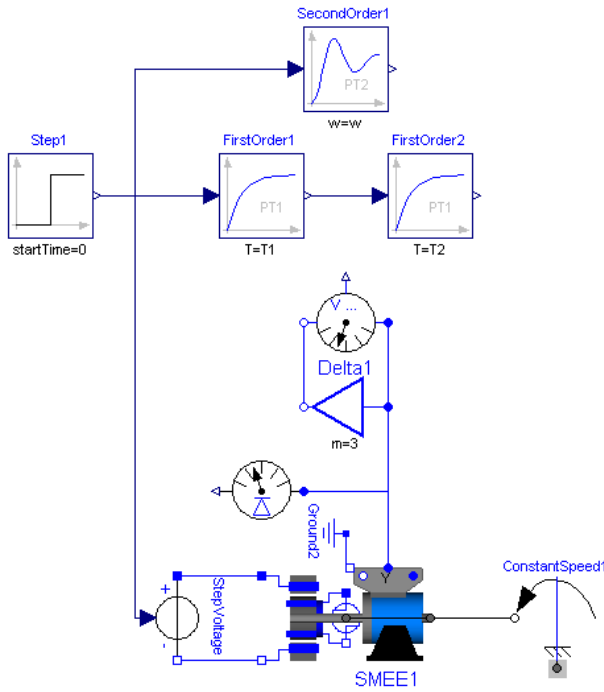


Figure 1: Scheme of the controlled system and approximation with delay elements. The identification is done with the simulation tool Dymola.

generator) can be observed. The controlled system is shown in Fig. 1 and consists of the (main) synchronous machine and the auxiliary excitation generator. From Fig. 2 it is obvious, that the analysis of the controlled system shows a non periodic second order system. The terminal voltage of the excitation synchronous generator arises with a first order delay (PT1) and the terminal voltage of the main synchronous machine arises with a first order delay, too. The system therefore appears as non periodic second order system (PT2 element). The structure of the second order controlled system is:

$$G(s) = \frac{k_S}{\left(\frac{s}{\omega}\right)^2 + 2D_S\left(\frac{s}{\omega}\right) + 1} = \frac{k_S}{(T_S s)^2 + 2D_S T_S s + 1} = \frac{k_S}{(1 + T_1 s)(1 + T_2 s)} \quad (1)$$

k_S	gain of the controlled system
ω	oscillation frequency of the undamped assumed system
T_S	reciprocal value of the oscillation frequency of the controlled system
D_S	attenuation ratio ($D_S \geq 1$) aperiodic system with real singularities $s = -T_1^{-1}$ resp. $s = -T_2^{-1}$
T_1, T_2	time constants T_1 and T_2

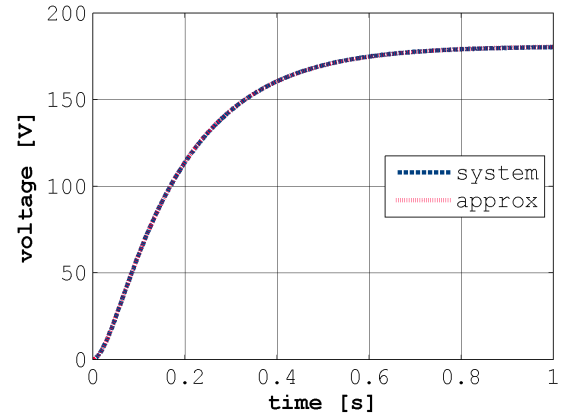


Figure 2: Step response of the real system (dashed line) and approximation with PT2-element (dotted line); criterion of least mean square is applied for optimization.

At time zero a step-function is applied to the input of the system and the response can be compared with the output of the two series connected PT1 elements, which are equivalent to a PT2 element. The two PT1 elements have the time constant characteristics T_1 and T_2 , respectively, and the resultant gain k_S . The series connected PT1 elements will be fine tuned such way that step responses of main controlled system and the step response of the series connected PT1 elements coincide. This can be done by adjusting the time constants T_1 and T_2 and the resultant gain k_S with respect to a criterion, e.g. the least mean square criterion. The comparison of both, the step response of the controlled system and the step response of the approximating (equivalent) PT2 element, is shown in Fig. 2. The use of the least square mean criterion results in a very close approximation of the system response. From the identified time constant characteristics T_1 and T_2 and the resultant gain k_S , the transfer function of the equivalent PT2-element can be estimated according to (1). The identification of the controlled system is performed semiautomatically. In the simulation tool Dymola a first coarse setting of the time constant T_1 and T_2 and the resultant gain k_S can be set to get a first approximation of the response of the controlled system. In a postprocessing application the least mean square criterion will be performed to get correctly identified values.

Parameterization of the Controller After the identification one of the following methods respectively

rules for setting the control parameters can be used:

1. tables (e.g. empirical formula „which controller to which controlled system“, see [4]),
2. analytical points of view (e.g. structure optimized controllers),
3. parameter adjustment strategies:
 - (a) compensation method [5] (identification via step response, cp. Fig. 2, or via open loop frequency response, see [6])
 - (b) via adjusting the control parameters in situ, if the controlled system is not critical
 - (c) simulation of the closed loop and use of empirical control settings.

Compensation Method In this paper the compensation method will be used. The idea behind this method is to implement the controller as an inverse function of the already identified controlled system enhanced with an additional integral time (T_K).

Then the controller of a PT2 system looks like:

$$K(s) = \frac{1}{G(s)} \cdot \frac{1}{T_K s} = \frac{2D_S T_S}{k_S T_K} \cdot \left(1 + \frac{1}{2D_S T_S} + \frac{T_{SS}}{2D_S} \right) \quad (2)$$

This control structure is equivalent to a PID control structure.

$$K(s) = k_R \cdot \left(1 + \frac{1}{T_I s} + T_D s \right) \quad (3)$$

k_R	gain of the controller
T_I	integral time
T_D	derivative time

The parameters can be found by comparing the coefficients,

$$T_I = 2D_S T_S, \quad (4)$$

$$T_D = \frac{T_S}{2D_S}, \quad (5)$$

$$k_R = \frac{2D_S T_S}{k_S T_K} = \frac{T_I}{k_S T_K}. \quad (6)$$

When $T_K = T_I$ is assumed, from (4) and (6) the conherence $k_R = k_S^{-1}$ can be deduced. The transfer function of the closed loop is, however, a first order

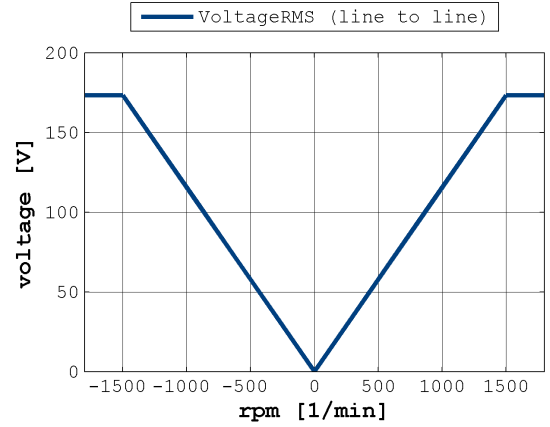


Figure 3: Dependence between desired setpoint value of the voltage and speed.

system. The value of k_R determined using the assumption $T_K = T_I$ can be used as an initial setting during the initial operation phase. It is possible to increment the gain of the controller to enhance the dynamic performance of the whole controlled system (e.g. five times the value of k_R could be a feasible setting).

Principle of the Voltage Measurement – Sensing of the Actual Value

The actual value of the line-to-line voltage of the (main) synchronous machine has to be measured and compared with the speed depended, desired set point value of the voltage, as shown as in Fig. 3. The resultant deviation from the desired value feeds the voltage controller, which influences the controlled system in such a way, that the deviation from the desired value decreases.

Implementations of voltage controllers often measure the main generator voltage with a three phase measuring transformer. In practice the secondary alternating voltage is rectified, smoothed by a capacitor and then measured. This measured value corresponds with the arithmetic mean value of the rectified three-phase voltage. In the simulation environment the rectification of the secondary alternating voltage is performed by an algebraic algorithm that determines the maximum value of the absolute values of the line-to-line voltages. To minimize non linear load effects due to the rectifier or static thyristor inverters (noise), the rectified three-phase measuring voltage is low pass filtered. The attenuation characteristic of the chosen low pass filter is "critical", which means, that the low pass filter is a non-oscillating Butterworth filter (asymptotic case). Taking into account the dynamic quality factor

on the one hand and the ripple suppression on the other hand the choice of a 3rd order butterworth low pass filter is useful. The reference filter decreases the dynamic performance of the whole controlled system but it is necessary to avoid instabilities due to the superposed noise in the actual value of the main generator voltage.

Predetermined Desired Value of the Line-to-Line Voltage (Speed Dependent) Quite often the shaft of the main synchronous generator is driven by a diesel engine and the predetermined desired value of the voltage of the main generator can be derived from the mechanical speed of the generator shaft: if the absolute value of the rotational speed decreases, the desired value of the voltage of the main generator shall decrease proportionally, too. If the mechanical shaft speed is too high, the desired value of the line-to-line voltage shall be limited to the nominal generator voltage. Furthermore the desired value of the voltage is a positive quantity, because we assume that the sense of the rotation (e.g. of a diesel-driven generating set) does not change. The coherence between speed and voltage is depicted in Fig. 3.

Rectifier with Electrical Load In a standard application a traction generator feeds a diode rectifier with an intermediate circuit and the electrical load, for instance inverter fed traction motors. If the inverter is working with a very high pulse rate, it is possible to model the inverter and the traction motors as a constant-power sink. Figure 4 shows the electrically excited synchronous machine with the diode rectifier, the smoothing capacitor in the DC circuit and the modelled constant-power sink (for example 15kW).

Mechanical Drive To simplify the simulation we assume, that the driving engine runs with constant mechanical speed, guaranteed by the speed controller of the diesel-driven generating set. The rotational speed ripple effect due to the inherent torque ripple, mainly caused by the diesel engine and, to a lesser extent, from the main synchronous generator (due to feeding the diode rectifiers), is neglected.

2.1.1 Initial Operation Phase of the Voltage Controller

The simulation environment Dymola includes the Modelica Standard Library. This library is a standardized and free package that is developed together

with the Modelica language and provides model components in many domains that are based on standardized interface definitions. One domain is the so called Machines Library. This package contains components to model electrical machines, specially three phase induction machines, based on space phasor theory. The following types of machines, however, can be modelled:

- three phase asynchronous induction machines
- three phase synchronous induction machines
- DC machines with different excitation

In this paper machine parameters for the main synchronous machine and the auxiliary exciting synchronous generator (with rotating three-phase winding and stationary excitation) are taken from the standard model "Electrical excited synchronous induction machine with damper cage" of the Machines library.

2.1.2 Machine Parameters

Machine parameters (main synchronous generator):

- rotor's moment of inertia = 0.29 kg m²
- number of pole pairs (Integer) = 2
- warm stator resistance per phase = 0.03 Ω
- stator stray inductance per phase = $0.1 / (2 \cdot \pi \cdot f_{Nominal})$ H
- main field inductance in d-axis = $1.5 / (2 \cdot \pi \cdot f_{Nominal})$ H
- main field inductance in q-axis = $1.5 / (2 \cdot \pi \cdot f_{Nominal})$ H
- damper stray inductance in d-axis = $0.05 / (2 \cdot \pi \cdot f_{Nominal})$ H
- damper stray inductance in q-axis = $0.05 / (2 \cdot \pi \cdot f_{Nominal})$ H
- warm damper resistance in d-axis = 0.04 Ω
- warm damper resistance in q-axis = 0.04 Ω
- nominal stator RMS voltage per phase = 100 V
- nominal frequency $f_{Nominal}$ = 50 Hz
- no-load excitation current @ nominal voltage and frequency = 10 A

- warm excitation resistance = 2.5Ω
- stray fraction of total excitation inductance = 2.5%

Excitation machine's parameters (simplified synchronous generator):

- number of pole pairs (Integer) = 3
- warm stator resistance per phase = $5/3 \cdot 0.03 \Omega$
- stator stray inductance per phase = $5/3 \cdot 0.1 / (2 \cdot \pi \cdot f_{Nominal})$ H
- main field inductance in d-axis = $5/3 \cdot 1.5 / (2 \cdot \pi \cdot f_{Nominal})$ H
- nominal stator RMS voltage per phase = 25 V
- nominal frequency $f_{Nominal} = 75$ Hz
- no-load excitation current @ nominal voltage and frequency = 1 A
- warm excitation resistance = 25Ω

2.1.3 Setting of the Control Parameters

If the main synchronous machine and the auxiliary exciting synchronous generator are modeled with the parameters specified in the previous subsection, the whole system including the voltage PID controller can be arranged and simulated. For the given machine parameters, the least mean square method leads to the following parameters of the controlled system results:

$k_S = 7.2387$	gain of the controlled system
$\omega = 14.01 \text{ s}^{-1}$	oscillation frequency of the undamped assumed system
$T_S = 0.071375 \text{ s}$	reciprocal value of the oscillation frequency of the controlled system
$D_S = 1.39$	attenuation ratio ($D_S \geq 1$) aperiodic system with real singularities $s = -T_1^{-1}$ resp. $s = -T_2^{-1}$
$T_1 = 0.16841 \text{ s}$	time constant T_1
$T_2 = 0.03025 \text{ s}$	time constant T_2

With these numeric values the parameter settings of the voltage PID controller can be determined according to (4)–(6):

$T_I = 2D_S T_S = 0.19867 \text{ s}$	integral time
$T_D = \frac{T_S}{2D_S} = 0.02565 \text{ s}$	derivative time
$k_R = \frac{T_I}{k_S T_K} = 0.13815$	gain of the controller

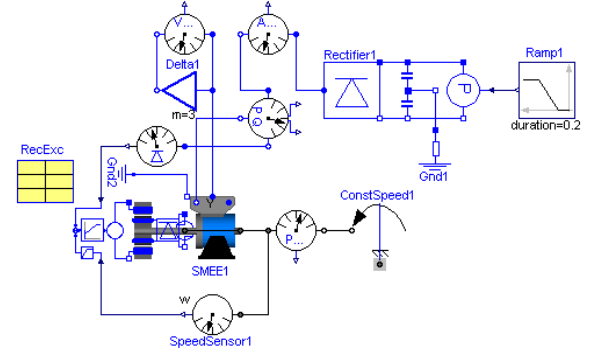


Figure 4: Closed loop with constant-power sink.

2.2 Simulation Results of the Three-Phase Generator for a Traction Application

Using the simulation environment Dymola it is possible to simulate and test the command action (Fig. 5) of the closed loop (Fig. 4) for plausibility reasons. At time zero, the auxiliary exciting synchronous generator and the main synchronous machine (without electrical load) will be accelerated until the mechanical nominal speed of the traction application is reached ($= 1500 \text{ min}^{-1}$). After the transient, the value of the desired line-to-line voltage will be achieved (rms phase-to-phase voltage is equal to $\sqrt{3} \cdot 100 \text{ V}$). In Fig. 5 the actual rms values of the filtered and the unfiltered line-to-line voltages are displayed. Using a constant power sink model, 1s later, a load step from 0 to 15 kW, is applied. The effect of the voltage ripple and the harmonic losses caused by the full bridge rectifier are discussed in the following section.

3 Simulation of the Harmonic Losses

The estimation of the harmonic losses due to the current ripple is performed with the entire model shown in Fig. 4. In particular, the observation of additional harmonic losses in the stator winding and the excitation circuit of the main synchronous machine is of interest, because in the beginning of the design process it is important to know, how these additional harmonic losses influence the size of the machine and if the reduction of masses is possible. As depicted in Fig. 6, the stator current shows a significant current ripple. Are serious additional harmonic losses the consequence? The higher the losses the higher is the thermal stress and the larger is the resultant size of the generator. If the simulation legitimates that these additional losses are not relevant, a reduction of masses is suggested and

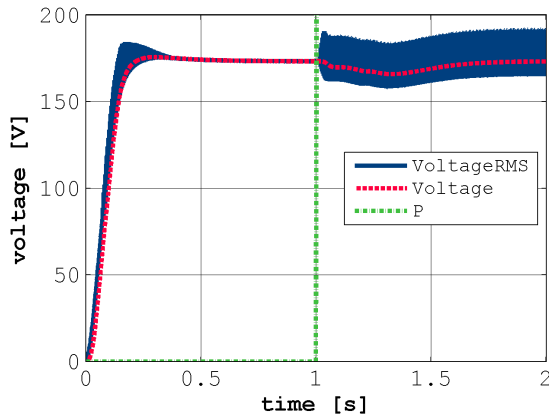


Figure 5: Observation of the unfiltered and filtered (smooth curve) voltage between the terminals: command action of the closed loop if an increase of power occurs (from 0kW to 15kW; activation of the constant-power sink after 1 s).

this results in a considerably energy saving of the drive train operation.

The voltage ripple due to effects caused by the full bridge rectifier can be observed in Fig. 5 during the transient in the beginning of the simulation and after applying a load step from 0 to 15kW. The salient distortion in the intermediate circuit voltage arise due to currents in the stator winding, caused by the mechanical load or the inherent moment of inertia of the entire rotating masses, and from switching events of the rectifying diodes (Fig. 6). A theoretical work about rectifying effects like current ripples and voltage distortion caused by converters can be found in [7].

The assumptions for the simulation of the harmonic losses are:

- main synchronous machine with constant reactances
- excitation synchronous generator with rotating rectifier bridge
- voltage controller with optimized parameters
- rectifier bridge on load side

Further boundary conditions:

- constant mechanical speed = 1500 rpm
- desired rms phase-to-phase voltage is equal to $\sqrt{3} \cdot 100 \text{ V}$

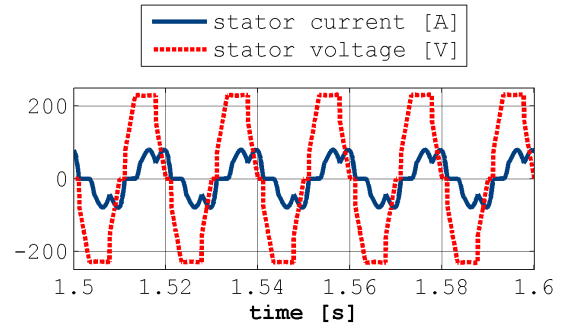


Figure 6: Current ripple due to diode switching effects. The observation of additional harmonic losses in the stator winding and the excitation circuit of the main synchronous machine is of interest.

This results in a corresponding intermediate circuit voltage of 233.9 V according to:

$$\frac{V_{DC}}{V_{RMS}} = \frac{6}{2\pi} \int_{-\frac{2\pi}{12}}^{\frac{2\pi}{12}} \sqrt{2} \cos(x) dx = \sqrt{2} \frac{3}{\pi} \quad (7)$$

- load step from 0 to 15kW by means of a power sink

The observation of the intermediate circuit voltage and current during the above mentioned increase of power is depicted in Fig. 7. During the initial no load operation ($t < 1 \text{ s}$), the current of the intermediate circuit is zero and the intermediate voltage reaches the peak value of the rectified terminal voltage. When the machine is loaded with 15kW ($t > 1 \text{ s}$), the intermediate current increases and the intermediate voltage reaches 233.9V, according to the desired pre-set value of the rectified terminal voltage, as determined by (7). The simulation result in Fig. 7 shows, that with the controlled synchronous machine the desired intermediate circuit voltage under load conditions can be reached and that the current in the DC link arise accordingly.

Discussion of the Simulation Results The stator current ripple and the square wave form of the stator voltage (note: *not* sinusoidal) depicted in Fig. 6, suggest a further inspection of the simulation results. By means of a harmonic distortion factor estimation, the additional losses can be estimated.

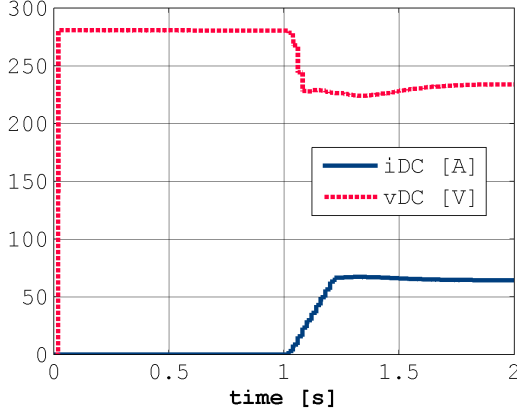


Figure 7: Observation of the intermediate circuit voltage and current if an increase of power occurs (from 0kW to 15kW; activation of the constant-power sink after 1 s).

The total harmonic distortion factor (THD) of the current in the stator winding of the main synchronous machine can be calculated by:

$$THD_s = \sqrt{\frac{\sum_{v=2}^{\infty} I_{sv}^2}{I_{s1}^2}} = 30\%$$

This results in additional harmonic losses about 9% in the main synchronous machine.

The total harmonic distortion factor (THD) of the current in the excitation circuit of the main synchronous generator can be calculated according to:

$$THD_e = \sqrt{\frac{\sum_{v=1}^{\infty} I_{ev}^2}{I_{e0}^2}} = 7.6\%$$

This results in additional harmonic losses about 0.6% in the excitation winding of the main synchronous generator.

These results are discovering, that the entire additional harmonic losses are not so severe and can be neglected when starting the design process of traction generators.

4 Conclusion

In this paper an electrically excited synchronous machine feeding a diode bridge rectifier, as used in traction applications, is investigated. The effects of this operation mode, particularly with respect to losses

due to the harmonics, can be simulated in the beginning of the design process for revealing possible optimizations. The presented innovative simulation model is based on the object oriented language Modelica. Based on the simulation tool Dymola with the provided comprehensive Modelica Standard Library, the electromagnetic behavior of the traction generator can be modelled for typical operation modes. To achieve appropriate terminal voltages of the traction generator under such typical operation conditions, it is necessary to implement an excitation voltage controller for the auxiliary excitation generator, which in turn determines excitation voltage and current of the main generator, respectively. The choice and the design of the controller has to be performed after identifying the controlled system. The identification is performed applying a step response. The control parameter are determined using the compensation method. After that it was possible to simulate the entire traction generator model. The analysis of the additional harmonic losses due to the full bridge rectifier shows, that the entire additional harmonic losses are not so severe and can be neglected in the beginning of a design process for traction generators.

References

- [1] H. Kleinrath, *Stromrichter gespeiste Drehfeldmaschinen*. Wien: Springer Verlag, 1980.
- [2] C. Kral and A. Haumer, "Modelica libraries for dc machines, three phase and polyphase machines," *Modelica Conference*, pp. 549–558, 2005.
- [3] H. Kleinrath, *Grundlagen elektrischer Maschinen*. Wiesbaden: Akademische Verlagsgesellschaft, 1975.
- [4] A. Weinmann, *Regelungen, Analyse und technischer Entwurf*, vol. 1. Wien: Springer Verlag, 3 ed., 1983.
- [5] O. Föllinger, *Regelungstechnik*. Heidelberg: Hüthig Verlag, 8 ed., 1994.
- [6] D. Schroeder, *Elektrische Antriebe- Regelung von Antriebssystemen*. www.springer.de: Springer, 2 ed., 2001.
- [7] J. Cano, G. Orcajo, J. Mayordomo, R. Asensi, M. Cabanas, and M. Melero, "New transfer functions of an accurate estimation of harmonic distortion in AC/DC converter working under unbalanced conditions," *Conference Proceedings of the*

IEEE Symposium on Electrical Machines, Power Electronics and Drives, SDEMPED, pp. 409–414, 1999.

Session 6a

Language, Tools and Algorithms 5

Dynamic Optimization of Energy Supply Systems with Modelica Models

CHR. Hoffmann

H. Puta

Technische Universität Ilmenau
P.O. Box 10 05 65, 98694 Ilmenau, Germany

Abstract

The paper describes a design tool for decentralised energy supply by preferable use of renewable energy. The powerful modelling language Modelica has been used to develop a model library for input data generation and/or predicting and modelling the main parts of the energy supply system for the simulation tool Dymola. An optimal control design using the well suited Dymola model in connection with Matlab [21] is performed with the solver HQP [15]. Application to a solar heating supply system with two storages demonstrates the effect of minimizing the use of additional heating and the realization by a model predictive control strategy.

Introduction

To build a moor it needs about 50 000 years, the youngest brown coal dates from 65 million years ago, about 1850 the first coal (also oil and a bit later gas) was excavated and newest estimations predict that coal, oil and gas from soil may be spent in about 2250, completely. That means human activity has consumed all the natural resources developed in millions of years within a delta impulse on time scale i.e. within about 400 years!

Realizing these facts it is high time that renewable energy sources become more and more important, mainly due to the shortage of fossil fuels, which is well known. Fortunately, we still can dispose over mixed energy supply, with great challenges arising for planning and control of heat supply systems, if these systems also work with different renewable energy sources (earth heat, solar radiation, exhaust air heat from building), which lead to control problems with a higher rank of complexity. The main goal of these control strategies (taking into account the weather of the past and predictions of energy requirement in the future) is the re-

duction of operational cost as well as an increase in the rate of return for the investment.

The paper describes the simulation of a heat supply system by means of object-oriented modelling with Dymola/Modelica. The final aim of such a simulation is to develop a model predictive control strategy on the basis of the proposed object-oriented model in Dymola/Modelica. To develop such a control strategy it is necessary to gain knowledge about the heat demand of buildings and the power usage of its different components. The use of Computer Aided Engineering (CAE) can lead to a further reduction of development costs. A Dymola library named **RECOMB** has been created with the purpose of modelling and simulation of such heat supply systems. The name **RECOMB** [16] stands for **R**enewable **E**nergy **C**omponents **m**odelling and **o**pti**M**isation of **B**uildings. The library itself consists of multi sub-libraries and following sub-libraries of crucial importance:

- Weather
- Predict
- HESys
- Buildings

The components and the resulting models for the thermodynamic models of the sub-library HESys were validated by comparing them with existing simulation software [11], [12], while the building models were validated with help of German guidelines [9] as well as with other simulation software [8],[10]. Fig. 1 show the structures of the library RECOMB.

RECOMB

Buildings	
	Complete rooms
	Heat transfer
	Heating
	Simple Building
	Ventilator
	Wall elements
Consumer Characteristics	
	Power Consumption
	Water Consumption
Examples	
	Buildings
	HESys
	Weather
Functions	
	Material constants
HESys	
	Additions
	Heating
	RenewableEnergy Converter
	Storages
Icons	
Interfaces	
Sources and Sinks	
Weather	
	Predict
	Weather

Fig. 1: Structure chart of RECOMB

This library constitutes the basic of the dynamic optimization by HQP und discrete dynamic programming by Matlab. It uses the advantage of the object-oriented modeling over the illustration the material flows and information flows.

Optimization

In this article three different possibilities the dynamic optimization using the Dymola/Modelica model are shown:

1. Offline optimization by HQP and Simulink S-function interface
2. Model predictive control (MPC) by HQP and Simulink S-function interface
3. Discrete dynamic programming (DDP) by Matlab coupled with Dymola models via Simulink S-function interface

The main goals of the different optimization possibilities are the maximization the renewable energy part to the supply of buildings and the analysis of the performance of the methods. The general continuous-time optimization problem to be solved is:

$$I(\mathbf{x}(t), \mathbf{u}(t)) = \vartheta[\mathbf{x}(t_f)] + \int_{t_0}^{t_f} \Phi[\mathbf{x}(t), \mathbf{u}(t), t] dt \rightarrow \min_{\mathbf{u}(t)}$$

with the constrains:

$$\mathbf{x}(t_0) = \mathbf{x}_0 \text{ (initial states)}$$

$$\mathbf{g}[\mathbf{x}(t_f)] = 0 \text{ (final state constraints)}$$

$$\mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), t] = 0 \quad \forall t \in [t_0, t_f] \text{ (equality constraints)}$$

$$\mathbf{h}[\mathbf{x}(t), \mathbf{u}(t), t] \leq 0 \quad \forall t \in [t_0, t_f] \text{ (inequality constraints)}$$

The form of this criterion is referred to as Bolza functional with Mayer term (end state evaluation) and Lagrange term (evaluation of the energy).

A first order differential equation system describes the system to be optimized in the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), t] \quad t \in [t_0, t_f]$$

The vector $\mathbf{x}(t)$ describes the state vector and the vector $\mathbf{u}(t)$ the control vector.

The optimization tool HQP is a numerical optimal solver and before this tool can be used for the solution of the optimization problem, the problem must be discretized. The exactly discretized optimization problem is shown in subsection 3. The simple form of the general nonlinear optimization problem is:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{u}^{K-1} \end{pmatrix}$$

$I(\mathbf{x}) \rightarrow \min_{\mathbf{x}}$ with $\mathbf{c}_{ineq}(\mathbf{x}) \leq 0$ (inequality constraints) and $\mathbf{c}_{eq}(\mathbf{x}) = 0$ (equality constraints)

1. Offline optimization by HQP and Simulink s-function interface

For dynamic optimization using the Dymola Modelica model the following criterion has to be maximized:

$$I(u) = \int_{t_0}^{t_f} [a_1 Q_{cum}(t) - a_2 E_F(t) + a_3 Q_{HP}(t)] dt$$

The weighting factors a_1 to a_3 allow to adjust proper priorities, if the use of fossil energy (E_F) should be minimized and renewable energy (Q_{cum} and Q_{HP}) should be preferably used, where the index “cum” stands for cumulated solar energy and “HP” for heat pump. Because the powerful optimization tool HQP [15] had worked very well together with Matlab, the Dymola/Modelica model of the system was transferred to Matlab/Simulink. Fig. 2 shows the steps of optimization.

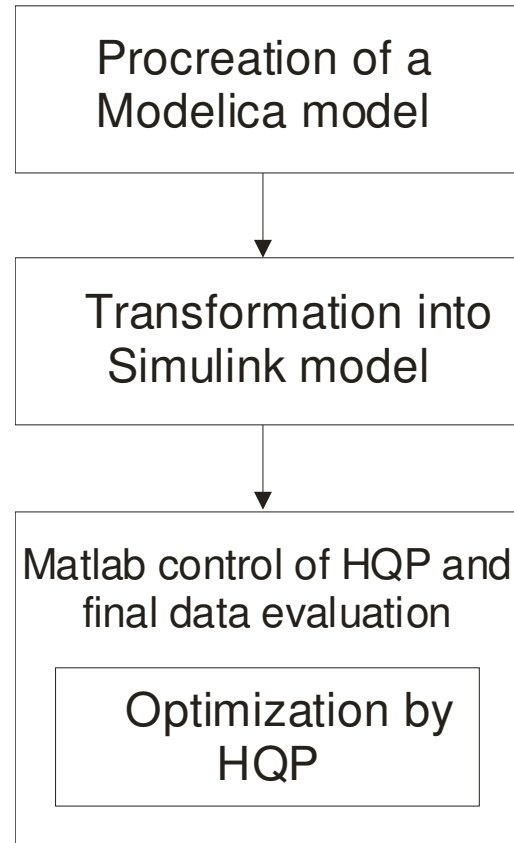


Fig. 2: Steps of optimization

The model of a two storage solar heating system was used to demonstrate how optimization works and to make sure the load strategy for these two storages will provide minimal additional heating.

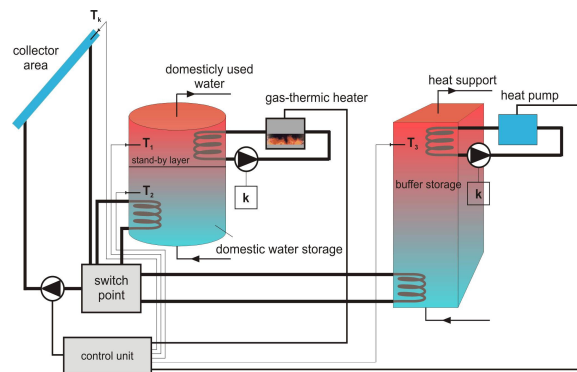


Fig. 3: Two storage heating system

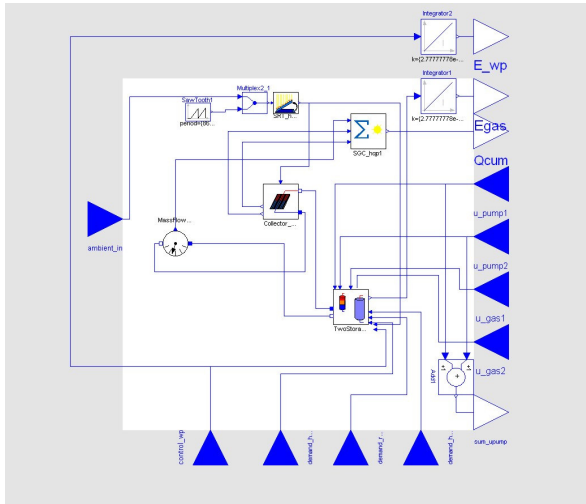


Fig. 4: Modelica model from two storage heating system

Fig. 3 shows this plant with the following data:

- collector area: 5m^2
- short term storage with a stand-by layer of 135 l and a rest part of 165 l, i.e. together 300 l of content.
- long term (buffer) storage of 1000 l
- heat pump: 5 KW
- gas burner: 10KW.

Fig. 4 shows the two storage heating system as a Dymola/Modelica model with different In-ports und Outports.

For safety the circulating pumps for heat pump and gas burner are always working, even when both devices are not in action, indicated by k. The aim of optimization is to calculate the load dependent control strategies, i.e. for the switch between the two storages, the heat pump and the gas burner. Constraints have to be taken into account such as: minimum temperature between short term storage and long term storage of 333K, control constraints with respect to mass flow of gas, electrical power of the heat pump (5 KW) and the mass flow of the variable pump for storages (0.139 kg/s). By weighting factors a_1 to a_3 in the cost functional regenerative energy use may be preferably controlled. For the example (Fig. 4) the following factors were used:

$a_1 = 700$, $a_2 = 100$, $a_3 = 200$. By these values a better scaling of the optimization problem is achieved. Figs. 6 - 10 show that the given constraints can be met.

For online application a model predictive control scheme was developed (Fig. 10), tested with the one storage and the two storage system. The results show, compared with tradi-

tionally two-point control, a win of about ten percent and better behaviour for short term weather change, which is very important for real applications.

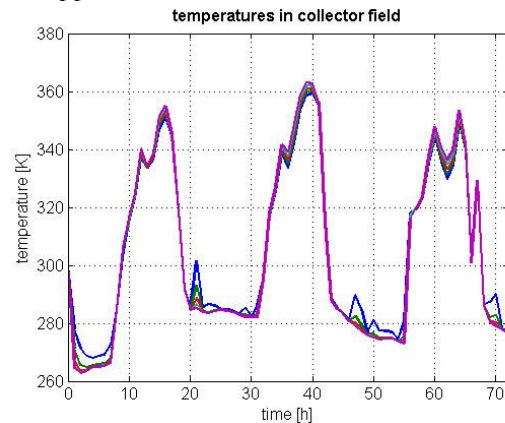


Fig. 6: Temperature of different collector areas

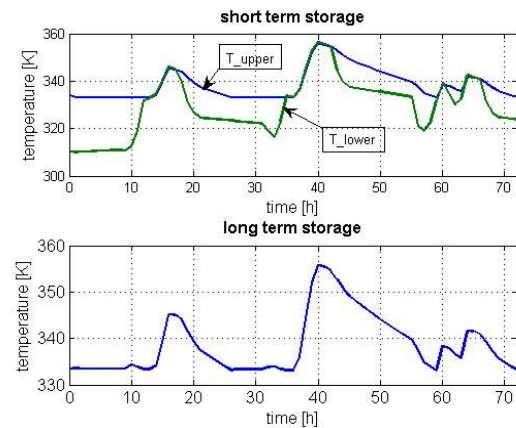


Fig. 7: Storage temperature courses

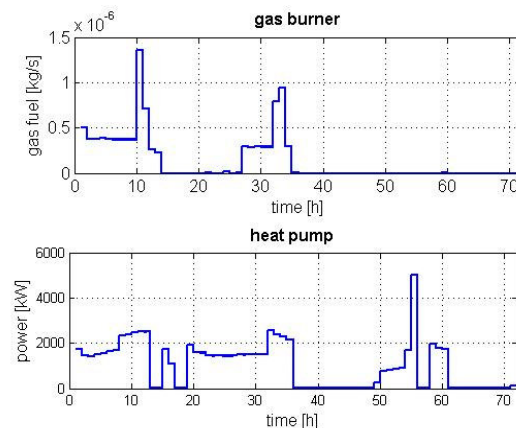


Fig. 8: Optimal courses for burner and heat pump

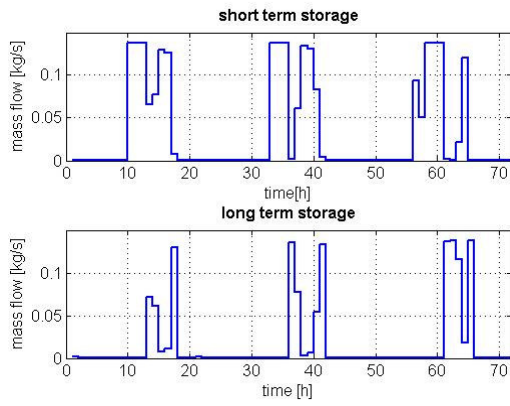


Fig. 9: Optimal loading courses

2. Model predictive control (MPC) by HQP and Simulink s-function interface

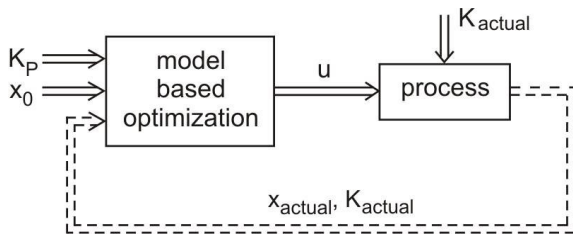


Fig. 10: Model predictive control scheme

For the optimization with MPC-technique [19] a simple plant model with one storage was used. Fig. 11 shows this plant with the following data:

- Collector area: 5m^2
- short term storage with a stand by layer of 135 l and a rest part of 165 l, i.e. together 300 l of content.
- gas burner: 10KW.

In Fig. 10 the variables have the following meaning: $x_0 = x(0)$, initial state, used for optimization, x_{actual} are the actual start values of x after some time steps, K_p are forecasted climate data for starting the optimization with x_0 , K_{actual} are the actual climate values after some time steps used also as starting values for optimization. As usual the MPC (model predictive control) is based on repetitive optimization, i.e. the control loop is only closed within discrete time steps to include actual data from the different sources (states, forecasts, measured data) for prediction of new optimized values (control and parameter).

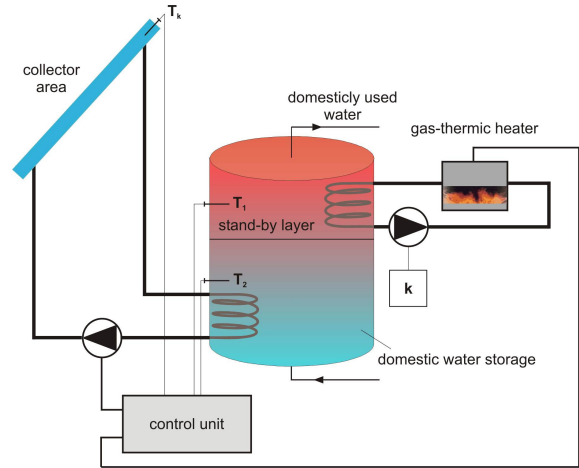


Fig. 11: One storage heating system

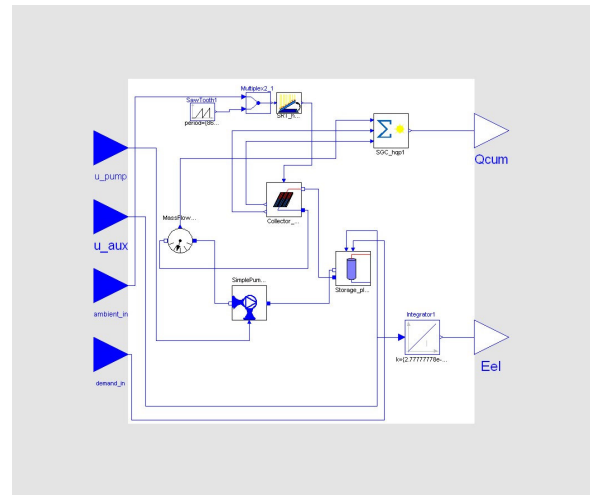


Fig. 12: Dymola/Modelica model from one storage heating system

For dynamic optimization using the Dymola/Modelica model with on storage the following criteria has to be maximized:

$$I(u) = \int_{t_0}^{t_f} Q_{\text{cum}}(t) dt$$

By means of this criterion only the solar energy was maximized this automatically implies the minimization of the fossil energy. The index “cum” stands again for cumulated solar energy. The horizon for the optimization in this example was three hours and the control was applied over a time horizon of two hours. Then the optimization cycles were repeated. In Fig. 14 the results of the MPC are shown.

The advantage of the method is the reaction to altered data from weather or the plant in comparison to the offline optimization. Especially,

Fig.13 shows the temperatures of the stand-by layer. Through reaction of the altered weather data with MPC-technique the limit can be met.

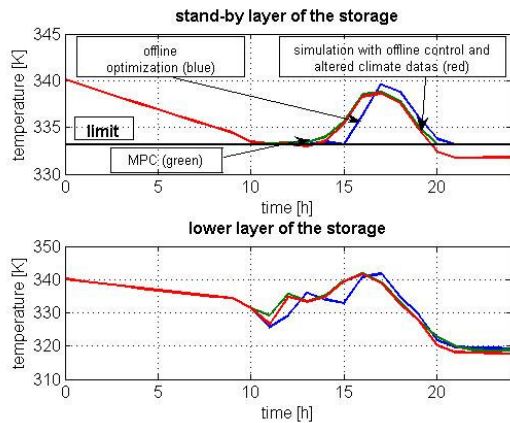


Fig. 13: Storage temperature courses

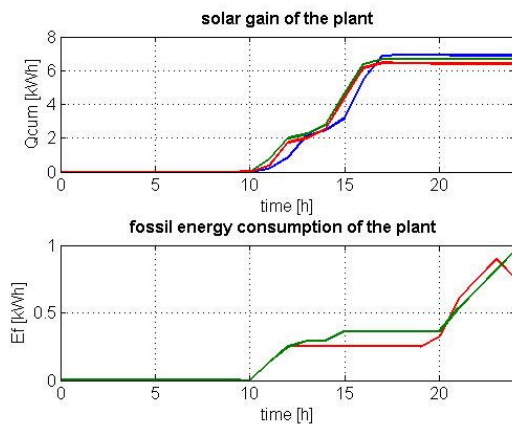


Fig. 14: Result courses

3. Discrete dynamic programming (DDP) by Matlab coupled with Dymola models via Simulink S-function interface

DDP [20] was applied to the same one storage heating system how in Fig. 11 shown in. The problem with this possibility is that the plant should not have too many states und control parameters because the numbers of variables increase exponentially. The problem formulation of the DDP is:

System description:

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), k], \quad k = 0, \dots, K-1$$

Criterion with constrains:

$$I_k = \vartheta[\mathbf{x}(K)] + \sum_{k=0}^{K-1} \Phi[\mathbf{x}(k), \mathbf{u}(k), k], \quad K \text{ fixed}$$

$$\mathbf{g}[\mathbf{x}(K)] = 0 \text{ (final states constraints)}$$

$$\mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), k] = 0, \quad k = 0, \dots, K-1 \text{ (equality constraints)}$$

$$\mathbf{h}[\mathbf{x}(k), \mathbf{u}(k), k] \leq 0, \quad k = 0, \dots, K-1 \text{ (inequality constraints)}$$

The calculation algorithm of the DDP [18] is divided into three parts:

1. Discretization of the considered continuous problem
2. Backward calculation – one stage minimization to calculate optimal transitions of the states (optimal control law)
3. Forward calculation – investigation of the optimal states trajectories and the control parameter courses from the optimal control law.

In the following figures a comparison between HQP and DDP should be shown. The aim of the transformation from HQP to DDP is the insertion to build a hierarchical controller for the heat supply system for buildings on the basis of an embedded microcontroller system. Fig. 15 shows the temperature in stand-by layer of the storage and the collector field temperature. The temperature in the lower layer of the storage is similar to the collector field temperature. The differences of courses are often problems of the discretization intervals of the:

- state with $\Delta \mathbf{x}$ and the
- control parameters with $\Delta \mathbf{u}$

between DDP and HQP.

In case of DDP the discretization is bigger than in case of HQP because of a strong increase of the CPU-time.

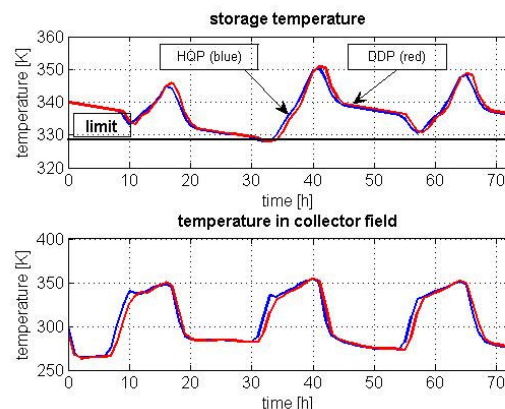


Fig. 15: Storage temperature courses

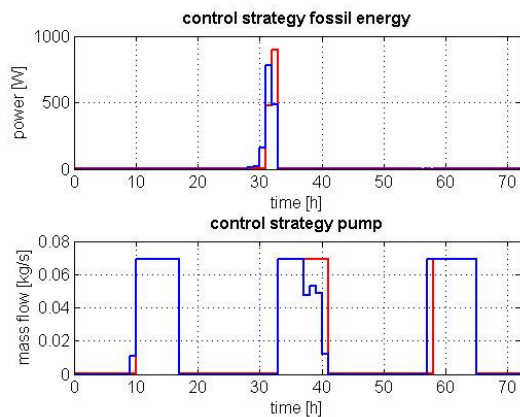


Fig. 16: Storage temperature courses

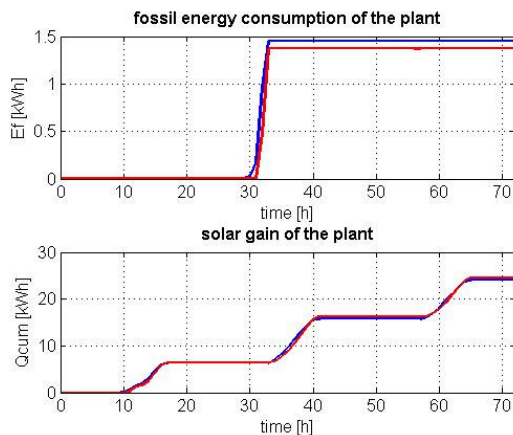


Fig. 17: Storage temperature courses

Conclusion

As a result of this article can be said that dynamic optimization from heating supply plants with Dymola/Modelica models is advantageous. The application of different optimization techniques shows the universal character of the Dymola/Modelica models. Especially the graphical programming of the plant model is of great value and it makes a relatively fast investigation of control trajectories.

References

- [1] V. Quaschnig, (1999), Regenerative Energiesysteme, 2. Auflage, Carl Hanser Verlag
- [2] Duffie and Beckmann, (1991), Solar Engineering of Thermal Processes
- [3] H. P. Garg, S.C. Mulick and A.K. Bhargava, (1985) Solar Thermal Energy Storage, D.Reidel Publishing Company
- [4] Dymola User's Manual, (2002), Dynamsin AB Lund
- [5] J. Kahler, (2002), Entwicklung einer Gebäudekomponentenbibliothek für kontrolliert-

belüftete und -entlüftete Niedrigenergiehäuser, Diplomarbeit TU Ilmenau, Inst. für Automatisierungs- und Systemtechnik

- [6] M. Kisser, (2002), Entwicklung einer Prognoseverfahrensbibliothek fuer Klimadaten in Dymola/Modelica, Diplomarbeit TU Ilmenau, Inst. für Automatisierungs- und Systemtechnik
- [7] W. Feist, (1994), Thermische Gebäudesimulation: kritische Prüfung unterschiedlicher Modellierungsansätze

[8] C. Nytsch-Geusen, (2001), Berechnung und Verbesserung der Energieeffizienz von Gebäuden und ihren energietechnischen Anlagen in einer objekt-orientierten Simulationsumgebung, Dissertation, TU Berlin

[9] VDI Richtlinie 6020, (2001), Anforderungen an Rechenverfahren zur Gebäude und Anlagensimulation

[10] F. Felgner et al., (2002), Simulation of thermal building behaviour in Modelica, Proceedings of the 2nd International Modelica Conference

[11] Solar-Institut Juelich, (1999), CARNOT Blockset Version 1.0 User manual

[12] Dr. Valentin EnergieSoftware GmbH, (1999), T-Sol 2.0 User manual

[13] Franke, R. (1998): Integrierte dynamische Modellierung und Optimierung von Systemen mit saisonaler Wärmespeicherung; Fortschritts-Berichte VDI, Reihe 6, Nr. 394

[14] MODELICA: Modeling of Complex Physical Systems. Modelica-Homepage.

<http://www.modelica.org>

[15] HQP: a solver for sparse nonlinear optimization. Omuses/HQP-Homepage.

<http://hqp.sourceforge.net>

[16] Ch. Hoffmann, J. Kahler, Puta, H.(2003): Object-oriented simulation of energy supply systems on the basis of renewable energy. In [Proceedings of the 3rd International Modelica conference](#), pp. 189-196, November 3-4, Linköping, Sweden.

[17] Liu, B.Y.H. and Jordan, R.C.(1960): The interrelationship and characteristic distribution of direct, diffuse and total solar radiation, Solar Energy 4

[18] M. Papageorgiou, Optimierung, 2. Auflage, Oldenbourg Verlag

[19] S. Martin, (2005), Optimierte Waermeverorgung von Gebaeuden, Diplomarbeit TU Ilmenau, Inst. für Automatisierungs- und Systemtechnik

[20] T. Flemming, (2005), Optimierte Steuerung von Energieversorgungsanlagen mit Hilfe

der Diskreten Dynamischen Programmierung,
Diplomarbeit TU Ilmenau, Inst. für Automati-
sierungs- und Systemtechnik
[21] Matlab: Matrix laboratory. Mathworks-
Homepage.
<http://www.mathworks.com/>

Robust Initialization of Differential Algebraic Equations

Bernhard Bachmann, Peter Aronsson*, Peter Fritzson⁺

Dept. Mathematics and Engineering, University of Applied Sciences,
D-33609 Bielefeld, Germany
bernhard.bachmann@fh-bielefeld.de

* MathCore Engineering AB, Teknikringen 1B, SE-583 30 Linköping, Sweden
peter.aronsson@mathcore.com

+ PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
petfr@ida.liu.se

Abstract

This paper describes a new solution method applied to the problem initializing DAEs using the Modelica language. Modelica is primarily an object-oriented equation-based modeling language that allows specification of mathematical models of complex natural or man-made systems. Major features of Modelica are the multidomain modeling capability and the reusability of model components corresponding to physical objects, which allow to build and simulate highly complex systems. However, initializing such models has been quite cumbersome, since initial equations have to be provided at the system level, where the user needs to know details on the underlying transformation and index-reduction algorithms, that in general are applied to simulate a Modelica model.

The new initialization concept allows to define the initial equations locally in each relevant component where the corresponding states appear. This approach also works for arbitrary “well-posed” higher-index problems and makes the initialization of complex systems more user friendly. A prototype implementation in the OpenModelica compiler is presented, and test results of non-trivial application examples are reported.

1 Introduction

So far, using model initialization in Modelica has only been possible for higher-index problems if the user formulates the initial equations globally. This was also the case, e.g. when using the OpenModelica OpenModelica compiler which is an open source implementation developed at PELAB, Linköping University. In order to do such a global formulation successfully, the user needs to know about index reduction, at least the

number of freedom left after applying the dummy derivative method is necessary. Therefore, only advanced users have been able to use this feature in the Modelica language, when higher index problems occur (which is very common). In order to provide a more complete simulation environment, we have started to add robust initialization techniques to the OpenModelica compiler.

2 Flattening of a Modelica Model to a Hybrid DAE

A Modelica model is typically translated to a basic mathematical representation in terms of a flat system of *differential and algebraic equations* (DAEs) before being able to simulate the model. This translation process elaborates on the internal model representation by performing analysis and type checking, inheritance and expansion of base classes, modifications and redeclarations, conversion of connect-equations to basic equations, etc. The result of this analysis and translation process is a flat set of equations, including conditional equations, as well as constants, variables, and function definitions. By the term *flat* is meant that the object-oriented structure has been broken down to a flat representation where no trace of the object hierarchy remains apart from dot notation (e.g. `Class.Subclass.variable`) within names.

3 Mathematical Formulation of Hybrid DAEs

3.1 Summary of notation

Below we summarize the notation used in the equations that follow, with time dependencies stated explicitly for all time-dependent variables by the arguments t or t_e :

- $p = \{p_1, p_2, \dots\}$, a vector containing the Modelica variables declared as `parameter` or `constant` i.e., variables without any time dependency.
- t , the Modelica variable `time`, the independent variable of type `Real` implicitly occurring in all Modelica models.
- $x(t)$, the vector of state variables of the model, i.e., variables of type `Real` that also appear differentiated, meaning that `der()` is applied to them somewhere in the model.
- $\dot{x}(t)$, the differentiated vector of state variables of the model.
- $u(t)$, a vector of input variables, i.e., not dependent on other variables, of type `Real`. These also belong to the set of algebraic variables since they do not appear differentiated.
- $y(t)$, a vector of Modelica variables of type `Real` which do not fall into any other category. Output variables are included among these, which together with $u(t)$ are algebraic variables since they do not appear differentiated.
- $q(t_e)$, a vector of discrete-time Modelica variables of type `discrete Real`, `Boolean`, `Integer` or `String`. These variables change their value only at event instants, i.e., at points t_e in time.
- $q_{pre}(t_e)$, the values of q immediately before the current event occurred, i.e., at time t_e .
- $c(t_e)$, a vector containing all `Boolean` condition expressions evaluated at the most recent *event* at time t_e . This includes conditions from all if-equations/statements and if-expressions from the original model as well as those generated during the conversion of when-equations and when-statements.
- $rel(v(t)) = rel(cat(1, x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p))$, a `Boolean` vector valued function containing the relevant elementary relational expressions from the model, excluding relations enclosed by `no-Event()`. The argument $v(t) = \{v_1, v_2, \dots\}$ is a vector containing all elements in the vectors $x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p$. This can be expressed using the Modelica concatenation function `cat` applied to these vectors; $rel(v(t)) = \{v_1 > v_2, v_3 \geq 0, v_4 < 5, v_6 \leq v_7, v_{12} = 133\}$ is one possible example.
- $f(\dots)$, the function that defines the differential equations $f(\dots) = 0$ in (1a) of the system of equations.
- $g(\dots)$, the function that defines the algebraic equations $g(\dots) = 0$ in (1b) of the system of equations.

- $f_q(\dots)$, the function that defines the difference equations for the discrete variables $q := f_q(\dots)$, i.e., (2) in the system of equations.
- $f_e(\dots)$, the function that defines the event conditions $c := f_e(\dots)$, i.e., (3) in the system of equations.
- $f_x(\dots)$, the function that defines the reinitialization values for the continuous variables $x(t_e) := f_x(\dots)$ at events.

In the context of hybrid DAE:s the *state* of a system is not only made up of the values of the set of variables that occur differentiated in the model. The overall *state* of a system may also include values of discrete variables. In this paper the word *state* is used in this sense, including the state of the discrete part of the system.

3.2 Continuous-Time Behavior

Now we want to formulate the continuous part of the *hybrid DAE* system of equations including discrete variables. This is done by adding a vector $q(t_e)$ of *discrete-time variables* and the corresponding predecessor variable vector $q_{pre}(t_e)$ denoted by `pre(q)` in Modelica. For discrete variables we use t_e instead of t to indicate that such variables may only change value at event time points denoted t_e , i.e., the variables $q(t_e)$ and $q_{pre}(t_e)$ behave as constants between events.

We also make the constant vector p of *parameters and constants* explicit in the equations, and make the time t explicit. The vector $c(t_e)$ of condition expressions, e.g. from the conditions of `if` constructs and `when` constructs, evaluated at the most recent event at time t_e is also included since such conditions are referenced in conditional equations. We obtain the following *continuous DAE* system of equations that describe the system behavior *between* events:

$$\begin{aligned} f(x(t), \dot{x}(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) &= 0 \quad (a) \\ g(x(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) &= 0 \quad (b) \end{aligned} \quad (1)$$

3.3 Discrete-Time Behavior

Discrete time behavior is closely related to the notion of an event. Events can occur asynchronously, and affect the system one at time, causing a sequence of state transitions.

An event occurs when any of conditions $c(t_e)$ (defined below) of conditional equations changes value from `false` to `true`. We say that an event becomes *enabled* at the time t_e , if and only if, for any sufficiently small value of ε , $c(t_e - \varepsilon)$ is `false` and $c(t_e + \varepsilon)$ is `true`. An enabled event is *fired*, i.e., some behavior associated with the event is executed, often causing a discontinuous state transition.

Firing of an event may cause other conditions to switch from `false` to `true`. In fact, events are fired until a stable

situation is reached when all the condition expressions are false.

However, there are also state changes caused by equations defining the values of the *discrete* variables $q(t_e)$, which may change value *only* at events, with event times denoted t_e . Such discrete variables obtain their value at events, e.g. by solving equations in when-statements or evaluating assignments in when-statements. The instantaneous equations defining discrete variables in when-equations are restricted to particularly simple syntactic forms, e.g. $var = expr$; . These restrictions are imposed by the Modelica language in order to easily determine which discrete variables are defined by solving the equations in a when-equation.

Such equations can be directly converted to equations in assignment form, i.e., assignment statements, with fixed causality from the right-hand side to the left-hand side. Regarding algorithmic when-statements that define discrete variables, such definitions are always done through assignments. Therefore we can in both cases express the equations defining discrete variables as *assignments* in the vector equation (1a), where the vector-valued *function* f_q specifies the right-hand side expressions of those *assignments to discrete variables*.

$$q(t_e) := f_q(x(t_e), \dot{x}(t_e), u(t_e), y(t_e), t_e, q_{pre}(t_e), p, c(t_e)) \quad (2)$$

The last argument $c(t_e)$ is made explicit for convenience. It is strictly speaking not necessary since the expressions in $c(t_e)$ could have been incorporated directly into f_q . The vector $c(t_e)$ contains all `Boolean` condition expressions evaluated at the most recent *event* at time t_e . It is defined by the following vector assignment equation with the right-hand side given by the vector-valued function f_c . This function has as arguments the subset of the discrete variables having `Boolean` type, i.e., $q^B(t_e)$ and $q_{pre}^B(t_e)$, the subset of `Boolean` parameters or constants, p^B , and a vector $rel(v(t))$ evaluated at time t_e , containing the elementary relational expressions from the model. The vector of condition expressions $c(t_e)$ is defined by the following equation in assignment form:

$$c(t_e) := f_c(q^B(t_e), q_{pre}^B(t_e), p^B, rel(v(t_e))) \quad (3)$$

The argument $v(t) = \{v_1, v_2, \dots\}$ is a vector containing all scalar elements of the argument vectors. This can be expressed using the Modelica concatenation function `cat` applied to the vectors, e.g. $v(t) = cat(1, x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p)$. For example, if $rel(v(t)) = \{v_1 > v_2, v_3 \geq 0, v_4 < 5, v_6 \leq v_7, v_{12} = 133\}$ where $v(t) = \{v_1, v_2, v_3, v_4, v_6, v_7, v_{12}\}$, then it

might be the case that $c(t) = \{v_1 > v_2 \text{ and } v_3 \geq 0, v_{10}, \text{ not } v_{11}, v_4 < 5 \text{ or } v_6 \leq v_7, v_{12} = 133\}$, where v_{10}, v_{11} are `Boolean` variables and $v_1, v_2, v_3, v_4, v_6, v_7$ might be `Real` variables, whereas v_{12} might be an `Integer` variable. $rel(v(t)) = rel(cat(1, x(t), \dot{x}(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p))$, is a `Boolean`-typed vector-valued function containing the relevant elementary *relational expressions* from the model, excluding relations enclosed by `noEvent()`.

Discontinuous changes of continuous dynamic variables $x(t)$ can be caused by so-called `reinit` equations in Modelica. As in the case of discrete variables, such discontinuous changes can only occur at events. The effect of a `reinit`-equation that is activated at t_e is an assignment to the continuous variable at time t_e of the form:

$$x(t_e) := f_x(x(t_e), \dot{x}(t_e), u(t_e), y(t_e), t_e, q_{pre}(t_e), p, c(t_e)) \quad (4)$$

For all variables in $x(t_e)$ that are not affected by an `reinit`-equation $f_x(\dots)$ takes the value of $x(t_e)$, leaving the variable unchanged.

3.4 The Complete Hybrid DAE

The total equation system consisting of the combination of (1), (2), (3) and (4) is the desired *hybrid DAE* equation representation for Modelica models, consisting of *differential*, *algebraic*, and *discrete* equations.

This framework describes a system where the state evolves in two ways: continuously in time by changing the values of the state vector $x(t)$, and instantaneously during events triggered when some of the conditions $c(t_e)$ change value from `false` to `true`. The set of *state variables* from which other variables are computed is selected from the set of differentiated variables $x(t)$, algebraic variables $y(t)$, and discrete-time variables $q(t)$.

4 Simulation of Models Represented by Hybrid DAEs

4.1 Well-defined problem description

A Modelica *simulation problem* in the general case is a Modelica *model* that can be reduced to a hybrid DAE in the form of equations (1), (2), (3) and (4), together with additional constraints on variables and their derivatives called *initial conditions*.

The initial conditions prescribe initial start values of variables and/or their derivatives at simulation time=0 (e.g. expressed by the Modelica `start` attribute value of variables, with the attribute `fixed = true`), or default estimates of start values (the `start` attribute value with `fixed = false`).

The simulation problem is *well defined* provided that the following conditions hold:

- The total model system of equations is consistent and neither underdetermined nor overdetermined.
- The initial conditions are consistent and determine initial values for all variables.
- The model is specific enough to define a unique solution from the start simulation time t_0 to some end simulation time t_1 .

The initial conditions of the simulation problem are often specified interactively by the user in the simulation tool, e.g. through menus and forms, or alternatively as default `start` attribute values in the simulation code. More complex initial conditions can be specified through `initial equation` sections in Modelica.

4.2 Simulation Techniques

There are three different kinds of equation systems resulting from the translation of a Modelica model to a flat set of equations, from the simplest to the most complicated and powerful:

- ODEs – Ordinary differential equations for continuous-time problems.
- DAEs – Differential algebraic equations for continuous-time problems
- Hybrid DAEs – Hybrid differential algebraic equations for mixed continuous-discrete problems.

In the following we present a short overview of methods to solve these kinds of equation systems. However, remember that these representations are strongly inter-related: an ODE is a special case of DAE without algebraic dependencies between states, whereas a DAE is a special case of hybrid DAEs without discrete or conditional equations. We should also point out that in certain cases a Modelica model results in one of the following two forms of purely algebraic equation systems, which can be viewed as DAEs without a differential equation part:

- Linear algebraic equation systems
- Nonlinear algebraic equation systems

However, rather than representing a whole Modelica model, such algebraic equation systems are usually subsystems of the total equation system.

4.3 The Notion of DAE Index

The DAE index is an important property of DAE systems. Consider once more a DAE system on the general form (neglecting the hybrid part, parameters and constants):

$$F(x(t), \dot{x}(t), y(t), u(t)) = 0 \quad (5)$$

We assume that this system is solvable with a continuous solution, given an appropriate initial solution. There are several definitions of DAE *index* in the literature, of which the following, also called *differential index*, is informally defined as follows:

- The index of a DAE system **Error! Reference source not found.** is the minimum number of times certain equations in the DAE must be differentiated in order to solve $\dot{x}(t)$ as a function of $x(t)$, $y(t)$, and $u(t)$, i.e. to transform the problem into ODE explicit state space form.

The index gives a classification of DAEs with respect to their numerical properties and can be seen as a measure of the distance between the DAE and the corresponding ODE

An ODE system on explicit state space form is of index 0 since it is already in the desired form:

$$\dot{x}(t) = f(t, x(t)) \quad (6)$$

The following *semi-explicit* form of DAE system is of index 1 under certain conditions:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), y(t)) & (a) \\ 0 &= g(t, x(t), y(t)) & (b) \end{aligned} \quad (7)$$

The condition is that the Jacobian of g with respect to y , $(\partial g / \partial y)$ – usually a matrix – is *non-singular* and therefore has a well-defined inverse. This means that in principle $y(t)$ can be solved as a function of $x(t)$ and substituted into (7a) to get state-space form. A DAE system in the general form (5) may have higher index than one. Mechanical models often lead to index 3 DAE systems. We conclude:

- There is no need for symbolic differentiation of equations in a DAE system if it is possible to determine the *highest order derivatives* as continuous functions of time and lower derivatives using stable numerical methods. In this case the index is at most 1.
- The index is zero for such a DAE system if there are no algebraic variables.

4.4 Mixed Symbolic and Numerical Solution of higher-index DAEs

A mixed symbolic and numerical approach to solution of DAEs avoids the problems of numeric differentiation. The DAE is transformed to a lower index problem by using index reduction. The standard mixed symbolic and numeric approach contains the following steps:

1. Use Pantelides algorithm to determine how many times each equation has to be differentiated to reduce the *index* to one or zero.
2. Perform *index reduction* of the DAE by analytic symbolic differentiation of certain equations and by applying the method of dummy derivatives.
3. Select the core state variables to be used for solving the reduced problem. These can either be selected statically during compilation, or in some cases selected dynamically during simulation.
4. Use a numeric ODE solver to solve the reduced problem.

In the following we will discuss the notions of index and index reduction in some more detail.

4.5 Higher Index Problems are Natural in Component-Based Models

The index of a DAE system is not a property of the modeled system but the *property* of a *particular model representation*, and therefore a function of the modeling methodology. A natural object-oriented component-based methodology with reuse and connections between physical objects leads to high index in the general case. The reason is the constraint equations resulting from setting variables equal across connections between separate objects.

Since the index is not a property of the modeled system it is possible to reduce the index by symbolic manipulations. High index indicates that the model has algebraic relations between differentiated state variables implied by algebraic relations between those state variables. By using knowledge about the particular modeling domain it is often possible to manually eliminate a number of differentiated variables, and thus reduce the index. However, this violates the object-oriented component-based modeling methodology for physical modeling that is intended to be supported by the Modelica language.

We conclude that high index models are natural, and that automatic index reduction is necessary to support a general object-oriented component-based modeling methodology with a high degree of reuse.

5 Finding Consistent Initial Values at Start or Restart

As we have stated briefly above, at the start of the simulation, or at restart after handling an event, it is required to find a consistent set of initial values or restart values of the variables of the hybrid DAE equa-

tion system before starting continuous DAE solution process.

At the *start* of the simulation these conditions are given by the initial conditions of the problems (including start attribute equations, equations in initial equation sections, etc., together with the system of equations defined by (1), (2), and (3). The user specifies the initial time of the simulation, t_0 , and initial values or guesses of initial values of some of the continuous variables, derivatives, and discrete-time variables so that the algebraic part of the equation system can be solved at the initial time $t=t_0$ for all the remaining unknown initial values. In some application examples it is even necessary to calculate initial values of parameters (`fixed = false`), that afterwards be kept constant during simulation.

At *restart* after an event, the conditions are given by the *new values* of variables that have changed at the event, together with the current values of the remaining variables, and the system of equations (5), (6), and (7). The goal is the same as in the initial case, to solve for the new values of the remaining variables. In the initial case, however, the causality can be different since initial equations are included to calculate start values for the state variables, whereas at restart the state variables are always known.

6 Robust Initialization of Higher-Index DAEs

Initializing DAEs using the Modelica language has been quite cumbersome in the past, since initial equations have to be provided on the system level, where the user needs to know details on the underlying transformation and index-reduction algorithms, that are in general applied to simulate a Modelica model. Especially, when higher-index DAEs are involved the number of locally defined state variables no longer coincide with the number of state variables of the overall system. Although, one can influence the index-reduction algorithm by setting some attribute values (`stateSelect=always,prefer,...`), cases can be constructed which don't allow the straight forward prediction of the number of state variables left after transformation.

In order to make the initialization procedure more convenient a new concept is necessary, which allows to define the initial equations locally in each relevant component where the corresponding states appear, even if these states are eliminated during index-reduction. Naturally, this leads to an overdetermined system of equations, which has to be solved during the initialization process. In this context, we call a higher-index problem “well-posed” if enough equations of the system are redundant so that initial values can be determined which fulfill the whole set of initial equations. The main idea of the new approach is to reformulate

the problem of finding roots of the set of non-linear equations to an equivalent optimization problem.

Considering the general mathematical description of the initialization problem:

$$\begin{aligned} f_1(z_1, \dots, z_n) &= 0 \\ &\vdots \\ f_m(z_1, \dots, z_n) &= 0 \end{aligned} \quad (8)$$

Cases where $m \geq n$ means that more equations (m) than variables (n) are given. Every solution to (8) minimizes the problem:

$$F(z_1, \dots, z_n) = \sum_{i=1}^m f_i(z_1, \dots, z_n)^2 \rightarrow \min \quad (9)$$

On the other hand, every global minimum of (9) is a solution to (8). In order to solve (9) a number of different algorithms have been developed during the past. The algorithm can be categorized depending on the order of derivatives needed during the solution process. In the OpenModelica environment the Simplex-method of Nelder and Mead as well as the Brent's method are currently implemented, only working with the minimization function F . The OpenModelica prototype already shows reliable results for the evaluated examples.

Further improvements can be achieved as soon as the Jacobian of F with regards to the unknown is available. In that case, more advanced algorithms like the method of Fletcher-Reeves, Quasi-Newton, and/or Levenberg-Marquardt methods can be applied which would provide a speed-up in convergence. We regard this as a quality of implementation, since the described approach is working in principle already.

7 Test and Evaluation with Open-Modelica

Consider the following electrical 3-phase power system, where two generating units $vs1$ and $vs2$ are connected via a transmission line modeled by components $LR1$ and $LR2$.

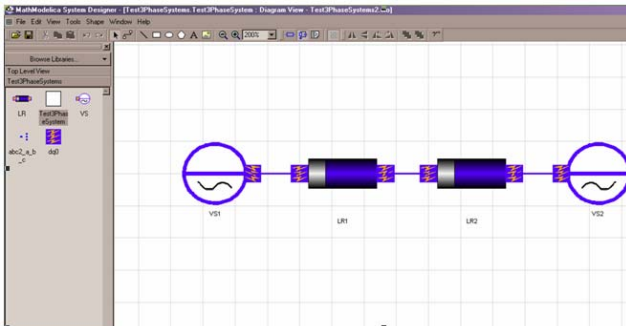


Figure 1. An electrical power system where two generating units $vs1$ and $vs2$ are connected via a transmission line.

The connectors are written in $dq0$ -coordinates implementing the potential variable u_{dq0} and the flow variable i_{dq0} . These quantities are constant in case of a nondistributed steady state, which is generally assumed during the initialization process. Introducing the Park-Transformation P the 3-phase rotating system (voltages u_{abc} and currents i_{abc}) can be calculated from the $dq0$ -representation and vice versa.

The transmission line ($LR1$ and $LR2$) is modeled by a purely inductive and resistive component, based on the Modelica Electrical Library. Since $LR1$ and $LR2$ are connected in series, giving a higher index system, index reduction has to be applied for simulation purposes.

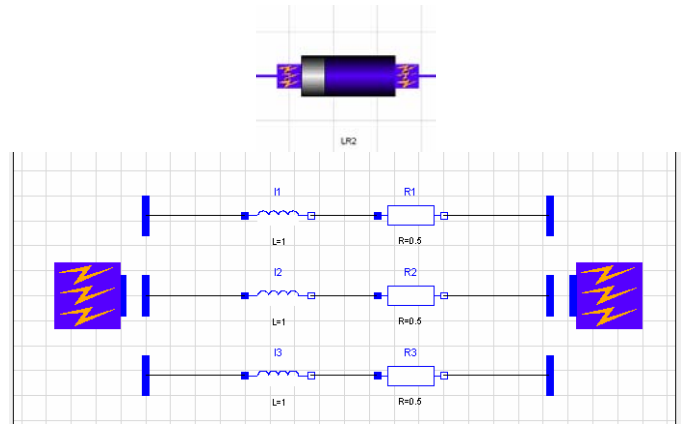


Figure 2. $LR2$ component with $dq0$ connectors.

The voltage source is described similarly using the Modelica Standard Library combined with the $dq0$ -connectors.

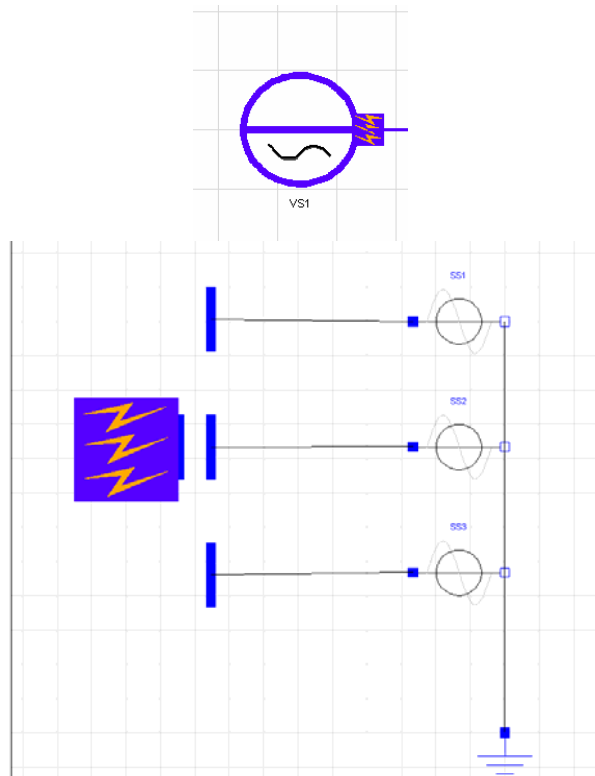


Figure 3. Voltage source.

In order to initialize the model correctly to steady state the following initial equations have been added to the local components LR1 and LR2.

```
model LR
...
equation
...
initial equation
  der(dq0_1.i_dq0)={0,0,0};
end LR;
```

Due to the higher-index of the overall system, index-reduction is applied. The system finally is determined by 3 state variables $LR1.I1.i$, $LR1.I2.i$, $LR1.I3.i$. The corresponding initial equation system has 3 equations more than number of unknowns, but these equations are redundant and could be eliminated. Due to the involvement of the Park-transformation, redundancy is not easy to detect. However, applying the concept described above correct initialization of the system is performed.

8 Implementation Status

An experimental prototype version of this method has been implemented in a special version of the OpenModelica compiler (not yet in the ordinary version), and tried on several small examples. We have worked for some time to automatically handle the example described in this paper have been delayed by a bug in the OpenModelica index reduction. The example has been

verified by partly manual efforts. However, we expect to soon fix this small remaining problem in the OpenModelica compiler.

9 Conclusions and Future work

In this paper we have presented an overview of our implementation of initializing Modelica models in the OpenModelica compiler. A new concept has been developed to describe the initial equations locally in the relevant component where the corresponding states appear, that also works for arbitrary well-posed higher-index problems. Due to the necessary index reduction some of the states get changed to dummy states that means that they will be algebraic during the simulation of the model. The corresponding initial equations are therefore redundant, but can be handled correctly by the new initialization process, if they are consistent. If not, an error/warning is issued to the user.

The implementation is however not yet complete. The current prototype just implements the concept, but the efficiency should be increased in the near future. We wish to implement calculation of the Jacobian matrix of the equation system with regards to the state variables. This gives the possibility to implement more advanced and robust numerical algorithms in order to solve the corresponding optimization (minimization) problem during initialization of the DAE.

10 Acknowledgements

This work was supported by the University of Applied Sciences in Bielefeld, by MathCore Engineering AB, by the Swedish Research Council (VR), and by SSF in the VISI-MOD project.

References

- [1] Peter Fritzson, et al. The Open Source Modelica Project. In Proceedings of The 2nd International Modelica Conference, 18-19 March, 2002. Munich, Germany See also: <http://www.ida.liu.se/projects/OpenModelica>.
- [2] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., ISBN 0-471-47163-1, Wiley-IEEE Press, 2004.
- [3] The Modelica Association. The Modelica Language Specification Version 2.2, March 2005. <http://www.modelica.org>.
- [4] The OpenModelica Users Guide, version 0.6, June 2005. www.ida.liu.se/projects/OpenModelica

- [5] The OpenModelica System Documentation, version 0.6, June 2006.
www.ida.liu.se/projects/OpenModelica
- [6] K. E. Brenan, S. L. Campbell, and L. R. Petzold, Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, Elsevier, New York, 1989.
- [7] B. Bachmann et. al. (Modelica Association): Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification. 2002.
- [8] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johansson, A. Karström: The Open Source Modelica Project. In: 2nd Modelica Conference 2002, Oberpfaffenhofen, 2002
- [9] S.-E. Mattson, H. Olson, H. Elmqvist: Dynamic Selection of States in Dymola. In: 1st Modelica Workshop 2000, Lund, Sweden, 2000
- [10] M. Otter: Objektorientierte Modellierung Physikalischer Systeme (Teil 4) – Transformationsalgorithmen. In: at Automatisierungstechnik, Oldenbourg Verlag München, 1999
- [11] M. Otter, B. Bachmann: Objektorientierte Modellierung Physikalischer Systeme (Teil 5,6) – Singuläre Systeme. In: at Automatisierungstechnik, Oldenbourg Verlag München, 1999
- [12] R. Fletcher: Practical Methods of Optimization John Wiley & Sons, 1995
- [13] J. Stoer, R. Burlisch: Einführung in die numerische Mathematik. Springer Verlag, 1994
- [14] S.E. Mattsson, G. Söderlind: Index reduction in differential-algebraic equations using dummy derivatives. SIAM Journal of Scientific and Statistical Computing, Vol. 14, 1993.
- [15] K.E. Brenan., S.L. Campbell, L.R. Petzold: Numerical Solution of Initial Value Problems in Differential Algebraic Equations. North-Holland, Amsterdam, 1989
- [16] C.C. Pantelides: The Consistent Initialization of Differential-Algebraic Systems, SIAM Journal of Scientific and Statistical Computing, 1988.
- [17] L.R. Petzold: A description of DASSL: A differential / algebraic system solver. Sandia National Laboratories, Albuquerque, 1982
- [18] H. Elmqvist: A Structured Model Language for Large Continuous Systems, PhD dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Schweden, 1978
- [19] R.E. Tarjan: Depth First Search and Linear Graph Algorithms. SIAM Journal of Comp., Nr. 1, 1972.

Calibration of Static Models using Dymola

Hans Olsson, Jonas Eborn*, Sven Erik Mattsson, Hilding Elmqvist

Dynasim *Modelon

Ideon Science Park, S-223 70 Lund, Sweden

{hans.olsson, svenerik, hilding.elmqvist}@dynasim.se, jonas.eborn@modelon.se

Abstract

A typical purpose of a static calibration is to tune static characteristics of components such pipes, valve, throttles, pumps, nonlinear resistors, frictions etc. Dymola's GUI supports setting up such a calibration without building a corresponding test rig model. The GUI allows simple redefinition of a variable to be an input. The measured data for such an input and of course also for an original input can then be specified to have a common value for all cases or to have a case dependent value read from a file.

Assume that we want to find a static relation from the variable v to w of the model and that we have measurements for v and w and all inputs of the model. First we need to decide a parameterized shape or in other words we need to come up with a function $w = f(p, v)$ where p is a parameter vector. In many cases we can use polynomials. In general it is nontrivial to come up with a good function that fits the data well. However, in this case it is possible to use the model and the measured data to back calculate w for each case. Dymola supports the setup of such a calculation is in a straightforward way very similar to the setup of the transient calibration. Having w makes the relation much more explicit and easier to visualize and inspect. Just by plotting w against v , we can get a good estimate of the chances to get a good result.

If the plot shows that the points seem to be lying on a line, the chance is much better than when the plot looks like a random scatter. Such a plot may also give us good insight in what kind of functional relation we should use. Classic pen and paper approaches as plotting in lin-log or log-log diagrams can be used to find out if exponential or potential relations could be useful.

Keywords: parameter estimation, static models, dynamic models, Modelica

1 Introduction

Physical modeling is an important tool for investigating models without the need for building them and performing experiments that may be expensive, dangerous, or delay projects.

Ideally these models should be built from first principles and easily measurable quantities. This is, however, not always the case and thus one needs to calibrate physical models to the reality.

In many cases the unknown relationship is seen as a static correlation, and the correlation is parameterized in certain ways (e.g. polynomial functions between dimensionless variables).

The goal is to determine these correlations from static measurements. The component model can then be used in the complete model, and the complete model can be validated against transient measurements.

2 Mathematical description

The actual parameter estimation uses the same numeric method as the transient calibration (a non-linear least squares method) and fits into this framework to allow analysis of the setup, e.g. to find non-identifiable subsets [2].

The mathematical background of parameter estimation (also known as data fitting [3]) is that we have a number of measured inputs, v_i , outputs w_i , and a static relation between them

$$w_i = f(p, v_i)$$

The goal is find the best set of parameters, P , and to minimize the residuals

$$r_i(p) = f(p, v_i) - w_i$$

To combine all of the residuals into one scalar to be minimized we will use the least squares formulation and minimize

$$\sum_i r_i^2(p)$$

Equivalently we can minimize the square root of this, i.e. the 2-norm of the residual.

There are two reasons for preferring to minimize the least squares problem compared to other formulations: numeric efficiency and statistical interpretation.

2.1 Statistical interpretation

The statistical interpretation of is that the parameter values minimizing the least squares residuals are the maximum likelihood parameter values, *assuming* that all errors are measurement errors in the outputs, and that these errors are normally distributed with zero mean and identical variance. Furthermore the function must be sufficiently linear [3].

These assumptions are in general not satisfied, but they can still provide some guidelines, e.g. that we should aim for outputs with “errors” of similar size.

Allowing measurement errors in both inputs and outputs lead to a more complex total least squares problem, which explains why it is not used even though it could be argued that it is more realistic in many scenarios.

An implicit assumption is that the correlation function is of the correct type. We will in the back-calculation chapter discuss how we verify that the function is of the correct type, alternatively select an appropriate type of function.

There are also well-known issues with having too many parameters, or extrapolating a correlation function far from the calibrated region. There are several statistical methods for avoid over-parameterization, including Akaike’s criterion and cross validation (and other computer intensive statistical methods).

2.2 Model errors

The errors above (measurement errors and incorrect correlation) are in one sense easy since they will (even for the optimal parameters) generate non-zero residuals, and based on this one could estimate a confidence interval for the parameters.

Modeling errors are a bit different, since we can in some cases get zero residual for a combination of incorrect model and incorrect parameters. To avoid the problem the model should be validated to ensure that it is sufficiently accurate – both for the measurement and for the actual behavior we want to study. These are normally two different validations for static calibrations, since we want to use a statically determined correlation function also for predicting transient behavior.

Related to model errors are unidentifiable sub-sets of parameters. These are handled by fixing some parameters so that remaining ones can be accurately identified. (The fixed parameters can be given reasonable values, ignored by setting them to zero or infinity, or automatically handled by static calibration). Also in this case it is important to ensure that the actual behavior does not depend on the fixed parameters. By applying the tools for finding unidentifiable sub-sets and parameters sweeps [3] to the actual behavior one can automatically verify this, or otherwise complement the static calibration with transient calibration.

2.3 Numeric solution procedure

The numeric efficiency for the solution of the non-linear least squares problem is due to the Gauss-Newton method that linearizes the residual and as one step minimizes the linear least squares problem:

$$\sum_i \left(r_i(p_0) + \frac{\partial r_i}{\partial p} \bigg|_{p=p_0} (p - p_0) \right)^2$$

This is a standard problem that can be solved using QR-factorization, and by iterating this linear procedure we get the solution for the original non-linear problem. The solution is given by the normal equations:

$$\sum_i \frac{\partial r_i}{\partial p} \bigg|_{p=p_0} \frac{\partial r_i}{\partial p} \bigg|_{p=p_0}^T (p - p_0) = - \sum_i \frac{\partial r_i}{\partial p} \bigg|_{p=p_0} r_i(p_0)$$

In practice some parameter sub-sets might be non-identifiable, or the ignored non-linear part could cause the new parameter estimate to have larger residual. We guard against both of these problems by modifying the algorithm to use the Levenberg-Marquardt method [3]:

$$\left(\nu I + \sum_i \frac{\partial r_i}{\partial p} \bigg|_{p=p_0} \frac{\partial r_i}{\partial p} \bigg|_{p=p_0}^T \right) (p - p_0) = - \sum_i \frac{\partial r_i}{\partial p} \bigg|_{p=p_0} r_i(p_0)$$

We use an automatic control of the iteration parameter, ν (which is non-negative). This problem can also be solved using the QR-factorization, and the modification gives robustness at the cost of slower convergence for problems where some parameters cannot be identified.

3 Modifying models

The previous work with parameter estimation from dynamic measurements parameter in [2] could be used. However, it is not an ideal solution since:

- Special test setup for the component model must be built, which increases the likelihood of errors.
- Either each measurement point requires one simulation (which will be slow), or a faked time is introduced leading to interpolation of the measurements (which causes slowdowns, and the model might be invalid in those regions).

The static calibration framework handles both of these, the first by modifying the model and the second by running all parameter cases in a special way. These two functionalities can be used independently of each other, and thus static calibration can be performed without modifying the model (useful if a test bench already has been set up), or modify models for other experiments.

The static calibration function handles both the case when the model is completely state-less, such as valve characteristics, and more complex steady-state cases for which either dynamics are ignored or each case is simulated until steady-state is obtained. For completely static models special code is generated, combining the model equations and the sweep over calibration cases, which gives a very efficient solution method. The more complex steady-state cases require individual simulations of each point, which gives longer solution times.

A future work is instead of selecting the component model directly select the component, which would allow the calibration function to directly update the parameters for the component. A related possibility is to automatically construct a new class extending from the base and with the updated parameters as modifiers. This allows estimation of parameters in read-only models.

3.1 Simple Example

To give a concrete example we consider a circuit with an electric resistive component. Instead of building test-rig with a source we just select to calibrate the component model Resistor. Dymola does then not default connect the two top-level pins (removing two equations), but instead asks us to transform two variables into inputs as shown in Figure 1 (to ensure that we have the same number of equations and variable):

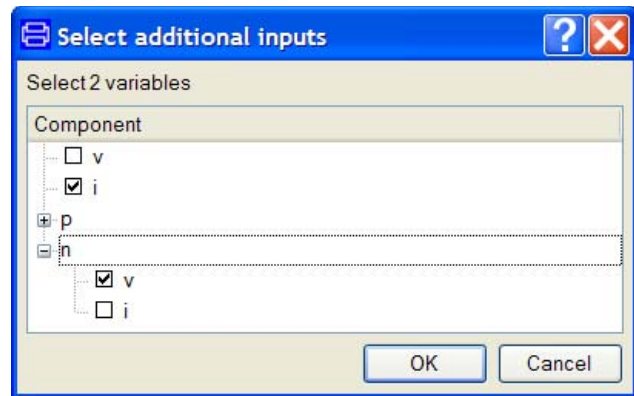


Figure 1. Selecting additional inputs, to be connected to measured data.

We can then set “n.v” to zero (arbitrary grounding) and can then use measurements for “i” and “v” to calibrate the resistance.

This corresponds to constructing a test-circuit with inputs for “i” and “n.v” and without the default connect, i.e. connecting a current source to the resistor and grounding the circuit.

The remainder of the setup is similar to the setup in [2], and the main differences are that one measurement file contains all of the cases and the possibility to provide fixed values for variables such as “n.v”, as shown in Figure 2 below.

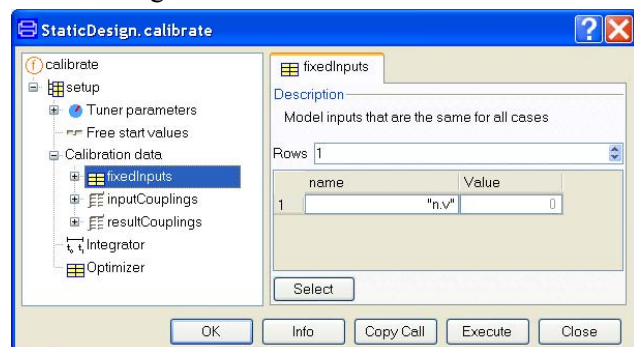


Figure 2. Assigning a fixed input signal to one potential, corresponding to grounding the resistor.

3.2 Back-calculation

Since the goal is to determine the correlations one can start without any correlation. By running the static calibration cases one gets data so that one can plot the correlations and either try to find correlations or simply verify that the selected correlation is appropriate.

For the resistor this would mean having a generic resistive load without any correlation, in that case one has to give three additional inputs: “v”, “i” and “n.v”, and can then plot voltage against current to find that Ohm’s law is valid, or that the resistive load is non-linear.

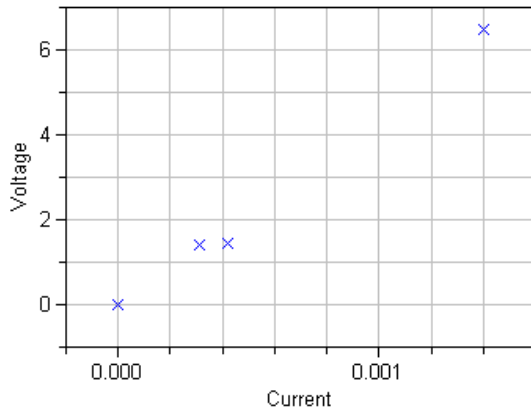


Figure 3. Simple validation of Ohm's law (errors likely due to inaccurate measurements).

The resistance can be calibrated by using the Resistor class and selecting "R" as the tuner parameter (similarly as in [2] and it is the only possible choice).

4 Application example

4.1 Calibrating steady-state compressor characteristics

One of the most important characteristics in a vapor compression cycle for refrigeration or air conditioning is the static characteristic of the compressor. This map usually describes flow rate directly as a function of compressor speed n and pressure ratio ($\pi = p_d/p_s$), or indirectly through efficiency functions, e.g. volumetric efficiency λ_{eff} and isentropic efficiency η_{is} , [4, 5]. As an example volumetric efficiency for an Ob-rist swash-plate compressor using CO_2 as refrigerant has been fitted to the functional form used in the Air-Conditioning library, the data is taken from [5] and is also available in the library. The function used for calibration is given below, for more details see 7.

$$\lambda_{\text{eff}} = \left(\pi_0 - \frac{p_d/p_s}{\pi_0 - 1} \right)^2 \left(\frac{x - x_0}{1 - x_0} \right) (a_2 n^2 x + a_1 n x + a_0)$$

where p_d and p_s are discharge and suction pressure, respectively. The relative displacement, x , was not part of the data set, so it is held fixed at $x=1$. The calibrated parameters are the maximum pressure ratio, π_0 , and the coefficients, a_i , of a first or second order polynomial. The graph below shows the result for λ_{eff} over the 30 data points.

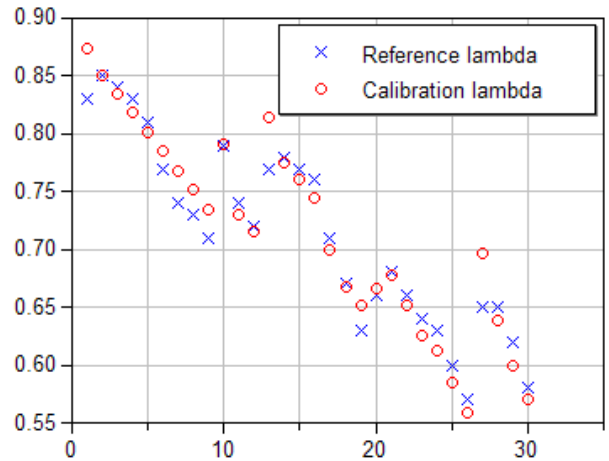


Figure 4. Calibrated volumetric efficiency over 30 data points.

Using compressor speed in rpm as independent variable gives a better view of the calibration result. The three lines in Figure 5 correspond to three different operating points at pressure ratio, $\pi = 2, 2.7$ and 4, respectively.

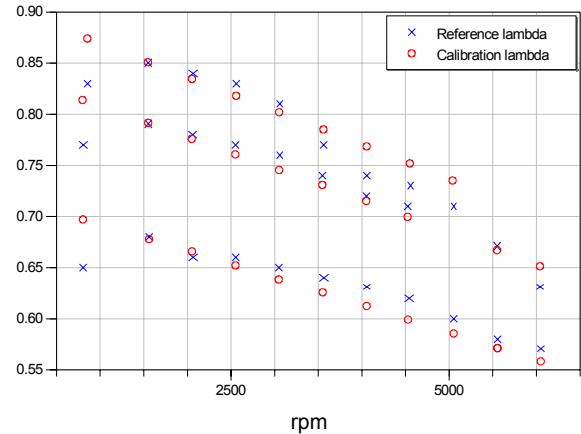


Figure 5. Volumetric efficiency as a function of rpm, showing three data sets at different pressure ratios.

5 Comparisons

Model calibration was previously available in Dymola using external optimization functions, e.g. the BasicOptimizer package using a simplex-method. This was often quite cumbersome to set up since a special model for calibration needed to be created that integrated:

- Model equations and parameter tuners
- Measurement data and residual function
- Evaluation of value function to be optimized

The case in section 4.1 is part of the example package ACWorkbench.CompressorOptimization included in the commercial AirConditioning library. Using the new Dymola calibration GUI most similar examples are set up without creating a task-specific model. For the volumetric efficiency function, a simple wrapper model needs to be created since direct optimization of functions is not yet supported in the GUI. The basic least squares optimizer has replaceable function and additional data and could be used without any model wrapper, but due to current restrictions (in Modelica and in Dymola) it uses packages with several replaceable classes that would not be convenient to directly support in the GUI.

The Modelica code for the wrapper is provided below.

```
model CalibrateVolumetricEfficiency
  "wrapper model for calibrating lambda"
  import CompEff = ThermoFluidPro.
  SubComponents.CompressorEfficiencies
  input SIunits.Frequency n "speed in Hz";
  input SIunits.Pressure pi "pressure ratio";
  output Real lambda "vol. efficiency";
  parameter CompEff.EfficiencyPars
    ce "compressor coefficients";
equation
  lambda =
    CompEff.EffectiveVolumetricEfficiency(
      n, pi*1e5, 1e5, 1, ce);
end CalibrateVolumetricEfficiency;
```

The wrapper contains definitions of the inputs and output. They are connected to measurements via the GUI, where the data values can be linearly scaled. In the case of multiple outputs the residuals can also be weighted. The tuners are found in the record class EfficiencyPars together with other coefficients that are not used. Which parameters that are active during calibration, as well as starting and min/max values are also entered in the GUI.

Also the execution of the calibration is significantly faster using the new calibration GUI. The example here is simple and calibrating the two parameters of a first order polynomial takes a few seconds. The calibration result in Figure 4 took less than five seconds and 15 model evaluations to generate. This can be compared with 78 evaluations in approximately 20 seconds using the BasicOptimizer package.

6 Complex steady-state cases

The calibration method has also been tested on a case fitting values to unknown coefficients of the

heat transfer and pressure drop correlations on the air side of an automotive condenser. This is a common task in the industry, and often requested by AirConditioning users. In the example two standard correlations from the AirConditioning library are used, both power laws with unknown multiplier and exponent:

$$\mathbf{Nu} = C_1 \times \mathbf{Re}^{C_2} \times \mathbf{Pr}^{1/3}$$

$$\Delta p = C_3 \times (\dot{m}/\dot{m}_0)^{C_4}$$

The first expression gives Nusselt number, \mathbf{Nu} , which is used in the condenser model to calculate heat transfer. The second expression gives the pressure drop as a function of mass flow. Reynolds number, \mathbf{Re} , depends on air flow and geometry. Prandtl number, \mathbf{Pr} , depends on the medium properties. The unknown coefficients are C_1 , C_2 , C_3 and C_4 .

The case was set up in a standard AirConditioning test bench model, supplying fixed outlet pressure and refrigerant subcooling as well as inflow boundary conditions on both the air and refrigerant sides of the heat exchanger. The condenser was set up with the correct geometry and the correlations corresponding to the ones showed above selected for the air side heat transfer and pressure drop. This completes the model setup, with a model corresponding to the same test bench that would be used for simulated validation or manual tuning of the condenser to measured data. The only difference is that a variable for air inlet velocity needs to be included, to connect to the corresponding measurement. Also the parameters to be calibrated were propagated to the top-level for convenience.

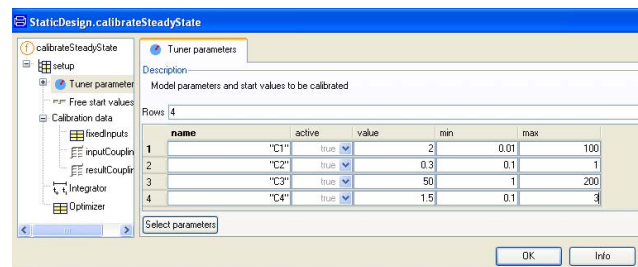


Figure 6. Dymola dialog for calibration setup of the heat exchanger case.

Using the new Dymola calibration GUI, the case was easily set up, as shown in Figure 6. On different tabs first the model is selected, then the parameters to be tuned, third the data file is supplied, and last columns with measured data connected to the corresponding inputs and variables in the model. Once setup is completed the model is translated and simulated multiple times. For a steady-state case like this each iteration requires $\# \text{data points} \times \# \text{parameters} \times 2$ simulations. Since the test bench model in question only required 1 second simulation to solve, the function converges in a few minutes. During the function

iterations, a plot shows the residual between the calculated and measured air side power and pressure drop respectively. The final residuals for power are displayed in Figure 7. The final residuals are below 100 W in each point, corresponding to less than 1% relative error. The shape of the residual suggests that the final measured value, at the highest air flow velocity, is either an outlier or that the chosen correlation is not suited for high flow speeds. The final residuals in pressure drop are around 1 Pa, and are not shown.

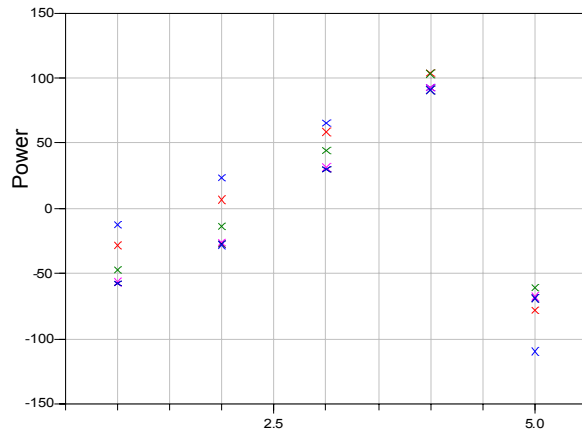


Figure 7. Power[W] residual for the last five iterations.

7 Conclusions

The above shows that Dymola is able to go from static measurements, via plots of correlations, to calibration of parameterized static correlations.

This is done in user-friendly way, without having to build additional models, and furthermore this is useful for real problems, and works more efficiently than the previous optimizer.

Furthermore the static calibration and the complex steady state cases both use the same underlying optimizer as in [2] – demonstrating its usefulness.

References

- [1] Dymola User's Manual, www.dynasim.com
- [2] Elmqvist, H., H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, M. Otter, Optimization for Design and Parameter Estimation. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany, 2005.
- [3] Fletcher, R., *Practical Methods of Optimization*, 2nd edition, John Wiley&Sons, 1987.
- [4] Tummescheit, H., J. Eborn, and K. Pröhl. AirConditioning – A Modelica Library for Dynamic Simulation of AC Systems. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany, 2005.
- [5] Försterling, S. *Vergleichende Untersuchung von CO₂-Verdichtern in Hinblick auf den Einsatz in mobilen Anwendungen*, Ph.D.-thesis, TU Braunschweig, Germany, 2004.

Automatic Fixed-point Code Generation for Modelica using Dymola

Ulf Nordström^{1,2} José Díaz López¹ Hilding Elmqvist¹

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

²Lund Institute of Technology, Lund, Sweden

Abstract

This paper describes a Modelica package for fixed-point arithmetics and automatic fixed point code generation for embedded systems and FPGA applications. Using Dymola [1] to investigate the dynamic behavior of the original model a fixed point representation is automatically generated. The model can then be simulated, using fixed point arithmetics to verify the fixed-point representation. Finally, code is generated for the desired target. Either integer C code for embedded systems or Mitrion-C code [2] for automatic VHDL code generation for FPGA targets.

Keywords: *fixed-point arithmetics; automatic code generation; embedded system; DSP; FPGA; Mitrion-C*

1 Introduction

Hardware-In-the-Loop simulations are widely used nowadays for design and testing of control systems in industry. Typical devices for such HILs are Digital Signal Processors (DSP) and Field Programmable Gate Arrays (FPGA). Typically, the development of algorithms or models is done in high level languages, not directly related to the target hardware. This is advantageous since the model keeps independent. But, specific code generation for the target platform has to be done.

Another important aspect is the following. During the development phase, floating point arithmetics is used for computations of algorithms and models. In many cases, the tolerances, characteristics of the system and performance of the target platform do not justify the use of such demanding floating point calculations. Furthermore, sometimes they are even an obstacle to HILs, since the computations are slower than required.

The possibility we explore in this paper is *fixed-point arithmetics* for modeling. We describe in this paper an

aid platform for automatic code generation from Modelica models using fixed-point arithmetics.

In a first step, Dymola generates a corresponding Modelica model using fixed-point arithmetics. This model is intended for bit configuration testing and result assessment.

In a second step, the platform generates code in two variants

1. C code using exclusively integer data types for all variables with corresponding binary operation implementation. This type of code is intended mainly for DSP applications.
2. Mitrion-C code, using also integer data types, but with different word lengths. This code is mainly intended for FPGA applications.

As a scenario example, let us consider the task of developing an electrical control unit (ECU) for speed control of a simple vehicle drive train using DSP. To test different control strategies, a model of the closed loop system is implemented in Modelica. The model is depicted in Fig. 1.

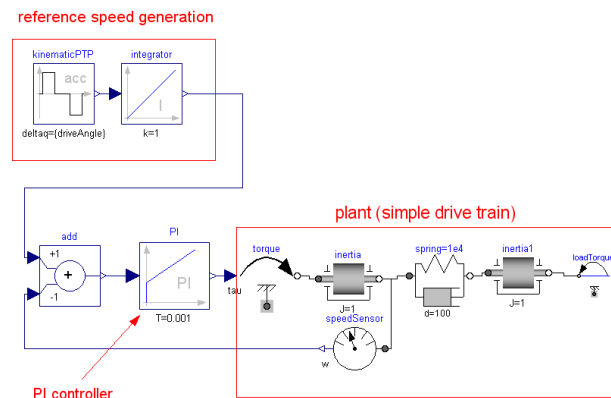


Figure 1: Simple drive train with PI speed controller in closed loop

After some experimentation we find appropriate parameter settings and verify the functionality of the controller. Our intention now is to realise the controller using an DSP.

Manually transforming the equations for the PI controller to a DSP program is a tedious and error prone task. In the following, we will see how the ModelicaFixedPoint package can be used to perform this task automatically.

Furthermore, the package provides error propagation analysis and fixed-point implementation of usual mathematical functions based on Newton-Rapson or table interpolation.

2 ModelicaFixedPoint package

The structure of ModelicaFixedPoint is presented in Fig. 2.

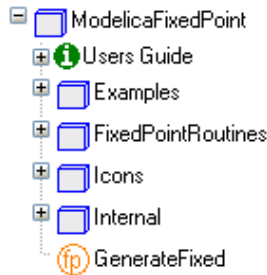


Figure 2: Library structure.

The main function of the library, `GenerateFixed`, is used for generation of the fixed-point representation and code generation. `GenerateFixed` preforms the 4 main steps of the conversion

1. Fixed-point translation
2. Range analysis
3. Precision analysis
4. Code generation

The internal symbolic engine of Dymola preforms symbolic manipulation of the original model and outputs a fixed-point converted model with all arithmetic operations replaced by function calls. Some examples of Modelica types and operations before and after conversion are shown in listings 1 and 2.

Listing 1: Modelica types and fixedpoint types

```
// before conversion
Integer x;
Real y;
```

```
Boolean z;

// after conversion
FixedPoint.Integer x;
FixedPoint.Real y;
FixedPoint.Boolean z;
```

The arithmetic operations are replaced by function calls and a unique identifier is inserted for every function call. *Type* can here be either Real or Integer depending on the types of the arguments, as example we present the basic operations.

Listing 2: Basic operations

```
// before conversion
a=u+v;
b=u-v;
c=u*v;
d=u/v;
```

```
// after conversion
a=Add.Type.Type(u, v, opId1);
b=Subtract.Type.Type(u, v, opId2);
c=Multiply.Type.Type(u, v, opId3);
d=Divide.Type.Type(u, v, opId4);
```

Consider now the PI controller of the ECU example. This controller is implemented in Modelica with the following equations

$$\dot{x} = \frac{u}{T}$$

$$y = k(x + u)$$

where x is the controller state variable, u is the input, k is the proportional gain and $1/T$ is the integral constant. Those equations converted to fixed point with ModelicaFixedPoint are presented in listing 3. The operator `FixedPoint2Derivative` is introduced to perform time integration in Dymola, during assesment of the fixed-point model.

Listing 3: Typical equations

```
der(x_aux)=FixedPoint2Derivative(Divide.Real.Real(u, T, op1));
y=Multiply.Real.Real(k, Add.Real.Real(x, u, op2), op3);
```

Range analysis

The proposed method uses a simulation based approach for determining ranges of variables and intermediate results in the equations. Simulating the original model and logging minimum and maximum values of all variables is important. This allows equation traversing and thereby compute accurate ranges for all intermediate results.

This method is more time consuming than methods like interval arithmetics [6] and affine arithmetics [4], but the resulting intervals are better.

ModelicaFixedPoint considers that every variable consists of *two parts*: an *integer* part and a *fractional* part. Both integer and fractional parts have a bit length

that sum up to the total *word length* used for the variable. The integer word length is denoted here by IW_L , the fractional word length by FW_L and the total word length by W_L .

The range information is used to assign the needed IW_L to all variables and operations.

Precision analysis

Precision analysis is done in two different ways depending on the target platform (DSP or FPGA). The two main parameters in `GenerateFixed` are `wordLength` (target platform word length) and `RTOL` (relative tolerance required for conversion). The function dialog box is shown in Fig. 3.

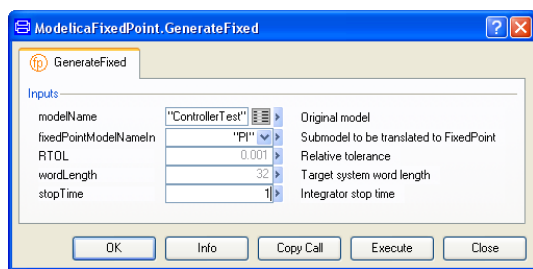


Figure 3: Dialog box.

Specifying `wordLength` indicates that integer C code is to be generated for a DSP target. The specified word length then sets a unique word length for the entire system and the FW_L is set to maximize precision given by `RTOL`.

For FPGA targets the task becomes more complicated due to the ability to use different W_L in the system. Our approach is to use a backward error propagation scheme based on the error propagation analysis in section 5 to determine all FW_L and shift operations from a user specified tolerance `RTOL` at the output of the system.

The package `FixedPointRoutines` is also included. This package contains fixed-point arithmetic functions and records for simulation of fixed-point Modelica code. The structure is shown Fig. 4.

3 Fixed-point scenario

Integer arithmetic operations execute much faster than their corresponding floating-point operations because of their simplicity. In the case of FPGA, silicon surface area and power consumption are also significantly reduced using integer arithmetics. On the other hand, DSP devices come normally with simple arithmetic

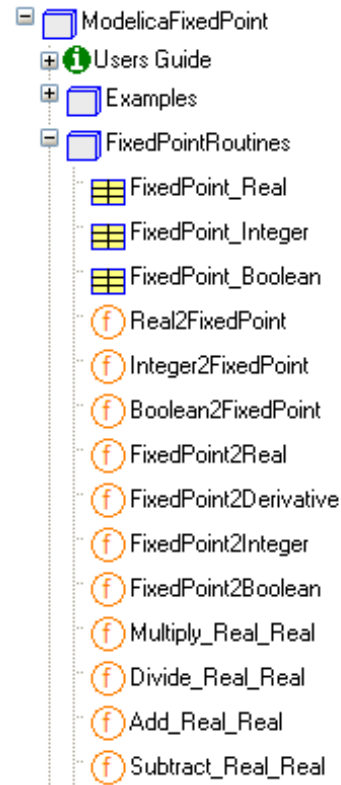


Figure 4: FixedPointRoutines.

logic units, missing completely hardware implementation of floating point arithmetics.

The achievable precision using integer arithmetics is closely related to the architectural word length of the target platform. In the case of DSP, the integer data word size, typically 16, 24 or 32 bits, limits the possible precision. For FPGA, there is the possibility of adapted word length for each operation. However, this might exceed practical limitations. A tradeoff must be considered, minimizing the number of bits used while not violating constraints on precision. Another interesting aspect of the FPGA is parallelism. The programmable structure of the FPGA allows exploiting independent computations, by implementing those in parallel structures.

The target language for FPGA is Mittrion-C. This language is a dataflow language with a syntax resembling C, developed by Mittrionics AB in Lund. The generated Mittrion-C code is compiled into a configuration of the Mittrion Virtual Processor. The Mittrion Virtual Processor is a fine grain massively parallel softcore processor which can be downloaded and ran on a variety of FPGA's, see [2] for details.

4 Fixed-point Arithmetics

From a hardware point of view, fixed-point arithmetics is essentially integer arithmetics with bit shifting. Using integers to represent non-integer values is done by considering an imaginary binary point as follows. Consider the binary representation of an integer in Natural Binary Code (NBC).

$$(b_n, b_{n-1}, \dots, b_2, b_1, b_0) = \sum_{j=0}^n b_j \cdot 2^j, \forall b \in \{0, 1\}. \quad (1)$$

Now, using the same set of bits to represent a non-integer value can be done by placing a binary point between $i - 1$ and i . Thus

$$(b_n, \dots, b_{i+1}, b_i, b_{i-1}, \dots, b_0) = \sum_{j=0}^n b_j \cdot 2^{j-i}. \quad (2)$$

The fixed-point representation in (2) with the binary point between $i - 1$ and i is said to have i bits of precision. By q we denote the value in NBC, i.e. the value seen by the arithmetic unit. The smallest representable number is the so-called the resolution of the representation and is equal to 2^{-i} . We define integer word length, $IW_L = (n + 1) - i$, and fractional word length, $FW_L = i$. The word length, that is, the total number of bits used, is denoted by W_L and is equal to $n + 1$.

Converting a floating-point number y to fixed-point representation q is done by

$$q = \lfloor 2^i \cdot y \rfloor, \quad i, q \in \mathbb{Z}, \quad y \in \mathbb{R}, \quad (3)$$

where $\lfloor \cdot \rfloor$ denotes rounding towards floor.

The basic operations on fixed-point numbers; addition, subtraction, multiplication and division, are implemented using ordinary integer operations and bit shifting. The bit shifts (left shift and right shift) of a fixed-point number q are

$$(q \ll i) = q \cdot 2^i \quad (4)$$

and

$$(q \gg i) = q \cdot 2^{-i}. \quad (5)$$

Bit shifting is used extensively to align binary points and to rescale variables. The shift operators are used to rescale both the inputs of an operation and the output. Consider a binary operation op on two fixed-point variables q_1 and q_2 , called operands. The implementation of such operation is

$$\text{Op}(q_1, q_2, s) = ((q_1 \ll s_1) \text{op} (q_2 \ll s_2)) \ll s_3 \quad (6)$$

where $\text{op} \in \{+, -, *, /\}$, or equivalently

$$\text{Op}(q_1, q_2, s) = ((q_1 \cdot 2^{s_1}) \text{op} (q_2 \cdot 2^{s_2})) \cdot 2^{s_3}. \quad (7)$$

Clearly, associating a 3-tuple of shifts $s = \{s_1, s_2, s_3\}$ with an operation we define the syntax of the basic operations. For readability reasons we will use the symbol for left shift, \ll , followed by s_i to denote shifts. When $s_i > 0$, the shift is left and when $s_i < 0$ the shift is right.

For addition and subtraction the binary point must be aligned before the operation. There will be a loss of precision in the result when right shifting is used. Thus, it is preferable to use left shifts when possible. However, right shifts may sometimes be needed in order to avoid *overflow*. Adding or subtracting numbers with very large difference in magnitude does not mean any problem. The smaller one will naturally be numerically insignificant compared to the larger one.

Multiplication and division are more difficult to implement. Multiplying two fixed-point variables each having W_L number of bits will generate a result having $2W_L$ number of bits. This is likely to cause an overflow.

Using the shift operators on the variables prior to the fixed-point multiplication, one could shift the operands so that no overflow can occur. For the situation above, the secure shiftings correspond to right shifting each of the operands by $W_L/2$ bits prior to the operation. This reduces of course the final resolution of the result.

Division has a similar problem. Dividing two variables with FW_L fractional bits generates a result having no fractional bits. Here we loose resolution unnecessarily. One solution is to right shift the denominator prior to the division, keeping as much precision as possible.

For operations other than the basic binary ones, there are often no straight forward implementation. Conditional operations such as $\{<, \leq, >, \geq, ==, \neq\}$ are an exception. These operations on fixed-point numbers are the same as their floating-point counterparts although the binary points of the operands must be aligned before evaluation. This alinement of binary points can reduce resolution and therefore cause the wrong conditional branch to be evaluated.

For other functions we use linear interpolation in tables or Newton-Raphson iterations to evaluate the result, as mentioned earlier. With information on the range of the variable and the result, tables covering the entire range with appropriate resolution are generated. Development of interpolation tables and Newton-Raphson iterations is currently ongoing.

In order to use fixed-point arithmetics efficiently one need to find appropriate shifts, and W_L in case of FPGA, for all variables and operations. The automatic *floating-point to fixed-point conversion problem* can then be summarized as finding IW_L and FW_L for all variables and respective appropriate shifts for all operations such that no overflow occurs, quantization errors are kept low and the total W_L is kept as low as possible.

5 Error propagation

Error propagation analysis has to be done to solve the fixed-point conversion problem, and assign FW_L to variables and operators. For further theory details, see [3].

A throughout error analysis would require *a priori* knowledge of all shifts and FW_L . For simplicity, we assume that all intermediate results can be stored, eliminating the the risk of overflow. Clearly and in practice, this assumption is relevant and valid for Mitron-C code only.

For addition and subtraction the function implementation is eq. (7). Multiplication and division are somewhat different.

For multiplication, the possibility of storing the intermediate result eliminates the shifting of the operands before the operation. Instead, the result has to be shifted to the appropriate FW_L .

For division, the assumption avoids shifting the denominator. Hence, only the numerator and the result are shifted to the appropriate FW_L at the output.

Conversion

The conversion to fixed-point causes an error, the so called *conversion error* or *quantization error*, denoted by δ . We have

$$|\delta| = |y - \tilde{y}|, \quad \delta \in \mathbb{R}. \quad (8)$$

where y is the floating-point value before conversion and \tilde{y} is the recovered floating-point value after conversion. From [3], the conversion error is bounded by

$$|\delta| < 2^{-i}. \quad (9)$$

Defining

$$\Delta = \sup |\delta| = \sup |y - \tilde{y}|, \quad (10)$$

the results of the error propagation analysis will be presented.

Addition and subtraction

Addition and subtraction have the same error propagation properties. Considering the absolute error for addition we have

$$\Delta_R = \Delta_1 + \Delta_2 = 2^{-i_1} + 2^{-o} + 2^{-i_2} + 2^{-o}. \quad (11)$$

where i is the resolution of the operand and $o = FW_L$ is the resolution of the result.

Multiplication

Considering the relative error for multiplication, we have

$$\left| \frac{\Delta_R}{R} \right| = \sup \left| \frac{\delta_R}{R} \right| = \left| \frac{2^{-i_1}}{y_1} \right| + \left| \frac{2^{-i_2}}{y_2} \right| + \left| \frac{2^{-o}}{y_1 y_2} \right| \quad (12)$$

where y_i is the largest value taken by each variable respectively.

Division

Again considering the relative error we have

$$\left| \frac{\Delta_R}{R} \right| = \left| \frac{2^{-i_1}}{y_1} \right| + \left| \frac{2^{-i_2}}{y_2} \right| + \left| \frac{2^{-i_2+o}}{y_1} \right|. \quad (13)$$

The error propagation is different from the multiplication.

6 Code generation

The information gained in the fixed-point conversion and analysis is used to generate code for different target platforms. In this paper we mainly focus on generating integer C code typically used in DSP's. But, also Mitron-C code for FPGA's is possible with appropriate error propagation analysis.

For easy interaction with standard generic integration routines, all variables are categorized and mapped to a set of vectors, as in table 1. All right-hand-side equations are gathered in one function, called *rhs-function*. This encapsulation allows code portability.

This encapsulation and code generation is used in cases where only a subsystem is intended for fixed-point representation. Then we substitute that particular part of the system with a function call and the rest of the system is treated as usual.

To avoid unnecessary computations, constants are shifted according to the fixed-point representation and evaluated during the code generation.

Generation of integer C code

Currently, the generated C code supports a format for easy interaction with Dymola to make it easy to validate the functionality of the generated code. Different

x	state variables
x_dot	derivatives of state variables
w	auxiliary variables
u	input variables
y	output variables
p	parameters

Table 1: Table of categorized variables

DSP targets could require different structures, mainly of the function call, i.e. the body of the function would remain the same.

The syntax of the Modelica interface to the rhs function is in listing 4.

Listing 4: Modelica Interface

```
// Modelica interface
function Name
  input Integer n;
  input Integer t;
  input Integer x[:];
  input Integer x_dot[:];
  input Integer w[:];
  input Integer u[:];
  input Integer y[:];
  input Integer p[:];
  input Integer idemand;
  output out_w[size(w,1)];
  output out_x_dot[size(x_dot,1)];
  output out_y[size(y,1)];
  external "C";
  annotation (Include="#include <Name.c>");
end Name;
```

and the function itself in listing 5.

Listing 5: C interface

```
// C interface
int Name(int n, int t, int *x, int size_x,
         int *x_dot, int size_x_dot, int *w,
         int size_w, int *u, int size_u,
         int *y, int size_y, int *p, int size_p,
         int idemand, int *out_w, int size_out_w,
         int *out_x_dot, int size_out_x_dot,
         int *out_y, int size_out_y){

  /* BODY OF FUNCTION */
}
```

The main body of the function is the section containing the equations for computing the auxiliaries, derivatives and outputs of the system. The basic operations, addition, subtraction, multiplication and division, uses basic integer operations and the appropriate shifts are inlined in the code according to the syntax in (6). The integer C code of the equations in listing 3 can be seen below in listing 6.

Listing 6: Integer operations in C

```
/* Compute derivatives */
x_dot[0] = ((u[0]) / (p[1] >> 15) << 10);

/* Compute outputs */
y[0] = ((p[0] >> 16) * (((x[0]) + (u[0] >> 1)) >> 15) << 1);
```

Conditional operators and the *IfThenElse*-statement are also inlined with appropriate shifts. Essential for the conditional operators are that the binary points are

aligned before evaluation. Hence, also here, shift operations are inlined in the code. An example can be seen in listing 7.

Listing 7: Conditional operations in C

```
// Computing "z = if x<y then x-y else y-x;"
w[0] = (((((x[0]>>6)<(x[1]))==1)?((x[0]>>6)-
(x[1]))):(((x[1])-(x[0]>>6))));
```

A C library for special fixed-point functions is under development for functions such as *min*, *max* and *abs*. Also for these functions, except for *abs*, the binary point must be aligned prior to the evaluation. This can be solved by having the appropriate shifts inlined in the function call. For other functions, such as *trigonometric* functions and *square root*, code for linear interpolation tables or Newton-Raphson iterations schemes will be generated. This is still in the development phase although successful implementation of linear interpolation using only integer arithmetics have been implemented using Modelica.

Generation of Mittrion-C code

Although the syntax of Mittrion-C resembles that of C, the generated code is quite different. This is mainly due to the possibility of having a mixture of different word lengths in the system. This allows us to allocate resources (word length of the data path) where they are needed the most.

The result of this is that it is no longer possible to inline any operations or shifts. Instead a unique function is generated for every operation. All shifts and word length declarations are hard coded in these functions.

7 Examples

7.1 Speed control with PI controller

Using the library presented in this paper lets us automatically generate a fixed-point model of a PI controller using various word lengths. Simulating the system using the fixed-point PI controller we can verify the functionality of the controller and also get hints regarding the word length needed to fulfill the specifications.

After some simulations, depicted in Fig. 5, we conclude that a word length of 8 bits is enough for our application. This results in a smaller/cheaper DSP for the ECU instead of the one originally intended. Also, integer C code for the controller is automatically generated.

For Modelica users this is a familiar way of working, implementing systems and algorithms at high abstraction levels by drag and drop of predefined components

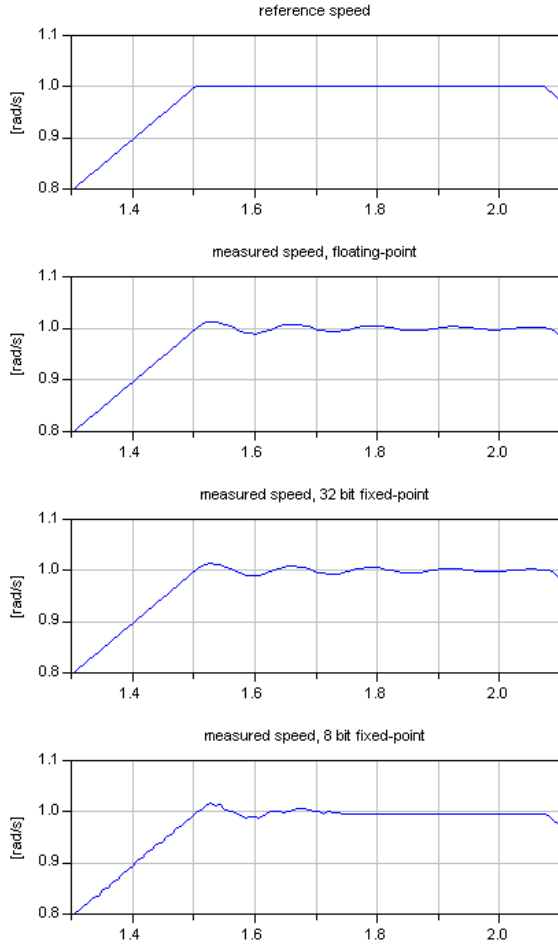


Figure 5: Plot of reference speed and measured speed using PI controllers.

in a diagram or manually typing Modelica code. With this library we aim to keep this way of working while extending it by automatically generating fixed-point code for external computational devices. The following example is the opposite situation: plant realisation with fixed point arithmetics.

7.2 ServoMotor with FixedPoint

This example explores the plant realisation using fixed point arithmetics. The situation is a simple electric drive used to place a inertia at a given angle. The simple model of electric drive is depicted in Fig. 6. The model consists of an ideal EMF device with shaft inertia and electric inductance and resistance.

We provide the model with the load and a gearbox for better precision on the load. The system is encapsulated in a model with inputs and output as depicted in Fig. 7.

Finally, we choose a PID regulator to control the angle

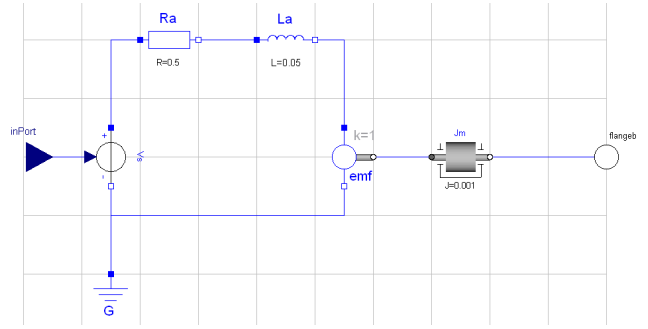


Figure 6: Electric Drive Model

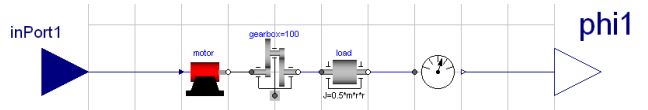


Figure 7: Electric Drive and load

of the load and set it to the reference. The final model is depicted in Fig. 8.

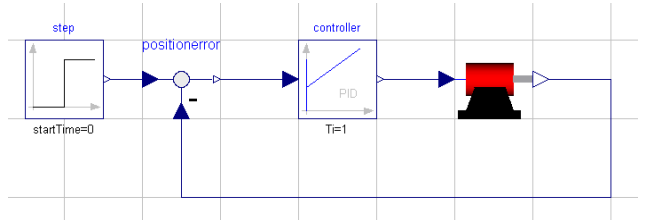


Figure 8: PID regulator to position the inertia to build the servo drive

The question now is if there is a cheap and fast fixed-point representation of the system, good enough to use with the PID regulator in hardware. After Fixed-point trasnlation, we get with the tool the system in Fig. 9. The first attempt is done analysing `servoMotorWithLoad` model within the time interval $[0, 1]$ and for 8, 16 and 24 bits. The result of the simulations in Dymola are depicted in Fig. 10. We observe that even though we have more precision, the steady state is never really reached. The solution becomes oscillatory.

The first though may be that the number of bits of the fixed point representation is not high enough to make the system reach steady state. This is not the case. What is happening is that the precision analysis is tightly adjusting the fixed-point representation to the time interval and span interval of every variable.

It is possible to resemble the correct dynamics of the system with just 8 bits. The result is depicted in Fig. 11. The only difference is that we performed the precision analysis in the time interval $[0, 100]$ instead.

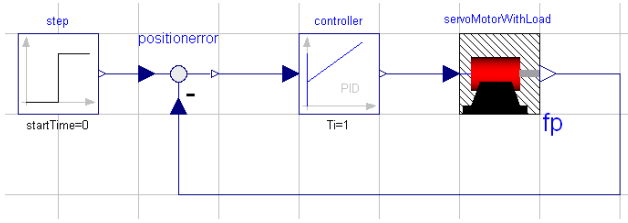


Figure 9: Fixed Point version of the motor with load.

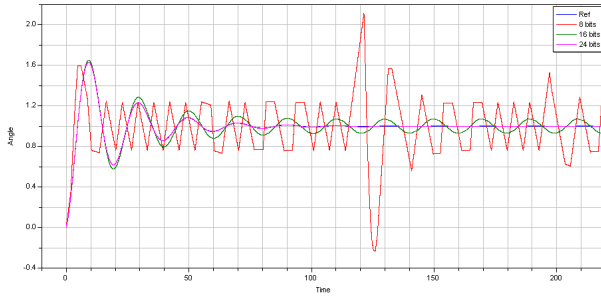


Figure 10: Simulation result with model analysed in time interval [0, 1]

We can see the C code generated for the example also in listing 8. Notice the implementation of the fixed-point operations using shifts. This implementation can be also encapsulated to generate code to link with external fixed-point libraries.

Listing 8: C function generated

```

/* BODY OF THE FUNCTION */
/* Compute Auxiliary variables */
if (idemand == 3) {
    w[0] = 0;
    w[1] = 0;
    w[2] = 0;
    w[3] = 0;
    w[4] = 0;
    w[5] = 0;
    w[6] = 0;
    w[7] = (((32 > 4) * (((((r >> 3) >> 4) * (r >> 3)) >> 4) << 2);
    w[8] = ((motor_emf_k >> 3) * (x[1] >> 4) << 1);
    w[9] = ((gearbox_ratio >> 3) * (derload_w_aux >> 4) << 2);
    w[10] = ((w[8] - (((motor_Jm_J >> 3) * (w[9] >> 4)) << 1);
    w[11] = (-(((gearbox_ratio >> 4) * (w[10] >> 3))) << 1);
    w[12] = ((motor_Ra_R >> 3) * (x[1] >> 4));
    w[13] = ((u[0] - w[12] >> 5));
    w[14] = ((gearbox_ratio >> 4) * (x[0] >> 3) << 1);
    w[15] = ((motor_emf_k >> 3) * (w[14] >> 4) << 1);
    w[16] = ((w[13] - w[15] >> 2);
    w[17] = ((gearbox_ratio >> 4) * (x[2] >> 3) << 1);
    w[18] = (-(((w[10] >> 7) + (w[11] >> 3)) << 1);
    w[19] = 0;
}

/* Compute derivatives */
if (idemand == 2) {
    x_dot[0] = (((((gearbox_ratio >> 4) * (w[8] >> 3)) << 1) +
    (((((motor_Jm_J >> 3) * (gearbox_ratio >> 4) >> 3) << 1) *
    (gearbox_ratio >> 4) << 4) >> 1) << 7);
    x_dot[1] = ((w[16] - w[15] >> 2);
    x_dot[2] = x[0];
}

/* Compute outputs */
if (idemand == 1) {
    y[0] = x[2];
}

```

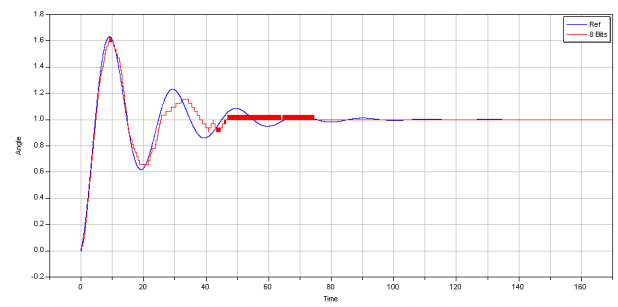


Figure 11: Simulation result with model analysed in time interval [0, 100]

8 Summary

We have presented a tool that enables a developer to use Modelica models for code generation for an embedded system or FPGA with a minimum of manual interaction.

References

- [1] Dymola, Dynasim AB, www.dynasim.com
- [2] Mittrion-C, Mittrionics AB, www.mittrion.com
- [3] Ulf Nordström. **To be published.** Automatic Fixed Point Code Generation in Modelica using Dymola. Lund, Sweden: Master's thesis, Department of Automatic control, Lund Institute of Technology, 2006.
- [4] Claire F.Fang, Rob A Rutenbar, Tsuhan Chen. Fast, Accurate Static Analysis for fixed-point Finite-Precision Effects in DSP Designs. Pittsburgh, USA, Department of Electrical and Computer Engineering, Carnegie Mellon University.
- [5] Float-to-Fixed Conversion Tool, www.float-to-fixed.com
- [6] R. E. Moore. Interval Analysis. Prentice-Hall, 1966.
- [7] The Mathworks, Simulink fixed-point, 2005.

Session 6b

Thermodynamic Systems and Applications

The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks

Francesco Casella¹, Martin Otter², Katrin Proelss³, Christoph Richter⁴, Hubertus Tummescheit⁵

¹Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italy

²German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany

³Technical University Hamburg-Harburg, Institute for Technical Thermodynamics, Germany

⁴Technical University Braunschweig, Institute for Thermodynamics, Germany

⁵Modelon AB, Ideon Science Park, Lund, Sweden

Abstract

The new library Modelica_Fluid is a free Modelica package providing components describing zero- and one-dimensional thermo-fluid components, which can be connected in arbitrary networks. The purpose of the library is to provide standard interfaces for thermo-fluid components, demonstrate how to build such models, and include a growing set of models of common use. The component equations are decoupled from the equations to compute the fluid properties, which are provided by the Modelica.Media library through standard interfaces; incompressible and compressible fluids, single or multiple substances, one- and multiple-phase fluids can be used, where appropriate. Newly introduced features of the Modelica.Media library are briefly reviewed. After extensive testing by interested users, the library will be included in the Modelica standard library as Modelica.Fluid.

1 Introduction

The Modelica_Fluid library provides basic interfaces and components to model thermo-hydraulic systems with zero-dimensional and one-dimensional components. It is not the intention that this library covers all possible application cases, because the modelling assumptions can vary widely. Instead, the goal of the Modelica_Fluid library is to **demonstrate how to implement** components of thermo-hydraulic processes in Modelica, provide **standard connectors** which fit for a wide range of applications, and provide a **reasonable set of components**, which can be used as they are, or can be modified to suit specific user needs. For special applications it is possible to implement libraries with simpler media and components, e.g., the Modelica.Thermal.FluidHeatFlow

library [4]. Other domains, such as gas dynamics, would require a more sophisticated setup.

The basic concepts of the Modelica_Fluid library, in particular the fluid connectors and the use of replaceable medium models, were laid out in [2]. Since then, the library design has been refined and tested by several people belonging to the Modelica Association. The structure of the library is now stable – contributions are welcome to increase the number of provided components. The goal is that this library becomes part of the Modelica standard library, after it has been tested by end users on a significant number of different applications and is improved based on the feedback.

A screen shot of the library is shown on the right side. The Examples package contains models that demonstrate various features of the library, as well as some system models, such as a drum boiler [6] and an experimental batch plant [5] model.

A typical (small) example is shown in Figure 1 below: It shows a system where water is pumped from a source by 4 pumps in parallel (fitted with check valves), through a pipe whose outlet is 50 m higher than the source, into a reservoir placed on an 18-m high tower. The users are represented by an equivalent valve, connected to the reservoir. The water controller is a simple on-off controller, acting on the gauge pressure measured at the base of the tower; the output of the controller is the rotational speed of the pumps. A typical simulation is over 2000 s. The pump turns on and off to keep the reservoir level around 2.5 meters, which means



20.5 meters higher than the base of the tower, corresponding to a gauge pressure of 2 bar.

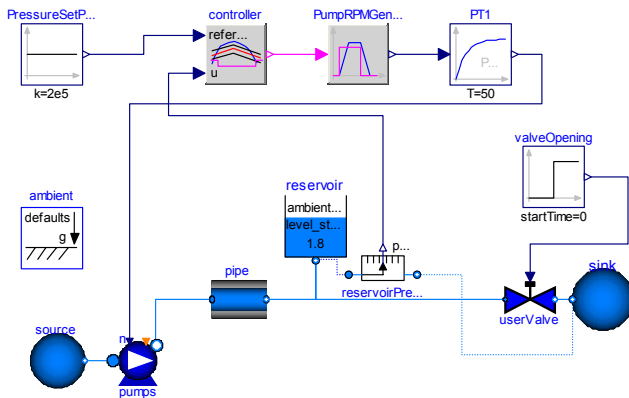


Figure 1: Pumping system for drinking water

2 General design principles

Compared to other engineering modeling fields, such as electrical systems or multibody systems, the task of providing a “standard” library for thermo-fluid systems is much more difficult, due to the much greater variety of modeling assumptions that can be made, depending on the specific application needs. The Modelica_Fluid library tries to strike a balance, providing a sufficiently general framework, which covers a wide range of applications without adding too much overhead to the simplest cases.

The scope of the library includes zero- and one-dimensional models of thermo-hydraulic components, i.e. objects where the flow of one or more fluids must be described, and energy transfer and storage phenomena play a significant role.

The thermo-fluid connectors are designed in order to ensure that mass and energy balances are fulfilled at the connection point, even in presence of flow direction reversal. On the other hand, the momentum balance is fulfilled exactly only when two aligned objects with equal flange diameters are connected; in other cases, the momentum balance at the connecting points is approximated. The exact treatment of momentum balances at the interfaces in those cases would add a significant complexity and overhead to the library, which is unnecessary in most technical thermodynamics applications, where gas dynamics phenomena (wave propagation, high Mach numbers) do not play a significant role. Gas dynamics systems are then outside the scope of the Modelica_Fluid library.

The library models can describe two-phase flows, as long as the flow is homogeneous, i.e., both phases have the same velocity.

The medium models, i.e., the equations to compute all the fluid properties from the independent thermodynamic state variables, are included in the component models as replaceable instances of objects from the Modelica.Media standard library. This allows to use the same component model with different fluids (or with different models of the same fluid) by just replacing the medium model.

3 Fluid Connectors

In this section the design of the fluid connectors is explained. A major design goal was that components can be arbitrarily connected and that the important balance equations are automatically fulfilled when two or more components are connected together at one point as shown in the next figure:

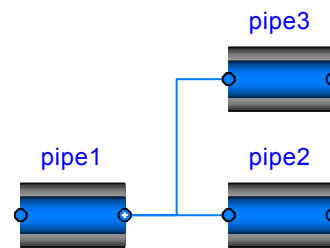


Figure 2: Connected pipes fulfilling the ideal mixing condition at the connection point.

As will be explained below, in such a case the balance equations define ideal mixing, i.e., the connection point has the mixing temperature if the fluids from the three components would be ideally mixed in an infinitely small time period. If more realistic modeling is desired that takes into account dissipation and other mixing losses, an explicit model has to be used in the connection point, e.g., from the Modelica_Fluid.Junctions library. An example is given in the next figure:

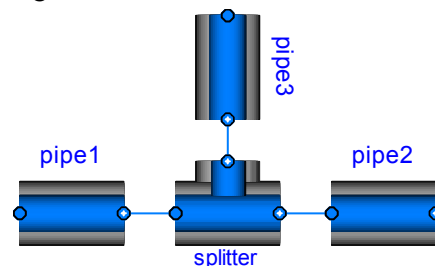


Figure 3: Connected pipes with a splitter junction where the losses are described in the junction model. For a single substance medium, the connector definition in Modelica_Fluid.Interfaces.FluidPort reduces to

```

connector FluidPort
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;
  Medium.AbsolutePressure      p;
  flow Medium.MasFlowRate      m_flow;
  Medium.SpecificEnthalpy      h;
  flow Medium.EnthalpyFlowRate H_flow
end FluidPort;

```

The first statement defines the medium flowing through the connector. In package Medium, medium specific types such as "Medium.AbsolutePressure" are defined that contain medium specific values for the min, max and nominal attributes. Furthermore, Medium.MassFlowRate is defined as:

```

type MassFlowRate =
  Modelica.SIunits.MassFlowRate(
    quantity="MassFlowRate." +
      mediumName, ...);

```

A Modelica translator will check that the quantity and unit attributes of connected interfaces are identical. Therefore, an error occurs, if connected FluidPorts do not have a medium with the same medium name.

The variables in the connector have the following meaning: p is the absolute pressure at the connection point, m_flow is the mass flow rate from the connection point in to the component, h is the specific mixing enthalpy in the connection point and H_flow is the enthalpy flow rate from the connection point into the component.

3.1 Balance Equations at Connection Points

Assume that 3 FluidPorts port1, port2, port3, are connected together. Since, m_flow and H_flow are flow variables, a Modelica translator will generate the following connection equations:

```

port1.p = port2.p = port3.p
port1.h = port2.h = port3.h
0 = port1.m_flow + port2.m_flow +
  port3.m_flow
0 = port1.H_flow + port2.H_flow +
  port3.H_flow

```

These are exactly the equations that state ideal mixing for an infinitesimal small control volume in the connection point: The intensive quantities at the ports are identical and the mass balance as well as the energy balance is fulfilled (note that no mass or energy is stored in the infinitesimal volume). The momentum balance is not taken into account, and therefore a connection without an explicit junction model is only valid, if the momentum balance has not much influence or is fulfilled since two ports with the same diameter are connected together.

3.2 Property Propagation over Ports

A connector should have only the minimal number of variables to describe the interface, otherwise there will be connection restrictions in certain cases. Therefore, in the connector no redundant variables are present, e.g., the temperature T is not present because it can be computed from the connector variables pressure p and specific enthalpy h .

This approach has one drawback: If two components are connected together, then the medium variables on both sides of the connector are identical. However, due to the connector, only the two equations

$$port1.p = port2.p; \quad port1.h = port2.h;$$

are present. Assume, that p , T are the independent medium variables and that the medium properties are computed at one side of the connections. This means, the following equations are basically present:

$$\begin{aligned}
 port1.h &= h(port1.p, port1.T); \\
 port2.h &= h(port2.p, port2.T); \\
 port1.p &= port2.p; \\
 port1.h &= port2.h;
 \end{aligned}$$

These equations can be solved in the following way:

$$\begin{aligned}
 port1.h &:= h(port1.p, port1.T); \\
 port2.p &:= port1.p; \\
 port2.h &:= port1.h; \\
 0 &= port2.h - h(port2.p, port2.T);
 \end{aligned}$$

The last equation states that $port2.T$ is computed by solving a non-linear system of equations. If $port1.h$ and $port2.h$ are provided as Modelica functions, a Modelica translator, such as Dymola [1], can replace this non-linear system of equations by the equation:

$$port2.T = port1.T;$$

because after alias substitution there are two function calls

$$\begin{aligned}
 port1.h &:= h(port1.p, port1.T); \\
 port1.h &:= h(port1.p, port2.T);
 \end{aligned}$$

Since the left hand sides of the function calls and the first arguments are the same, the second arguments must also be identical, i.e., $port2.T = port1.T$. This type of analysis seems to be only possible, if the specific enthalpy is defined as a **function** of the independent medium variables. Due to this requirement, all media in the Modelica.Media library define the specific enthalpy always as a function and therefore by appropriate tool support no unnecessary non-linear system of equation appears and in the generated code, propagation of medium properties over a connector does not lead to an overhead.

3.3 Upstream Discretization

When implementing a fluid component, the difficulty arises that the value of intensive quantities (such as p , T , ρ) shall be accessed from the upstream

volume. For example, if the fluid flows from volume A to volume B, then the intensive quantities of volume B have negligible influence on the fluid between the two volumes. On the other hand, if the flow direction is reversed, the intensive quantities of volume A have negligible influence on the fluid between the two volumes. Such a situation is handled with the following code fragment:

```
import IF = Modelica_Fluid.Interfaces;
replaceable package Medium =
  Modelica.Media.Interfaces.PartialMedium;
IF.FluidPort_a port1(redeclare package
  Medium = Medium);
IF.FluidPort_b port2(redeclare package
  Medium = Medium);
equation
  // Handle reverse and zero flow
  port1.H_flow = semiLinear(port1.m_flow,
    port1.h, port2.h);

  // Energy and mass balance; here:
  port1.H_flow + port2.H_flow = 0;
  port1.m_flow + port2.m_flow = 0;
  ...
```

The enthalpy flow rate in port1 is in principle computed with an if clause:

```
port1.H_flow = port1.m_flow *
  (if port1.m_flow > 0 then
    port1.h
  else
    port2.h);
```

However, instead of using this if-clause, the corresponding built-in Modelica operator **semiLinear()** is actually used:

```
port1.H_flow = semiLinear(port1.m_flow,
  port1.h, port2.h);
```

The main reason is that this operator will allow a Modelica translator certain symbolic transformations that lead to a more robust numerical computation (see explanation in the Modelica Specification 2.2).

If the above component is connected between two port volumes (*Modelica_Fluid.Pipes.BaseClasses.PortVolume*), i.e., the independent medium variables in port1 and port2 are states, then port1.h and port2.h are either states (i.e., known quantities in the model) or are computed from states at each integration time step. In such a situation, the above if-clause represented by the "semiLinear" operator is uncritical, because it depends only on known variables and can be directly computed.

If instead, say, pressure loss components are connected, then all port variables are unknown and systems of equations occur. For example, three ports, A.port, B.port, C.port, are connected together. This results in the following equations:

Equations due to

connect(A.port,B.port), connect(A.port,C.port):

```
A.port.p = B.port.p = C.port.p
A.port.h = B.port.h = C.port.h
0 = A.port.m_flow + B.port.m_flow +
  C.port.m_flow
0 = A.port.H_flow + B.port.H_flow +
  C.port.H_flow
```

Equations inside components A,B,C:

```
A.port.H_flow = A.port.m_flow*(
  if A.port.m_flow > 0 then A.port.h
  else A.h;
B.port.H_flow = B.port.m_flow*(
  if B.port.m_flow > 0 then B.port.h
  else B.h;
C.port.H_flow = C.port.m_flow*(
  if C.port.m_flow > 0 then C.port.h
  else C.h;
```

where A.h, B.h, C.h, is the specific enthalpy inside the respective component. All equations together form a linear system of equations to compute the mixing enthalpy $A.port.h = B.port.h = C.port.h$ in the connection point. It has the solution [2]:

```
A.port.h = -( (if A.port.m_flow > 0 then 0
  else A.port.m_flow*A.h)+
  (if B.port.m_flow > 0 then 0
  else B.port.m_flow*B.h)+
  (if C.port.m_flow > 0 then 0
  else C.port.m_flow*C.h) )
/ ( (if A.port.m_flow > 0 then
  A.port.m_flow else 0)+
  (if B.port.m_flow > 0 then
  B.port.m_flow else 0)+
  (if C.port.m_flow > 0 then
  C.port.m_flow else 0) )
```

Therefore, independently of the flow directions in the 3 ports, the mixing enthalpy is always uniquely computed, provided at least one mass flow rate does not vanish (see [2] for details how to handle the case if all mass flow rates vanish). From the mixing enthalpy and the port pressure, all other mixing quantities can be computed, such as mixing temperature.

If two ports A and B are connected together, the resulting system of equations has a solution that is unique also for zero mass flow rates:

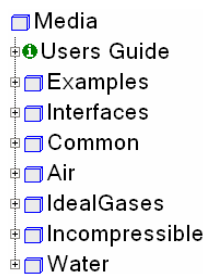
```
A.port.h = if A.port.m_flow > 0 then B.h
  else A.h
B.port.h = A.port.h
```

In some situations, the user can guarantee that the fluid flows only in one direction. In the *Modelica_Fluid* library this can be defined in the *Advanced* menu of components by parameter **flowDirection**. Based on this parameter setting, corresponding "min" and "max" attributes are defined for the mass flow rate in a connector, such as:

```
FluidPort_a port_a(m_flow(min = if
    allowFlowReversal then
    -Modelica.Constants.inf else 0))
```

When `port_a.m_flow` is referenced in a `semiLinear()` operator, the tool can deduce that only one branch of the if-clause can appear and can utilize only this branch for the further symbolic processing. As a result, if-clauses that define the reversing flow are removed.

4 Medium models



Modelica_Fluid uses the free library Modelica.Media that was developed to provide a standardized interface to media models and a large number of ready-to-use media models based on that interface. The basic concept of Modelica.Media is described in [2]. It was included in the Modelica Standard Library in version 2.2. The library has been continuously improved to fit the requirements of Modelica_Fluid. The picture on the left shows the structure of Modelica.Media. Modelica.Media allows for a decoupling of the formulation of the balance equations within a Modelica_Fluid component model and the definition of the medium. Different interfaces are provided in Media.Interfaces that are used as base classes for the implementation of different medium models of different nature, e.g., ideal gases, real gases, two-phase mediums. For every medium a record called ThermodynamicState is implemented that contains the minimum set of variables required to describe the state of the medium. The thermodynamic state record for a pure component ideal gas is

```
record ThermodynamicState
    SI.AbsolutePressure p;
    SI.Temperature T;
end ThermodynamicState;
```

The thermodynamic state record can be used to compute all other fluid properties except for the saturation properties which will be explained later. The functions to compute additional fluid properties are all contained within package Media.Interfaces. A function without an underscore in its name assumes the thermodynamic state record as an input. The function `specificEnthalpy()` for example will compute the specific enthalpy from the thermodynamic state.

The following code fragment demonstrates how the thermodynamic state record could be used in a sim-

ple component model to compute all required fluid properties:

```
replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;
    Medium.ThermodynamicState state;
    Medium.SpecificEnthalpy h;
    ...
state = Medium.setState_pT(1e5, 273.15);
h = Medium.specificEnthalpy(state);
```

The function `setState_pT()` will return the state for the given input variables pressure (p) and temperature (T) independently from the actual entries in the thermodynamic state record. For example, if the medium state is p and h and `setState_pT(.)` is called, for most media a non-linear equation in one unknown will be solved to compute h (this computation is performed reliably and efficiently). The second part of the function name following the underscore indicates the required input variables which is the standard for all function names within Modelica.Media. The more general function to compute the state would be `setState_pTX()` which also requires the nX mass fractions $X[nX]$ for a multiple substance medium as input. Using the thermodynamic state record in models is a more function-based approach to medium modeling and is used in static components, e.g., pressure loss models or the heat transfer to the wall of a pipe.

Modelica.Media also offers an object-oriented approach that uses the model `BaseProperties` defined for each medium interface. This approach is more suitable for dynamic component models, e.g., a volume or a tank, than the function-based approach. The provided base property model can be extended by the user to best meet the specific requirements. The purpose of using the thermodynamic state model in the function based and in the object oriented approach is to be able to write models that are independent of the input variables to the fluid property model. The state selection mechanism described in [2] makes it possible to obtain numerically efficient models for different fluids with the same component models. The basic idea is sketched at hand of the following implementation of a port volume:

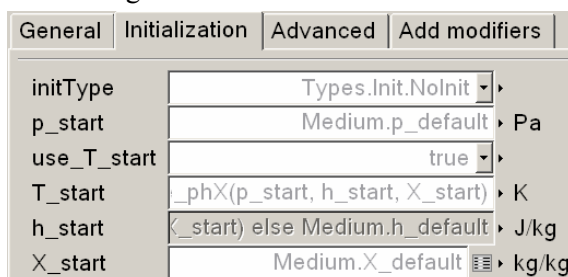
```
replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;
    Modelica_Fluid.Interfaces.FluidPort_a
    port(redeclare package Medium = Medium);
    Medium.BaseProperties medium (
        preferredMediumStates = true);
equation
    medium.p = port.p;
    medium.h = port.h;
    M = V*medium.d;
    U = M*medium.u;
    der(M) = port.m_flow; // mass balance
    der(U) = port.H_flow; // energy bal.
```


In a port volume it is desired that the independent medium variables are used as states (e.g., p,T or p,h depending on the medium). The BaseProperties instance medium contains the basic medium equations. If parameter *preferredMediumStates* is set to **true**, then attribute StateSelect.prefer is set to the independent medium variables and therefore the tool will use these variables as states for the mass and energy balance, if this is possible. This means, that the port volume equations can be implemented without knowledge about the independent medium variables.

Modelica.Media requires the implementation of medium models in Modelica. This approach allows the solver to use as much analytical information about the medium models as possible when manipulating the system of equations. However, it is often also very desirable to use existing fluid property libraries written in C or in FORTRAN. A new interface to an external medium library has been developed for Modelica.Media that supports external medium libraries. This new interface is currently included in the developer version of Modelica.Media and will be tested thoroughly before including it in the Modelica Standard Library.

5 Initialization

Every fluid component with states has a menu “Initialization”. A screen shot of this menu of model Modelica.Fluid.Volumes.MixingVolume is shown in the next figure:



Parameter *initType* defines the type of the initialization and has the following options:

- *initType* == InitialValues:
Initial values of p,X and of T or h are defined.
- *initType* == SteadyState:
The derivatives of the states are set to zero during initialization. Since usually non-linear systems of equations occur, guess values for the states are defined for p, X and for T or h.
- *initType* == SteadyStateHydraulic:
The pressure derivatives are set to zero during initialization, but the thermal states (T or h) are initialized with a start value. Therefore, a guess

value for p and initial values for X and for T or h are defined.

Depending on the selected option, a value such as “p_start” is interpreted from the component as either being an **initial value** (i.e. introducing an initial equation $p = p_start$) or a **guess value** (i.e. setting the start value of p to p_start with fixed = false).

For every medium either T or h can be defined as start value. Assume that T_start is selected as value to be provided (either initial or guess value). Depending on the situation, a tool might use h as iteration variable for a non-linear system of equations, e.g., because h is the independent medium variable. Then, the setting of T_start would have no effect. For this reason, modifiers are defined in the initialization menu, e.g. for h_start:

```
parameter Medium.SpecificEnthalpy h_start=
  if use_T_start then
    Medium.specificEnthalpy_pTX(
      p_start, T_start, X_start)
  else Medium.h_default;
```

If use_T_start is true, the menu for h_start is disabled, i.e., the user cannot input a value and therefore function specificEnthalpy_pTX(.) is called to compute the start value of the specific enthalpy based on p_start and T_start. If use_T_start = false, the user can provide a modifier with a new value that overwrites the if-clause in the modifier. Otherwise the default value of h for this medium is used as initial value.

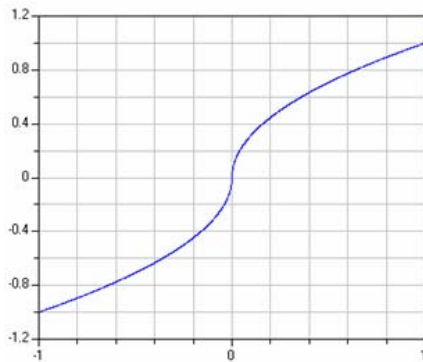
To summarize, the medium is always initialized with a consistent set of variables p, T, h, X where either T or h is computed from the other 3 variables with the corresponding medium function.

6 Regularizing characteristics

Pressure drop equations and other fluid characteristics are usually computed by **semi-empirical** equations. Unfortunately, the developers of semi-empirical equations nearly never take into account that the equation might be used in a simulation program. As a consequence, these semi-empirical equations can nearly never be used blindly but must be slightly modified or adapted in order that obvious simulation problems are avoided. For example, turbulent flow in a pipe might be described by the following type of equation:

$$y = \text{if } x \geq 0 \text{ then } \sqrt{k_1 \cdot x} \\ \text{else } -\sqrt{k_2 \cdot \text{abs}(x)} ;$$

A plot of this characteristic is shown in the next figure:



The difficulty with this function is that the derivative at $x=0$ is infinity. The actual physical characteristic doesn't show this singularity. E.g., for pipe flow, the flow becomes laminar for small velocities and therefore around zero the `sqrt()` function is replaced by a linear function. Since the laminar region is usually of not much practical interest, the above approximation is used.

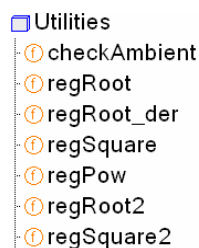
The direct implementation above does not work in Modelica, because an event is generated when $x < 0$ changes sign. In order to detect this event, event iteration takes place. During the event iteration, the active if-branch is not changed. For example, assume that x is positive (= "else" branch) and shall become negative. During the event iteration x is slightly negative and the **else** branch, i.e., `sqrt(x)`, is evaluated. Since this results in an imaginary number, an error occurs. It would be possible to fix this, by using the `noEvent()` operator to explicitly switch off an event:

```
y = noEvent( if x<0 then sqrt(k1*x)
              else -sqrt(k2*abs(x)) );
```

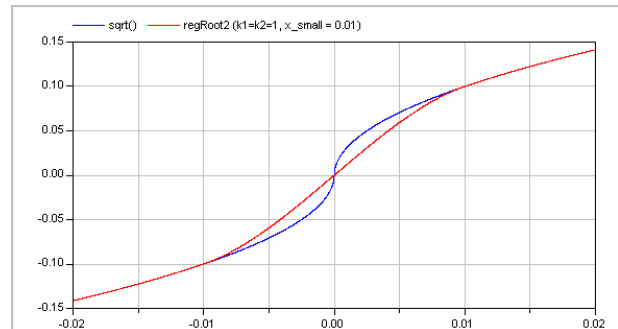
Still, it is highly likely that good integrators will not work well around $x=0$, because they will recognize that the derivative changes very sharply and will reduce the step size drastically.

In Modelica.Fluid.Utilities several utility functions are provided to regularize such types of equations (see screen shot on right side). For example, `regRoot2(.)` replaces the function above by two polynomials of third order around zero, so that the overall function is continuous, is strict monotonically increasing and has a continuous first derivative everywhere. Additionally, either the second derivatives of the two polynomials at zero are identical (= default) or a user defined first derivative at zero can be provided, to, e.g., correctly describe the laminar region around zero. In the first case, the equation above is replaced by:

```
y = regRoot2(x, x_small, k1, k2);
```



where x_small defines the region of the newly introduced two polynomials around $x = 0$. The result of applying this function is shown in the next figure.



The "blue" curve is the exact characteristic according to the equation above, whereas the "red" curve is the regularized approximation of `regRoot2(.)` that has much better numerical properties.

7 Selected Components

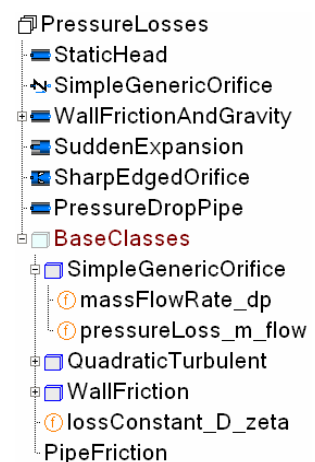
In the previous sections, the features have been described that are needed in order that component models can be implemented. In this section some of the provided component models will be shortly sketched.

7.1 Pressure Losses

Package PressureLosses contains models and functions providing pressure loss correlations. All models in this library have the property that no mass and no energy is stored in the component. Therefore, none of the models has a state. The basic correlations are **models** that are implemented with **functions** of sublibrary PressureLosses.BaseClasses.

These functions might also be directly called (e.g. in an implementation of another component, such as the distributed pipe).

All functions are continuous and have a finite, non-zero, smooth, first derivative. The functions are all guaranteed to be strict monotonically increasing. The mentioned properties guarantee that a unique inverse of every function exists. In fact, for all correlations a function is provided in the form $m_flow = f(p)$ and also its inverse, $p = g(m_flow)$ is given. A similar naming convention as in the Media

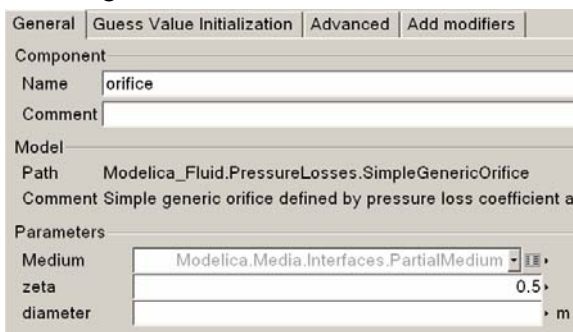


library is used, e.g. `massFlowRate_dp(..)` means that the functions compute the mass flow rate and that the input argument is `dp` (the pressure difference between two ports). Most functions consist of one statement, so that, e.g., Dymola inlines the function and therefore no call overhead is present.

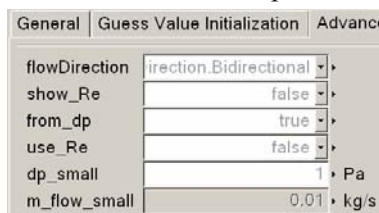
The pressure loss correlation “SimpleGenericOrifice” defines a standard quadratic correlation of the form:

$$\Delta p = \frac{1}{2} \cdot \zeta \cdot \rho \cdot v |v|$$

where Δp is the pressure difference between two ports, v is the fluid velocity (that can be computed from the mass flow rate, density and pipe area) and ζ is the constant pressure loss coefficient, for the fluid flow from port_a to port_b that can be, e.g., deduced from some of the standard books like Idelchick [3]. Screen shots of the parameter menu are shown in the next two figures:



Basically, the medium, the correlation factor and the diameter has to be defined at which ζ is defined. The “Advanced” menu is the same for all components of the PressureLosses package and defines how the computation of the correlation is performed:

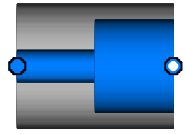


If `from_dp` is true, the mass flow rate is computed from the pressure drop, otherwise the computation is reversed. The “flowDirection” defines whether reversal flow shall be taken into account. “use_Re” defines the laminar region by the Reynolds number (e.g. $Re < 2000$ for smooth wall friction), otherwise it is defined approximately by a small pressure drop or a small mass flow rate depending on the selected computation direction. Finally, if `show_Re` = true, the Reynolds-Number is computed in order to utilize it, e.g., in a plot. By default the computations with the Reynolds number are not performed, since a me-

dium model may not provided a function to compute the viscosity.

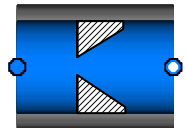
Model “suddenExpansion” defines a sudden expansion of a pipe and computes the correlation factors for the two flow directions from the two pipe diameters according to Idelchick [2].

suddenExpansion



In the same way “orifice” defines a sharp edged orifice where the correlation factors for the two flow directions depends, e.g., on the opening angle of the orifice [2].

orifice



Model “StaticHead” models only the pressure drop due to gravity.

Finally, model “WallFrictionAndGravity” models wall friction and also takes into account gravity. The implementation is based on [2,3]. The user can select either the different regions (only laminar, only quadratic turbulent, laminar + quadratic turbulent) or the detailed characteristic. The latter one is shown in the next figure [2,3].

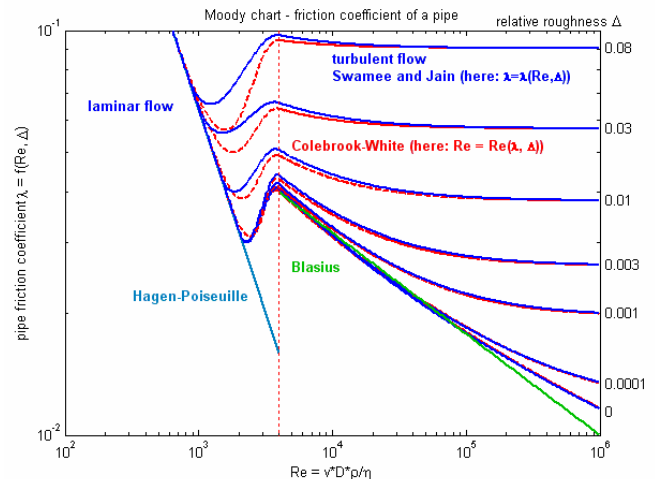


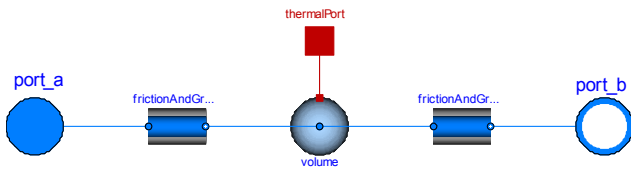
Figure 1. Moody Chart: $\lg(\lambda) = f(\lg(Re), \Delta)$, $\zeta = \lambda L/D$

7.2 Pipes

Different pipe models are defined in package Pipes, as shown in the screen shot at the right. LumpedPipe is a simple pipe model consisting of one volume and two pressure loss correlations for the wall friction, as well as a heat transfer port to describe the heat transfer through the wall. The model is especially useful for demonstration purposes because it is just built from basic components:

Pipes

- LumpedPipe
- DistributedPipe_thermal
- DistributedPipe_hydraulic
- BaseClasses



The other two pipe models are discretized pipes consisting of n volumes. More details are given in the next subsection.

7.3 Heat Exchanger

A basic heat exchanger model can be found under `Components.HeatExchangers.BasicHX`. It demonstrates the usage of several models from the Fluid library and the interfaces provided to adapt them to fit personal needs. The heat exchanger is composed of two pipe flow models and one wall element as shown in figure 4. The wall determines a co- or counterflow orientation of the two medium flows. It also adds the major thermal capacity to the set. Heat conduction is assumed to be one-dimensional, perpendicular to both fluid flows.

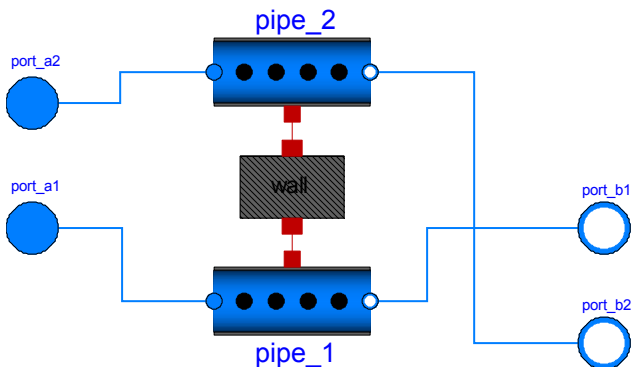


Figure 4: Heat exchanger component

On both fluid sides medium packages from the Modelica.Media library can be chosen. An instance of the respective `BaseProperties` model as described in section 4 is automatically included in each of the two distributed flow models from the component package `Pipes`. They follow an upwind discretization scheme, the number of segments being the same for both pipes and the wall. Dynamic energy and mass balances interlace on a staggered grid with static momentum balances for each control volume. Two half momentum balances on each end make the component fully symmetric. The port interface corresponds to the general design principle outlined in section 3 and allows for flow reversal. A uniform cross sectional area is assumed along the entire flow path.

Empirical heat transfer and pressure drop correlations allow us to reduce 3D fluid flow problems to

one dimension. They largely depend on the specific application, thus have to be replaceable in a model in order to provide the required flexibility, but at the same time need to be known in the lowest hierarchical level of a system, the governing balance equations.

The distributed pipe model contains a replaceable heat object that determines the relationship between the thermal port properties, heat flow and temperature, and the bulk flow, namely the medium temperature and the sensible heat term in the energy balance. The library currently only provides the simplest model possible to describe a sensible heat transfer, by means of a constant heat transfer coefficient. But an implementation of e.g. Nusselt correlations from the literature is easily done by inheriting from the base model `Pipes.BaseClasses.HeatTransfer.PartialPipeHeatTransfer`. Besides geometrical parameters, such as the hydraulic diameter and cross sectional flow area the heat object also “knows” mass flow rate and the `medium.state`

record (see section 4) of the fluid flow, which makes it possible to compute required transport properties by function call if and only if needed in the respective correlation. For further information concerning the models mentioned here the reader may be referred to the online documentation of the library.

Figure 5 shows the results of an example model in the library. One of the two fluid flows in the heat exchanger changes its direction midway, and because it is fed from a colder source changes the direction of heat flow.

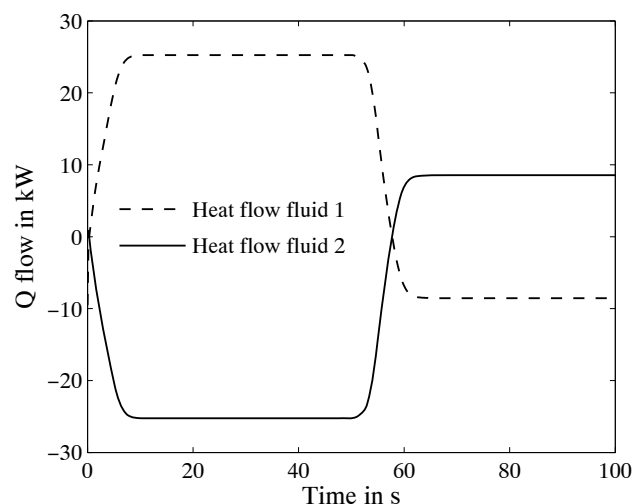


Figure 5: Heat flow rates in both heat exchanger fluids (water) while one of them changes direction.

8 Conclusions

The 1.0 Beta 1 version of the Modelica_Fluid library described in this article is in a rather stable stage and the most important basic problems have been resolved. Especially, it was possible to reach the following quite ambitious goals:

(a) The component equations are independent from the medium equations (especially, a component can be used for media that have different sets of independent variables, such as T , pT , or p,h , or p,T,X or T,X etc.). This has the big advantage that pump, pipe, valve models etc., can be implemented just once and utilized for quite different media. Of course, there are limits, e.g., one and two phase flow is always differently described in a component. On the other hand, all components of the Fluid library support incompressible and compressible as well as one and multiple substance media.

(b) Components can be arbitrarily connected together. Also models such as a pipe can be flipped. The Modelica connection semantics generates ideal mixing equations so that the mass and energy balance is fulfilled. If this is not desired, junction models have to be used. This is especially the case when the momentum balance in a junction cannot be neglected. There are still some unresolved issues, e.g., the Pipes.DistributedPipe model is discretized in such a form that at the two ends of a pipe momentum balances are present (and not mass and energy balances of a volume). When connecting pipes of this form directly together (without using a port volume in the connection point), non-linear systems of equations appear.

The goal is to continuously improve the Modelica_Fluid library, especially to include more component models. Contributions from users of the library are welcome. The actual version of the library can be downloaded from

<http://www.modelica.org/library/>

9 Acknowledgments

The development of the Modelica_Fluid library started in year 2002 and many have contributed:

The Fluid library development was organized in 2002-2004 by Martin Otter and since 2004 it is organized by Francesco Casella. The essential basic design of the Fluid library, especially component interfaces, handling of reversing flow with the semiLinear() operator, property propagation is from Hilding Elmqvist. Besides the authors, the following people contributed to the fluid component models,

examples, symbolic algorithms and the further design of the library (alphabetical list): John Batteh, Jonas Eborn, Rüdiger Franke, Anton Haumer, Henning Knigge, Chuck Newman, Hans Olsson, Katja Poschlad, Manuel Remelhe, Sven Erik Mattsson, Mike Tiller, Allan Watson.

The Modelica.Media library development is organized by Hubertus Tummescheit. For the long list of contributors, see

Modelica.Media.UsersGuide.Contact.

10 References

- [1] Dynasim (2006). *Dymola Version 6.0*. Dynasim AB, Lund, Sweden. Homepage: <http://www.dynasim.se/>.
- [2] Elmqvist, H., Tummescheit H., and Otter M. (2003). *Object-Oriented Modeling of Thermo-Fluid Systems*. Proceedings of 3rd Int. Modelica Conference, Linköping, Sweden, ed. P. Fritzson, pp. 269-286. http://www.modelica.org/Conference2003/papers/h40_Elmqvist_fluid.pdf
- [3] Idelchik I.E. (1994): *Handbook of Hydraulic Resistance*. 3rd edition, Begell House, ISBN 0-8493-9908-4.
- [4] Kral A., Haumer A. Plainer M. (2005): *Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library*. 4th int. Modelica Conference, Hamburg-Harburg. http://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3b1.pdf
- [5] Poschlad K., Remelhe M.A.P., and Otter M. (2006): *Modeling of an Experimental Batch Plant with Modelica*. 5th int. Modelica Conference, Vienna.
- [6] Rüdiger Franke (2003): *On-line Optimization of Drum Boiler Startup*. Proceedings of the 3rd Int. Modelica Conference, Linköping, 2003. http://www.modelica.org/events/Conference2003/papers/h29_Franke.pdf

Shock Wave Modeling for Modelica.Fluid library using Oscillation-free Logarithmic Reconstruction

José Díaz López
email: jose.diaz@dynasim.se
Dynasim AB

Research Park Ideon, SE-223 70 Lund, Sweden

Abstract

An oscillation-free discretisation of conservation laws has been implemented using Modelica.Fluid library [1] with a novel logarithmic reconstruction technique. Roe's approximation flux is used in combination with it. The results show that this logarithmic reconstruction has many advantages: limiter-free, absence of spurious oscillations near shock waves as well as third order of approximation to the solution.

Keywords: *Finite Volume Methods, Shock waves, Roe's Flux approximation, Logarithmic reconstruction*

1 Introduction

This paper presents an implementation of the logarithmic reconstruction method for hyperbolic partial differential equations in conservation law form, as presented in [7], using Modelica.Fluid [1].

The main objective is to resolve numerically shock waves avoiding spurious oscillations, limiters, artificial viscosity or wave velocity estimators.

2 Problem Formulation

Scalar hyperbolic partial differential equations (HPDE's) are usually formulated in the following compact way

$$u_t + f(u)_x = 0 \quad (1)$$

where $u : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^m$ is the solution and $f : \mathbb{R}^m \rightarrow \mathbb{R}^m$ is the flux. The subindices represent partial derivation respect to time (t) and space (x) variables. Some classical case studies in HPDE's are

- **Advection Equation** where $x \in [a, b]$, $u : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, $f \equiv I$ with boundary condition $u(t, a) = g(t)$ and initial value $u(0, x) = h(x)$.
- **Burgers' Equation** where $x \in [a, b]$, $u : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, $f(u) = \frac{1}{2}u^2$, boundary condition $u(t, a) = g(t)$ (or periodic) and initial value $u(0, x) = h(x)$.

- **Euler System.** If the section A of the pipe is constant we have that $x \in [a, b]$,

$$u = \begin{pmatrix} \rho A \\ \rho v A \\ \rho e_0 A \end{pmatrix},$$

$$f(u) = \begin{pmatrix} \rho v A \\ (\rho v^2 + p) A \\ \rho v h_0 A \end{pmatrix}$$

boundary conditions and initial values for ρ, v and p . The variables denote physical quantities as follows

ρ	mass density
v	fluid speed
e_0	stagnation internal energy
h_0	stagnation enthalpy
p	pressure

The stagnation internal energy is related to the specific internal energy as

$$e_0 = e + \frac{1}{2}v^2,$$

and h_0 satisfies

$$h_0 = e_0 + \frac{p}{\rho}.$$

A source term $S(u)$ is included if the section A varies gradually. In this case, the flow is called *quasi-one-dimensional*. The conservation law takes the form

$$u_t + f(u)_x = S(u). \quad (2)$$

This source term is defined as

$$S(u) = \begin{pmatrix} 0 \\ -p \frac{dA}{dx} + \rho q A \\ -\rho q A \end{pmatrix}$$

where q is the thermal conductivity constant and $G = \frac{1}{2}\rho v |v| k \frac{4}{D}$, k the wall friction coefficient and D the equivalent diameter of the duct. The source term can also include effects related to temperature

$$S(u) = \begin{pmatrix} 0 \\ -p \frac{dA}{dx} + \rho q A \\ -\rho q A + k A \frac{\partial^2 T}{\partial x^2} + \dot{Q} \end{pmatrix}$$

where T is temperature, k is the thermal conductivity (considered constant along the pipe) and \dot{Q} is the heat flow. The system is completed by equations that are characteristic to the medium, that is

$$p = p(\rho, T), \quad e = e(\rho, T).$$

The three equations of this system have physically relevant names: *mass*, *momentum* and *energy* balance equations. We will refer to them later on with those names.

3 Numerical flux and reconstruction

3.1 Finite volume methods for HPDE's

Consider now the semidiscretisation of (1) by means of finite volumes (FV). This method yields a system of ODE's that are numerically integrated in time. For a survey in FV-standard notation, see also [2].

The method can be briefly summarised as follows. Consider the domain $\Omega = [a, b]$, divided into n segments (x_i, x_{i+1}) with $a = x_0 \leq x_1 < \dots < x_i < x_{i+1} < \dots < x_{n+1} = b$. We use also as notation $\Delta x_i = x_{i+1} - x_i$ and

$$x_{i+\lambda} = \lambda x_i + (1 - \lambda)x_{i+1} \quad \lambda \in (0, 1).$$

The so-called computing cells are therefore defined as

$$C_i = [x_{i-1/2}, x_{i+1/2}], \quad i = 1, \dots, n.$$

For the numerical integration in time, consider as state variables the averages

$$\bar{u}_i(t) = \frac{1}{\Delta x_i} \int_{C_i} u(x, t) dx, \quad (3)$$

where $\Delta x_i = x_{i+1/2} - x_{i-1/2}$, the length of C_i . The main objective now is to state the governing equations for this averages. This is the process of *semidiscretisation* of the HPDE. This method is inspired by the integral form of (1). The *semidiscretised* ODE system is based on the construction of a flux approximation from the left and right states at $x_{i+1/2}$

$$f(u(x_{i+1/2}, t)) \approx \hat{f}_{i+1/2}(u^-(x_{i+1/2}, t), u^+(x_{i+1/2}, t)).$$

The function \hat{f} is the so-called *numerical flux approximation*, and governs the evolution of \bar{u}_i as follows

$$\frac{d}{dt} \bar{u}_i = \frac{1}{h} (\hat{f}_{i+1/2} - \hat{f}_{i-1/2}), \quad i = 1, \dots, n \quad (4)$$

where the sign \pm indicates the side of x_i taken from the reconstruction of u .

The focus now on lies exclusively on \bar{u}_i and the numerical flux approximation $\hat{f}_{i+1/2}$. The reason is that different properties of the conservation law are inherited by the numerical simulation, depending on the reconstruction method for u^\pm and $\hat{f}_{i+1/2}$.

To show the reasons for requiring limiter-free, non-oscillatory reconstruction and viscous-free flux approximation, two simpler approaches are presented before. One of them is already implemented in Modelica.Fluid.

3.2 A primer approach

In [1] we find an approach to discretising the pipe example presented before using Modelica. The discretisation uses the following approximations

$$\bar{u}_i = u(x_i, t) \Delta x_i \quad (5a)$$

$$\hat{f}_{i+1/2} = f(u^-(x_{i+1/2}, t)) \quad (5b)$$

and piecewise constant reconstruction of u in C_i for the mass balance and energy balance equations. The same setup is used for the moment balance equation, except that the grid used here is then staggered, see [6]. The argumentation presented in [6] is that staggering results in a velocity and pressure fields that are physically meaningful (pp. 120). Spurious oscillations are not allowed as solutions in the velocity field. It is also stated that the implementation is cumbersome. This staggered-grid formulation is even more troublesome for object-oriented modeling.

The example model of a distributed pipe found in Modelica.Fluid.Components.Pipes, called DistributedPipeFV, implements a non-staggered grid, with the setup described above. The discretisation is found in Modelica.Fluid.BaseClasses.Pipes.Flow1d_FV.

3.3 Lax-Friedrichs numerical flux

The Lax-Friedrichs numerical flux is classically the easiest possibility of a numerical flux. This numerical flux is unstable and needs a damping factor α to avoid spurious oscillations. The numerical flux is defined as

$$\hat{f}_{i+1/2}^{LF} = \frac{1}{2} (f(u^-(x_{i+1/2}, t)) + f(u^+(x_{i+1/2}, t))) - \frac{1}{2} \alpha (u^+(x_{i+1/2}, t) - u^-(x_{i+1/2}, t)) \quad (6)$$

The principal problem with this approach is that the parameter α has to be tuned to avoid unnecessary damping of shock waves and corners. This parameter is related to the traveling wave velocities and is in general difficult to estimate them *a priori*. Furthermore, there is no easy way to treat waves traveling in both left and right directions with this scheme.

We want to avoid those artificial factors as much as possible. The cause of such spurious oscillations is a too coarse estimations of u^+ and u^- .

3.4 Reconstruction of u

Another classical approach is to consider linear reconstruction of u in C_i , see [4, 12]. In this case,

$$u^+ = \bar{u} + \frac{\Delta x_i}{2} m^+$$

and

$$u^- = \bar{u} - \frac{\Delta x_i}{2} m^-$$

, where the slopes m^+ and m^- are estimated from the averages \bar{u}_{i-1} , \bar{u}_i and \bar{u}_{i+1} . This approach needs the so called

limiter, simply a bounded function $L(m)$. The reconstruction takes the form

$$u^+ = \bar{u} + \frac{\Delta x_i}{2} L(m^+)$$

and

$$u^- = \bar{u} - \frac{\Delta x_i}{2} L(m^-)$$

instead. The main advantage is that limiters avoid spurious oscillations, but the maximum approximation order of reconstruction is two, if the limiter is constructed in an appropriate way, see [4].

Higher order of approximation requires other techniques.

3.5 Oscillations in DistributedPipeFV

Consider the Sod problem, described in [12]. The Sod problem consists of a domain with a perfectly isolating membrane in the middle that separates a zone with high pressure and temperature of a zone with low pressure and temperature. The dynamics of the fluid in the whole domain are governed by Euler equations. At time $t = 0$, the membrane is removed instantaneously. Further details about the configuration are given later in Sec. 4.

The corresponding Modelica model is implemented with two DistributedPipeFV models connected with adequate initial conditions, depicted in Fig. 1. The reason for having two pipes is that the Sod problem has initially waves traveling in both left and right from the membrane. Since we want to experiment with Lax-Friedrichs numerical flux also, we need to specify two different α coefficients in two different domains.

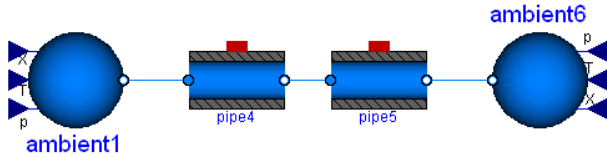


Figure 1: Sod problem model using two connected pipes

3.5.1 Original approach

The implementation of the original approach is in Modelica.Fluid.BaseClasses.Pipes.Flow1D.FV. We present only the dynamical equations of the model since this is the focus of this paper.

The implementation of equations (5) can be observed in the listings 1 (mass and energy balance) and 2 (momentum balance).

Listing 1: Mass and Energy Balance

```
//Mass and energy balance
for i in 1:n loop
  if static then
    ...
  else
    der(m[i]) = m.flow[i] - m.flow[i+1] + ms.flow[i];
    der(mXi[i, :]) = mXi.flow[i, :] - mXi.flow[i+1, :] + msXi.flow[i, :];
    der(U[i]) = H.flow[i] - H.flow[i+1] + Qs.flow[i];
  end if;
end for;
```

Listing 2: Momentum Balance

```
// Momentum Balance
for i in 2:n loop
  F.p[i] = (medium[i-1].p-medium[i].p)*A.inner;
  F.f[i] = -dp[i]*A.inner;
  (if dynamicTerm then der(m.flow[i])*length/n else 0) =
    F.p[i] + F.f[i] + (I.flow[i-1]-I.flow[i+1])/2;
end for;
```

In this approach, the boundary conditions are easily included: the ports port_a and port_b are considered the nodes 0 and $n+1$ of the discretisation, and included when the array index of the component medium is less than 1 or larger than n .

The result is depicted in Fig. 2, and the spurious oscillations appear clearly in the two shock waves.

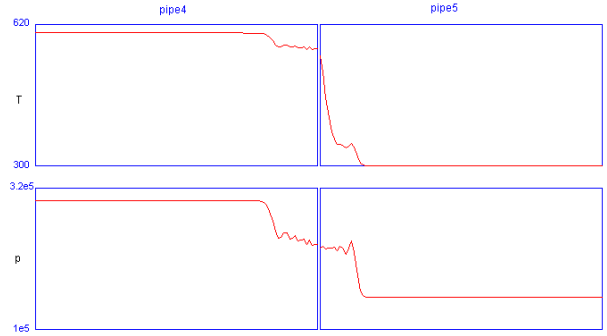


Figure 2: Spurious oscillations on top the shock waves in the Sod problem

3.5.2 Lax-Friedrichs Flux and upwinding

Lax-Friedrichs flux avoids oscillations and is easy to implement. With some tuning of the α factor, this flux can yield better results. The *sine qua non* condition of LF-flux is that the shock waves are traveling in one direction.

The implementation is presented in the listings 3 and 4.

Listing 3: Mass and Energy Balance (LF)

```
//Mass and energy balance
for i in 2:n-1 loop
  if static then
    ...
  else
    der(m[i]) = m.flow[i] - m.flow[i+1] + ms.flow[i]
      + alpha*(-m.flow[i-1]+2*m.flow[i]-m.flow[i+1]);
    der(mXi[i, :]) = mXi.flow[i, :] - mXi.flow[i+1, :] + msXi.flow[i, :]
      + alpha*(-mXi.flow[i-1, :]+2*msXi.flow[i, :]-mXi.flow[i+1, :]);
    der(U[i]) = H.flow[i] - H.flow[i+1] + Qs.flow[i]
      + alpha*(-H.flow[i-1]+2*H.flow[i]-H.flow[i+1]);
  end if;
end for;
```

Listing 4: Momentum Balance (LF)

```
for i in 2:n-1 loop
  F.p[i] = (medium[i-1].p-medium[i].p)*A.inner
    + alpha*(-medium[i-1].p+2*medium[i].p-medium[i+1].p)*A.inner;
  F.f[i] = -dp[i]*A.inner;
  (if dynamicTerm then der(m.flow[i])*length/n else 0) =
    F.p[i] + F.f[i]
    + (I.flow[i-1]-I.flow[i+1])/2;
end for;
```

Notice that the main difference is the added terms multiplied by α . The results are depicted in Fig. 3, before the left shock wave bounces against ambient1, and in Fig.4 after

the wave bouncing. As we already mentioned, the spurious oscillations appear since α is negative in pipe4.

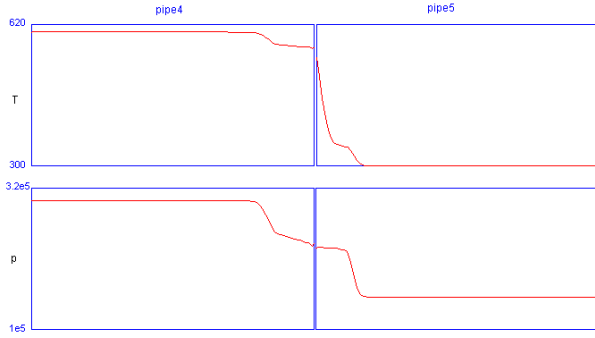


Figure 3: Lax-Friedrichs Flux with $\alpha = -0.1$ in pipe4 and $\alpha = 0.1$ in pipe5. (Before first bouncing.)

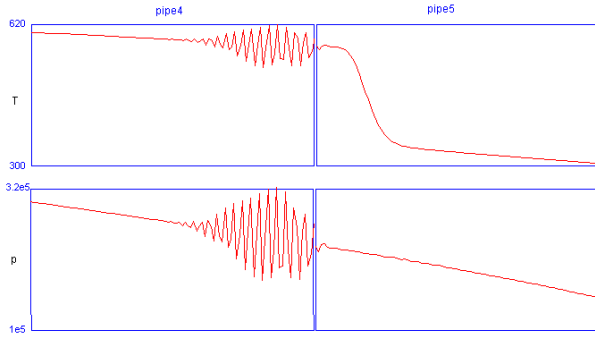


Figure 4: Lax-Friedrichs Flux with $\alpha = -0.1$ in pipe4 and $\alpha = 0.1$ in pipe5. Spurious oscillations appear in pipe4 after the reflexion of a small shock wave at the left boundary.

3.6 Logarithmic reconstruction

Logarithmic reconstruction is a technique recently developed mainly by A. Schroll in the papers [8, 9, 7] with examples in [10]. Schroll's scheme is called **LDLR**, standing for *Local Double Logarithmic Reconstruction*. LDLR is as a natural generalisation of A. Marquina's hyperbolic reconstruction presented in [5]. LDLR has also the advantage over other approximations of higher order (as those studied by S. Serna and A. Marquina in [11], based on their own variant of the weighted ENO schemes) that shock and rarefaction waves are resolved correctly without any estimates, thresholds or heuristics.

When using LDLR for reconstruction, u is represented as a piecewise logarithmic function. In particular, the left state u^+ and the right state u^- at $x_{i+1/2}$ are estimated as

$$u^\pm = u_i + c_3 \Delta x_i \eta^\pm(c_1) + c_4 \Delta x_i \eta^\pm(c_2) \quad (7)$$

where

$$\eta^+(t) = -\frac{\log(1-t) + t}{t^2}$$

$$\eta^-(t) = -\frac{(t-1)\log(1-t) - t}{t^2}$$

with appropriate constants c_1, c_2, c_3 and c_4 for third order convergence.

The computation of the constants is relatively simple. We describe the computation in the following algorithm

1. Calculate

$$d_1 = \frac{\bar{u}_i - \bar{u}_{i-1}}{\Delta x_{i-1}}, d_2 = \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x_i}. \quad (8)$$

2. The coefficient c_1 is obtained from

$$c_1(d_1, d_2) = (1 - \text{tol}) \left(1 + \text{tol} - \frac{2|d_1|^q |d_2|^q + \text{tol}}{|d_1|^{2q} + |d_2|^{2q} + \text{tol}} \right)$$

where $\text{tol} = 0.1 \Delta x_i^q$ and typically $q = 1.4$. This factor q controls the local variation, and the larger q the smaller the local variation.

3. The constants c_2, c_3 and c_4 are calculated from

$$\begin{aligned} c_2 &= \frac{c_1}{c_1 - 1} \\ c_3 &= \frac{(c_1 - 1)(d_2(1 - c_2) - d_1)}{c_2 - c_1} \\ c_4 &= d_1 - c_3. \end{aligned}$$

4. Using (7) we can now calculate u^- .

The procedure for u^+ is very similar. The only difference is that d_1 and d_2 are defined instead as following

$$d_1 = \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x_i}, d_2 = \frac{\bar{u}_{i+2} - \bar{u}_{i+1}}{\Delta x_{i+1}}.$$

3.7 Roe's numerical flux and upwind schemes

Upstream and downstream propagation is used in upwind schemes. This is intrinsic and physically meaningful information contained in the HPDE. The cornerstone is to decompose the flux difference at $x_{i+1/2}$ as $\hat{f}^+ - \hat{f}^- = W^+ + W^-$, according to the physics of the Euler equations. Consider the left and right states of u (as defined for **Euler system**) at $x_{i+1/2}$. Define Roe's averages as (we drop the subindex for simplicity in this case)

$$\bar{\rho}^2 = \rho^+ \rho^- \quad (9a)$$

$$\bar{v} = \frac{\sqrt{\rho^+} v^+ + \sqrt{\rho^-} v^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \quad (9b)$$

$$\bar{h} = \frac{\sqrt{\rho^+} h_0^+ + \sqrt{\rho^-} h_0^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \quad (9c)$$

$$\bar{a}^2 = (\gamma - 1)(\bar{h} - 1/2 \bar{v}^2). \quad (9d)$$

Using Roe's averages and the differences

$$\Delta p = p^+ - p^-, \Delta \rho = \rho^+ - \rho^-, \Delta v = v^+ - v^-,$$

we can estimate the traveling wave velocities as follows

$$\lambda_1 = \tilde{u} - \tilde{a} \quad (10a)$$

$$\lambda_2 = \tilde{u} \quad (10b)$$

$$\lambda_3 = \tilde{u} + \tilde{a}. \quad (10c)$$

These travel velocities have associated wave forms, estimated as

$$W_1 = \begin{pmatrix} 1 \\ \tilde{u} - \tilde{a} \\ \tilde{h} - \tilde{u}\tilde{a} \end{pmatrix} \quad (11a)$$

$$W_2 = \begin{pmatrix} 1 \\ \tilde{u} \\ \frac{1}{2}\tilde{h} \end{pmatrix} \quad (11b)$$

$$W_3 = \begin{pmatrix} 1 \\ \tilde{u} + \tilde{a} \\ \tilde{h} + \tilde{u}\tilde{a} \end{pmatrix} \quad (11c)$$

and associated strengths, estimated as

$$a_1 = \frac{1}{2\tilde{a}^2}(\Delta p - \tilde{\rho}\tilde{a}\Delta u) \quad (12a)$$

$$a_2 = \frac{1}{\tilde{a}^2}(\tilde{a}^2\Delta \rho - \Delta p) \quad (12b)$$

$$a_3 = \frac{1}{2\tilde{a}^2}(\Delta p + \tilde{\rho}\tilde{a}\Delta u). \quad (12c)$$

$$(12d)$$

With all this information, we can decompose the traveling waves, according to velocities λ_i . In a compact notation we can write

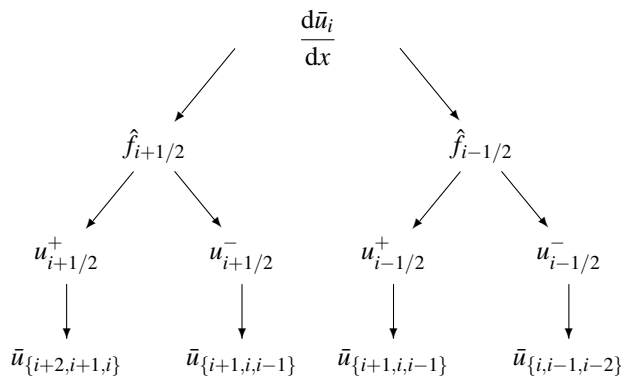
$$W^+ = \sum_{\lambda_i > 0} a_i W_i \quad (13a)$$

$$W^- = \sum_{\lambda_i \leq 0} a_i W_i \quad (13b)$$

with $i = 1, 2, 3$. For more complex schemes considering contact traveling waves, see [4, 12].

3.8 LDLR Stencil and Roe's Flux

The LDLR is symmetric as long as the numerical flux keeps symmetry. The following dependency graph shows the dependency of $\frac{d\tilde{u}_i}{dx}$ on \tilde{u}_i , $i = 1, 2, \dots, n$.



We observe that $\frac{d\tilde{u}_i}{dx}$ depends on $\tilde{u}_{i+2}, \tilde{u}_{i+1}, \tilde{u}_i, \tilde{u}_{i-1}$ and \tilde{u}_{i-2} . Thus, treatment of the boundary conditions is needed for $\frac{d\tilde{u}_1}{dx}$ and $\frac{d\tilde{u}_n}{dx}$.

Boundary Conditions Consider $i = 1$ in Eq. (4). To compute $\frac{d\tilde{u}_1}{dx}$ we need the values of $\hat{f}_{1+1/2}$ and $\hat{f}_{1-1/2}$, constructed from the available data and the boundary conditions. We use the *ghost cell* approach to solve this problem. The ghost cell technique is applied by extending the domain beyond the boundary with new cells. The amount of those new cells is as many as needed to use the same central numerical approximation.

In our case, to compute $\frac{d\tilde{u}_1}{dx}$ we would need u_3, u_2, u_1, u_0 and u_{-1} because of the size of the stencil. We have added two ghost cells u_0 and u_{-1} . Their value has to be design to meet the boundary conditions. For simplicity, we just consider $u_0 = u_{-1} = u(a, t)$ for this study.

The same reasoning is also valid for $i = n$, where two ghost cells u_{n+1} and u_{n+2} are to be introduced. In the same fashion, we consider $u(b, t) = u_{n+1} = u_{n+2}$.

Clearly, this approximation means a loss of accuracy at the boundaries, where the order of approximation drops. Of course, more sophisticated numerical schemes may be derived to meet higher approximation orders, using the two new degrees of freedom introduced.

3.9 Source term discretisation

Even though we don't explore the source term in this paper, we provide a way of handling the source term for completeness.

The semidiscretised system considering source term S takes the form

$$\frac{d}{dt}\tilde{u}_i = \frac{1}{h}(\hat{f}_{i+1/2} - \hat{f}_{i-1/2}) + S_i, \quad i = 1, \dots, n. \quad (14)$$

To model the term S_i , we use the speeds obtained by Roe's numerical flux. The decision is then taken using λ_1 and λ_3 from Eq. (10)

$$S_i = \begin{cases} S(u_{i-1/2}^+), & \lambda_1 > 0 \\ \frac{\lambda_1 S(u_{i-1/2}^-) - \lambda_3 S(u_{i+1/2}^+)}{\lambda_1 - \lambda_3}, & \lambda_1 \leq 0 \leq \lambda_3 \\ S(u_{i+1/2}^-), & \lambda_3 < 0 \end{cases} \quad (15)$$

In case we include thermodynamical effects in S_i , we have to estimate the temperature gradient. It is important then to use a scheme with at least *third order*. This is important to keep the convergence order, since LDLR approximation is of third order, [7].

4 Applications

We will explore now the results of the LDLR implementation with Roe's flux using Modelica.Fluid. For details of the implementation, see section 5. We come back first to the

two pipe problem presented in Fig. 1, and then some other examples showing different aspects of the approach.

In all experiments, the chosen medium was `IdealGases.SingleGases.CH3COOH` from `Modelica.Media`. The length of the pipes is 10 cm and 20 cm the larger ones. The discretisation used is 10 cells/cm, that is, 100 cells for the short pipes and 200 for the large pipes. As before, the high pressure and temperature zone has $p_h = 3 \cdot 10^5$ Pa and $T_h = 600$ K. The low pressure and temperature zone has $p_l = 1.5 \cdot 10^5$ Pa and $T_l = 300$ K.

4.1 Two pipe Sod problem

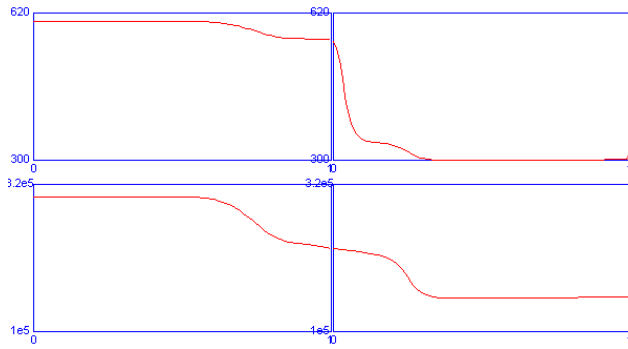


Figure 5: Oscillation-free LDLR solution and Roe's flux of the two pipe Sod problem

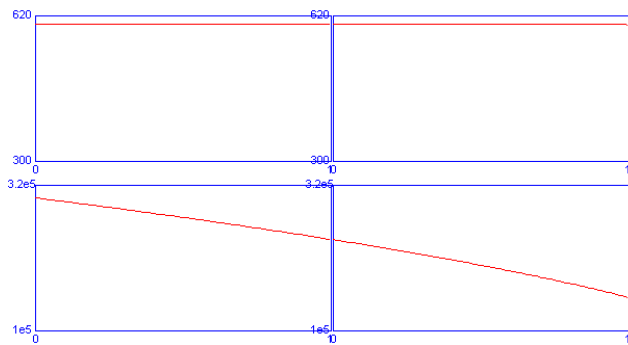


Figure 6: Steady State of the Two pipe Sod problem with LDLR and Roe's Flux.

With this scheme, the right upwinding and the logarithmic reconstruction makes the solution oscillation-free. We observe in particular that steady state is reached without any trouble, depicted in Fig. 6. This is not possible with the original approach or the Lax-Friedrichs flux in the general case. Fig. 5 depicts the solution given by the new scheme at the same time point depicted in Fig. 2 and Fig. 3.

4.2 One pipe Sod problem

We want to establish now how is the behavior of the ghost cell implementation of the boundary conditions and connectors. For that, we simulate the model depicted in Fig. 7.

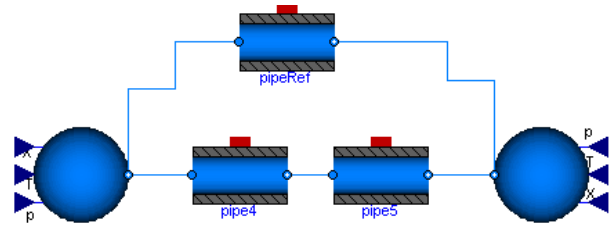


Figure 7: Model for boundary condition testing.

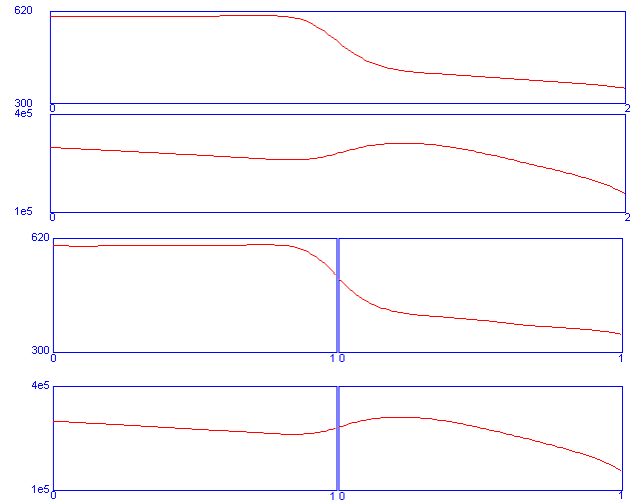


Figure 8: Simulation result of the comparison model

The model has a large pipe that undergoes the same conditions as two series connected pipes. The pressure and temperature profiles are then compared. Optically, the solution is not distinguishable, as depicted in Fig. 8.

When plotting the difference, depicted in Fig. 9, we observe the differences. In both temperature and pressure, the difference is around two orders of magnitude less than the variables. This difference causes traveling waves that does not influence very much the behavior of the system. Furthermore, this difference goes to zero as the number of discretisation intervals is increased.

4.3 Symmetry

For testing symmetry of the LDLR with Roe's approximation, we simulate the model depicted in Fig. 10. Simply two pipes in parallel and one of them with interchanged connections.

Fig. 11 shows the result of the simulation. Notice that the index of the left graph goes from 0 to 1 while the right graph goes from 1 to 0.

4.4 Three pipe problem

Three pipe problem is a very interesting application for boundary condition testing. The connection of three pipes at a point is challenging because of the mass balance in an infinitesimal volume at the connector.

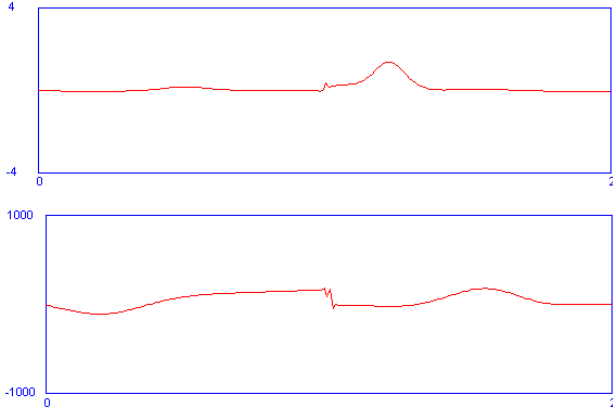


Figure 9: Difference between single pipe and two series pipe.

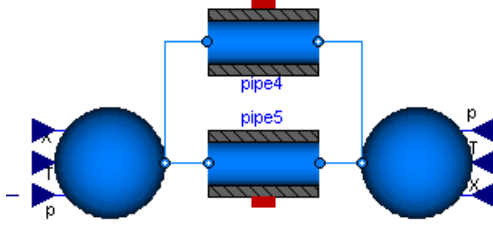


Figure 10: Model used for symmetry testing.

In this particular application, the wave interference can be viewed for inviscid or low viscosity fluids, if the length of the pipes are different. In fig. 12 is depicted the system modeled. Fig. 13 shows the pressure and temperature profiles when the traveling wave bounces in the short pipe. We observe in the larger pipe how the shock pressure wave is still traveling farther.

The profiles after the shock wave bounces at the end of the larger pipe is presented in Fig. 14.

5 Implementation details

5.1 Structure

The structure of the Modelica code follows the same dependency shown before. The parts of the code can be identified with the following diagram (from reconstruction to derivative)

$$\bar{u}_i \xrightarrow{\log} u_{i+1/2}^+, u_{i+1/2}^- \xrightarrow{W^+, W^-} \hat{f}_{i+1/2} \rightarrow \frac{d}{dt} \bar{u}_i$$

The function `RoeWaves` computes the traveling waves W^+, W^- from the reconstruction scheme. The function `logarithmicminus` computes the approximation $u_{i+1/2}^-$ from the averages u_i and its companion `logarithmicplus` computes $u_{i+1/2}^+$ from the averages u_i , see listing 5.

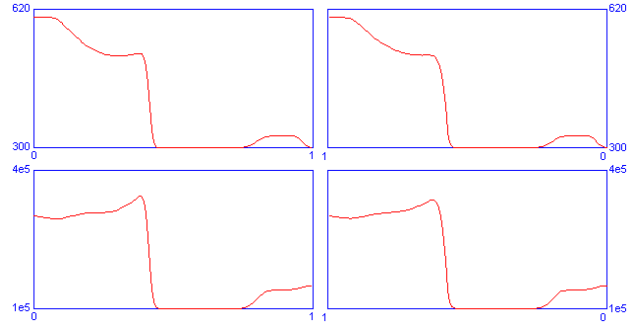


Figure 11: Solution of the symmetry test.

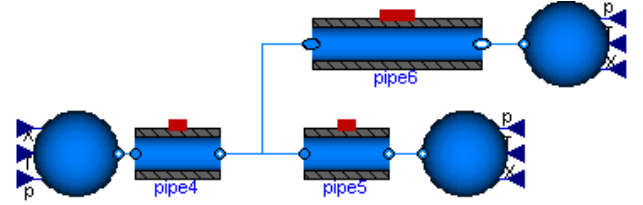


Figure 12: Three pipe model for testing of boundary conditions for several connections.

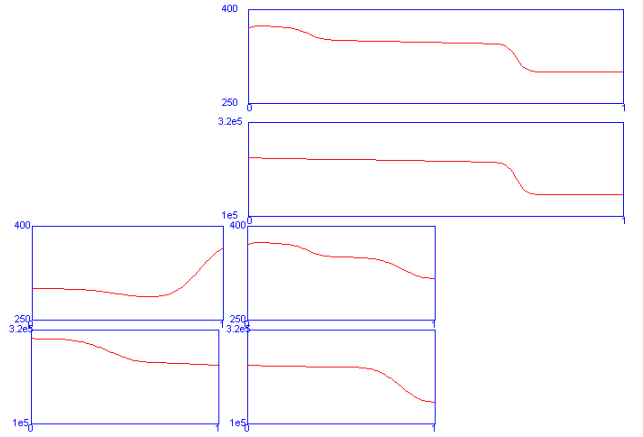


Figure 13: Solution after the traveling wave in the short pipe bounces.

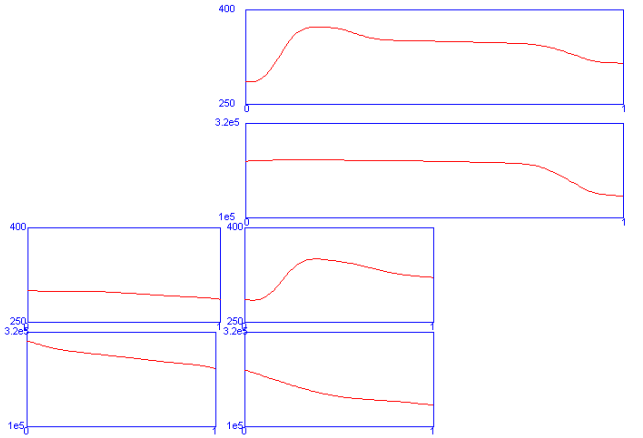


Figure 14: Solution after the traveling wave bounces at the large pipe.

At last, the approximations contained in the arrays `f1p`, `f2p`, `f3p` and `f1m`, `f2m`, `f3m`, that are equated to the derivatives $\frac{d}{dt}\bar{u}_i$, see listing 6 and 7

Listing 5: Roe's Flux with logarithmic reconstruction

```

(f1p[i], f2p[i], f3p[i], f1m[i], f2m[i], f3m[i]) =
  RocWaves(
    {logarithmicminus(medium[i+1].d,
                      medium[i].d,
                      medium[i-1].d, length/n,q),
     logarithmicminus(medium[i+1].d*v[i+1],
                      medium[i].d*v[i],
                      medium[i-1].d*v[i-1],
                      length/n,q),
     logarithmicminus(medium[i+1].d*medium[i+1].u + medium[i+1].d*(v[i+1])^2/2,
                      medium[i].d*medium[i].u + medium[i].d*(v[i])^2/2,
                      medium[i-1].d*medium[i-1].u + medium[i-1].d*(v[i-1])^2/2,
                      length/n,q)},
    {logarithmicplus(medium[i+2].d,
                     medium[i+1].d,
                     medium[i].d, length/n,q),
     logarithmicplus(medium[i+2].d*v[i+2],
                     medium[i+1].d*v[i+1],
                     medium[i].d*v[i],
                     length/n,q),
     logarithmicplus(medium[i+2].d*medium[i+2].u + medium[i+2].d*(v[i+2])^2/2,
                     medium[i+1].d*medium[i+1].u + medium[i+1].d*(v[i+1])^2/2,
                     medium[i].d*medium[i].u + medium[i].d*(v[i])^2/2,
                     length/n,q)},
    1.4);

```

Listing 6: Mass Balance with Roe's Flux

```
der(m[i]) = - (f1p[i-1] + f1m[i]) * A_inner + ms_flow[i];
der(mXi[i, :]) = mXi_flow[i-1, :] - mXi_flow[i, :] + msXi_flow[i, :];
der(U[i]) = - (f3p[i-1] + f3m[i]) * A_inner + Qs_flow[i];
```

Listing 7: Momentum Balance with Roe's Flux

```
(if dynamicTerm then
    der(m_flow[i])*length/n else 0) = -(f2m[i]+f2p[i-1])*A_inner
    + F_f[i] + (I_flow[i-1]-I_flow[i+1])/2;
```

5.2 Including connectors and boundary conditions

The ghost cells $u_{-1} = u_0$ are implemented in `medium_a`, at the right boundary. At the left boundary, the ghost cells $u_{n+1} = u_{n+2}$ are implemented in `medium_b`. Then, they are connected to the discretisation using the `semiLinear` operator and adequate relations for `m_flow` variables, see listing 8.

Listing 8: Ports and Ghost cells

```
//Port medium models for ghost cells
port.a.p = medium.a.p;
port.b.p = medium.b.p;
port.a.h = medium.a.h;
port.b.h = medium.b.h;
port.a.Xi = medium.a.Xi;
port.b.Xi = medium.b.Xi;

// Boundary conditions
port.a.H.flow = semiLinear(port.a.m.flow, port.a.h, medium[1].h);
port.b.H.flow = semiLinear(port.b.m.flow, port.b.h, medium[n].h);
port.a.mXi.flow = semiLinear(port.a.m.flow, port.a.Xi, medium[1].Xi);
port.b.mXi.flow = semiLinear(port.b.m.flow, port.b.Xi, medium[n].Xi);
port.a.m.flow = m.flow[1];
port.b.m.flow = -m.flow[n];
```

The traveling waves at $x = 1 - 1/2$ are called f1p_a, f2p_a, f3p_a, f1m_a, f2m_a, f3m_a and those at $x = n + 1/2$ are called f1p_b, f2p_b, f3p_b, f1m_b, f2m_b, f3m_b. Their implementation of their computation is in listing 5. Their incorporation into the mass, momentum and energy balance is in listing 10.

Listing 9: Connectors in Roe approximation

```
...
// port_a
(f1p-a, f2p-a, f3p-a, f1m-a, f2m-a, f3m-a) =
```

```

RoeWaves(
{logarithmicminus( medium[1].d,
                    medium.a.d,
                    medium.a.d,
                    length/n,q),
logarithmicminus( medium[1].d*v[1],
                    medium.a.d*v[1],
                    medium.a.d*v[1],
                    length/n,q),
logarithmicminus( medium[1].d*medium[1].u + medium[1].d*(v[1])^2/2,
                    medium.a.d*medium.a.u + medium.a.d*(v[1])^2/2,
                    medium.a.d*medium.a.u + medium.a.d*(v[1])^2/2,
                    length/n,q)},
{logarithmicplus( medium[2].d,
                    medium[1].d,
                    medium.a.d, length/n,q),
logarithmicplus( medium[2].d*v[2],
                    medium[1].d*v[1],
                    medium.a.d*v[1],
                    length/n,q),
logarithmicplus( medium[2].d*medium[2].u + medium[2].d*(v[2])^2/2,
                    medium[1].d*medium[1].u + medium[1].d*(v[1])^2/2,
                    medium.a.d*medium.a.u + medium.a.d*(v[1])^2/2,
                    length/n,q)},
1.4);
...
// port_b
(f1p_b, f2p_b, f3p_b, f1m_b, f2m_b, f3m_b) =
RoeWaves(
{logarithmicminus( medium.b.d,
                    medium[n].d,
                    medium[n-1].d,
                    length/n,q),
logarithmicminus( medium.b.d*v[n],
                    medium[n].d*v[n],
                    medium[n-1].d*v[n-1],
                    length/n,q),
logarithmicminus( medium.b.d*medium.b.u + medium.b.d*(v[n])^2/2,
                    medium[n].d*medium[n].u + medium[n].d*(v[n])^2/2,
                    medium[n-1].d*medium[n-1].u + medium[n-1].d*(v[n-1])^2/2,
                    length/n,q)},
{logarithmicplus( medium.b.d,
                    medium.b.d,
                    medium[n].d,
                    length/n,q),
logarithmicplus( medium.b.d*v[n],
                    medium.b.d*v[n],
                    medium[n].d*v[n],
                    length/n,q),
logarithmicplus( medium.b.d*medium.b.u + medium.b.d*(v[n])^2/2,
                    medium.b.d*medium.b.u + medium.b.d*(v[n])^2/2,
                    medium[n].d*medium[n].u + medium[n].d*(v[n])^2/2,
                    length/n,q)},
1.4);

```

Listing 10: Boundary Conditions

```

// Mass and Energy Balance
...
// side a
if static then
...
else
    der(m[1]) = - (f1p.a+f1m[1]) *A_inner + ms_flow[1];
    der(mXi[1, :]) = mXi_flow[1, :] - mXi_flow[2, :] + msXi_flow[1, :];
    der(U[1]) = - (f3p.a+f3m[1]) *A_inner + Qs_flow[1];
end if;
...
//side b
if static then
...
else
    der(m[n]) = - (f1p[n-1] + f1m.b)*A_inner + ms_flow[n];
    der(mXi[n, :]) = mXi_flow[n, :] - mXi_flow[n + 1, :] + msXi_flow[n, :];
    der(U[n]) = - (f3p[n-1] + f3m.b) *A_inner + Qs_flow[n];
end if;
...
//Momentum Balance
if lumped_dp then
...
else
    (if dynamicTerm then der(m_flow[1])*length/n/2 else 0) =
        -f2m[1]+f2p.a)*A_inner + F.f[1] + (I_flow[1]-I_flow[2])/2;
    ...
    (if dynamicTerm then der(m_flow[n])*length/n else 0) =
        (f2m.b+f2p[n-1])*A_inner + F.f[n] + (I_flow[n-1]-I_flow[n+1])/2;
    ...
end if;

```

Finally, the numerical scheme is incorporated to DistributedPipeFV by extending from `Flow1D_FV_Log` instead of `Flow1D_FV`.

Listing 11: LDLR in DistributedPipeFV

```

model DistributedPipeFV "Distributed_pipe_model_with_optional_wall"
extends FlowID.FV.Log(
  Qs.flow=heat.Q_flow,
  ms.flow=zeros(n),
  msXi.flow=zeros(n, Medium.nXi));
...
end DistributedPipeFV;

```

6 Conclusions and Future Work

We have shown how the logarithmic reconstruction and appropriate upwinding captures shock waves without spurious oscillations. The numerical performance of these FV schemes is not more complicated than the originals used in Modelica.Fluid library, in terms of implementation and are more reliable. The CPU time observed was similar.

Some further investigations have to be done for time integration. The control of the Courant-Friedrichs-Lax number is critical for an even better capture of shock waves. This CFL number has to be controlled using special numerical integration schemes. Special Runge-Kutta methods have been developed and investigated, with successful results, in [3].

References

- [1] Elmqvist, H., Tummescheit, H. and Otter, M. *Object-Oriented Modeling of Thermo-Fluid Systems* Proceedings of Modelica'2003 conference. Linköping, Sweden. pp.269
- [2] Gallouët, T. and Hérard, J.-M. and Seguin, N. *Some recent finite volume schemes to compute Euler equations using real gas EOS* Internat. J. Numer. Methods Fluids, 2002, pp. 1073–1138.
- [3] Gottlieb S., Shu C. and Tadmor E. *Strong stability-preserving high-order time discretization methods* SIAM Rev., 43, 2001:1 pp. 89-112.
- [4] LeVeque, R., *Numerical Methods for Conservation Laws*, Birkhauser-Verlag, 1990
- [5] Marquina, Antonio. *Local piecewise hyperbolic reconstruction of numerical fluxes for nonlinear scalar conservation laws*, SIAM J. Sci. Comput., 15:4, 1994, pp. 892–915
- [6] Patankar, S. *Numerical heat Transfer and Fluid Flow*, Series in Computational Methods in Mechanics and Thermal Sciences, Hemisphere Publishing Corporation 1980, USA.
- [7] Artebrant, Robert and Schroll, H. Joachim. *Limiters-free Third Order Logarithmic Reconstruction* To appear.
- [8] Artebrant, Robert and Schroll, Hans Joachim. *High-resolution Riemann-solver-free methods for conservation laws* Hyperbolic problems: theory, numerics, applications, Springer Verlag 2003, pp. 305–314
- [9] Schroll, H. Joachim. *Relaxed high resolution schemes for hyperbolic conservation laws*, J. Sci. Comput., vol. 21:2, 2004, pp. 251–279
- [10] Hall, Oskar and Schroll, Hans Joachim and Svensson, Fredrik. *High-resolution simulation of inviscid flow in general domains*, Internat. J. Numer. Methods Fluids, 47:10-11, pp. 1061–1067
- [11] Serna, Susana and Marquina, Antonio. *Power ENO methods: a fifth-order accurate weighted power ENO method*, J. Comput. Phys., 194:2, pp. 632–658
- [12] Winterbone, D., Pearson, R., *Theory of engine manifold design*, Professional Engineering Publishing, 2000

Modeling of an Experimental Batch Plant with Modelica

Katja Poschlad¹, Manuel A. Pereira Remelhe¹, Martin Otter²

¹University of Dortmund, Process Control Laboratory (AST), Germany

²German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany

Abstract

A Modelica model of an experimental batch plant installed at the Process Control Laboratory (AST) of the University of Dortmund was developed. The plant model consists of tanks, pipes, valves, pumps and a control system to mix, buffer, heat, cool and evaporate a mixture of water and sodium chloride. Details of the most important components, of the plant model and of the sequence control implemented with the Modelica StateGraph library are discussed.

1 Introduction

The Modelica_Fluid library [4] is a free library to be included in the Modelica standard library. It is supposed to be usable for several application areas. Based on this library, a model of a laboratory batch plant was developed with the following goals:

- to further develop the Modelica_Fluid library,
- to demonstrate the ability to model and simulate batch plants with Modelica,
- to have a free, non-trivial, controlled plant model to test and verify new modeling ideas and algorithms,
- to use it as benchmark problem for hybrid control systems.

To achieve these goals and in order to be able to implement the batch plant, new components have been developed such as a much more generic tank model, a model for cooling and heating tanks, an evaporator, a condenser, and several simple components such as a non-horizontal pipe and a controlled valve model. Furthermore, a model for a medium system consisting of water and sodium chloride has been constructed. In this article an overview of this project is given. More details can be found in [5].

2 Description of the Batch Plant

The process under consideration is an evaporation plant for a student lab at the Process Control Laboratory (AST) of the University of Dortmund that evaporates a water sodium chloride mixture so that a higher concentrated solution is produced [1]. The task of the students is to learn how to program the process control system. A picture of the batch plant is shown in figure 1. The flow sheet diagram is shown in figure 2.



Figure 1: Picture of AST batch plant.

Pure water from tank B1 and concentrated sodium chloride solution from tank B2 are mixed in a mixing tank B3. After buffering in tank B4 the mixture flows to the evaporator B5. Here the water sodium chloride mixture is evaporated until the desired concentration is reached. The steam is condensed in the condenser K1 and cooled afterwards in the cooling tank B6. The concentrated solution is also led to a cooling tank B7. The cooled fluids are pumped back to the charging vessels by the pumps P1 and P2. Between the tanks several valves are present that are regulated by a central control system.

3 Main Fluid Components

The plant is modeled with components of the Modelica_Fluid library, such as pipe and pump models [4]. Several fluid flow components had to be extended or newly developed and implemented, in order to model and simulate this batch plant.

3.1 Generic tank model

A generic tank model was newly implemented that describes a tank which is open to the environment at fixed ambient pressure. Heat transfer to the environment and to the tank walls is neglected. The tank is

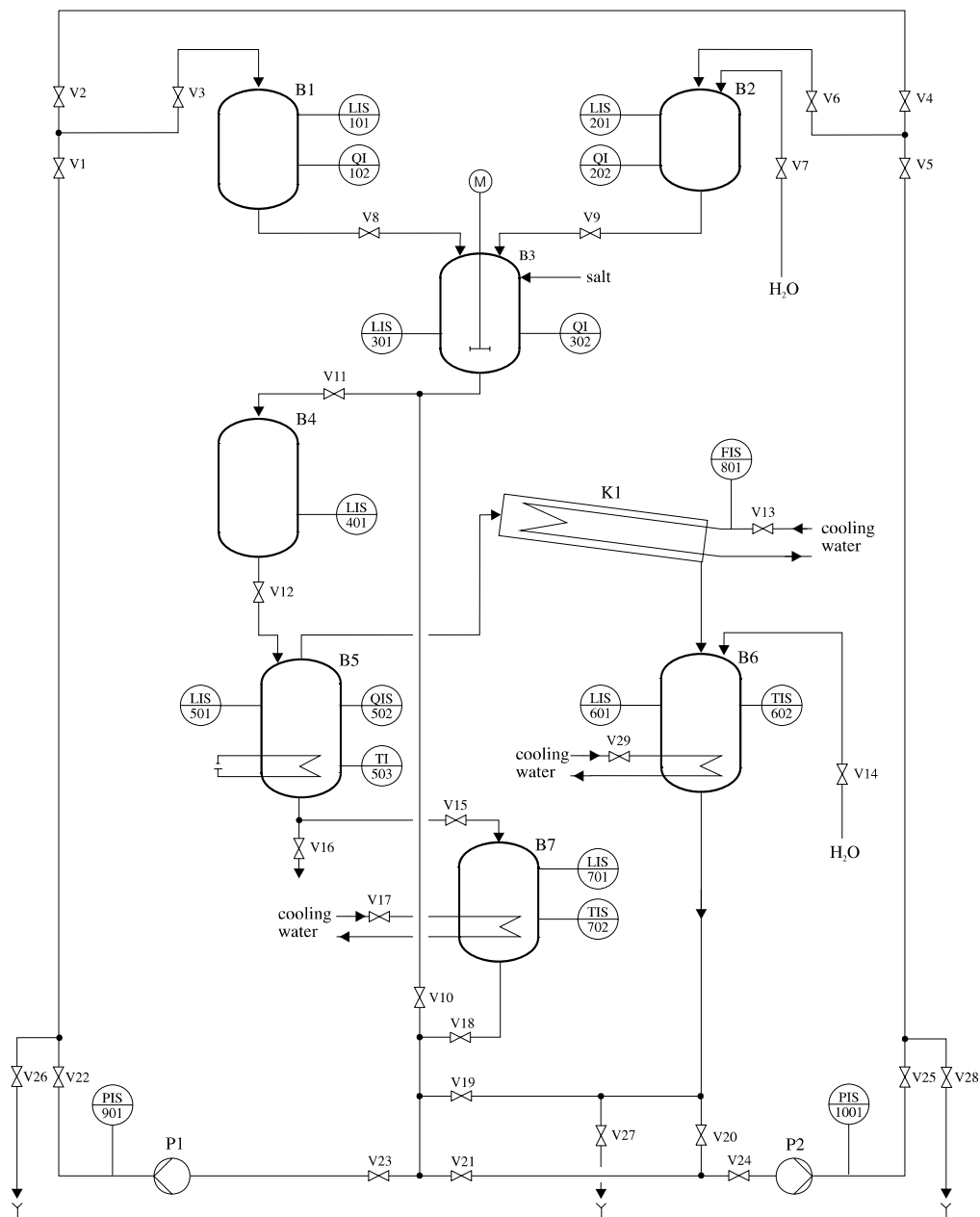


Figure 2: Flow sheet diagram of AST batch plant.

filled with a single or multiple-substance liquid, assumed to have uniform temperature and mass fractions.

The tanks in the AST batch plant, as well as most batch plants in industry, have more than one inlet/outlet at different heights. For this reason, the “Tank” model has a vector of FluidPort connectors at the top and at the bottom of the tank icon, as seen in the screenshot of the tank icon in figure 3.

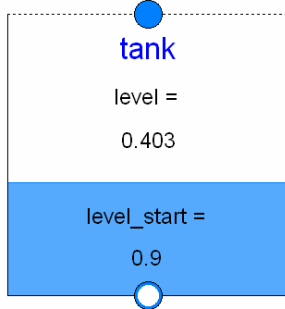


Figure 3: Screen shot of the tank icon

The fill level is dynamically visualized with the blue square and also the actual value is given (in figure 3: “level = 0.403”). This is implemented in the icon annotations by function calls provided from the Beta-release of the UserInteraction library.

The top connectors “topPorts[:]” are assumed to be always above the overflow level. Therefore, fluid from the tank can never flow into them.

The connectors “ports[:]” at the bottom of the icon are either attached at the bottom or at the side of the tank and fluid can flow in both directions.

For the top connectors it is sufficient to know the number of connectors, since the pressure at the connectors is always identical to the ambient pressure. The bottom and side connectors are defined by (hydraulic) inner diameters of the inlet/outlet pipe and at which position “portLevel” the connector is present.

The parameter menu of the tank is shown in figure 4.

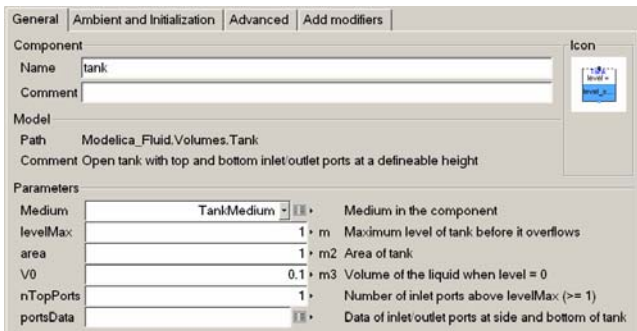


Figure 4: Tank parameter menu

Parameter “levelMax” defines the maximum level until the tank overflows. Currently, this triggers just an assert. Parameter “portsData” is an array of records that defines diameter and port level of the bot-

tom and side connectors. An example is shown in figure 5 below.

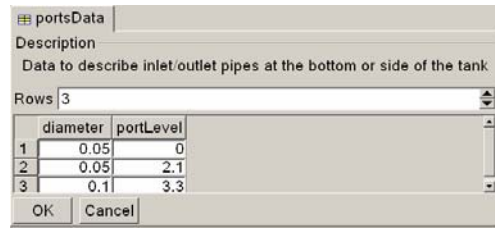


Figure 5: Parameter menu to define tank ports

The tab “Ambient and Initialization”, see figure 4, defines the ambient and initial conditions.

Due to the equation based nature of Modelica, the implementation of the *basic* tank equations is straightforward:

```

model tank
  import IF = Modelica_Fluid.Interfaces;
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium;
  IF.FluidPort_a topPorts[:](
    redeclare package Medium = Medium);
  IF.FluidPort_b ports[:](
    redeclare package Medium = Medium);
  Medium.BaseProperties medium(..);
  ...
  // Total quantities
  medium.p = p_ambient;
  V = area*level + V0 "Volume of fluid";
  m = V*medium.d "Mass of fluid";
  mXi = m*medium.Xi "Mass of fluid comp.";
  U = m*medium.u "Internal energy";

  // Mass balances
  der(m) = sum(topPorts.m_flow) +
    sum(ports.m_flow);
  for i in 1:Medium.nXi loop
    der(mXi[i]) = sum(topPorts.mXi_flow[i])
      + sum(ports.mXi_flow[i]);
  end for;

  // Energy balance
  der(U) = sum(topPorts.H_flow) +
    sum(ports.H_flow) -
    p_ambient*der(V)
  ...
end tank;

```

If the fluid is incompressible, the term “ $p_{\text{ambient}} \cdot \text{der}(V)$ ” is removed from the energy balance. The reason is that otherwise unphysical small temperature changes occur, if the tank level changes. By removing this term, mechanical work is neglected since it is also neglected in the medium model.

As usual in the Modelica_Fluid library (for details, see [4]), mass flow rate and enthalpy flow rate at the ports are computed with the semiLinear(..) operator

in order to describe the potential bidirectional flow of the fluid, e.g.,

```
ports[i].H_flow = semiLinear(
  ports[i].m_flow, ports[i].h, medium.h);
```

Finally an equation for the port pressures has to be provided. For the topPorts, this is simple, because they are by definition always above the fluid level and therefore $\text{topPorts.p}[i] = p_{\text{ambient}}$.

If a bottom or side port $\text{ports}[i]$ is below the actual fluid level, the Bernoulli equation can be used to compute the port pressure (it is assumed that the tank has a uniform pressure, temperature and density; the Bernoulli equation holds also in cases where the density is varying with time, provided it has the same value along the path under consideration):

If the fluid flows out of the tank, it is applied from the tank level to the corresponding port. In this situation no significant pressure losses occur, with exception of a potential small one that depends on the shape of the outlet pipe.

If the fluid flows into the tank, the whole kinetic energy is dissipated and does not lead to a pressure increase. Taken this into consideration the basic equation is given as:

```
ports[i].p = p_ambient +
  (level - portLevel[i])*g*medium.d -
  if ports[i].m_flow < 0 then
    ports[i].m_flow^2/
      (2*medium.d*area[i]^2)
  else 0;
```

Severe difficulties occur if the fluid level drops below an inlet/outlet port and therefore the corresponding pipe might no longer contain fluid or is filled only partially with fluid. An example is shown in figure 6:

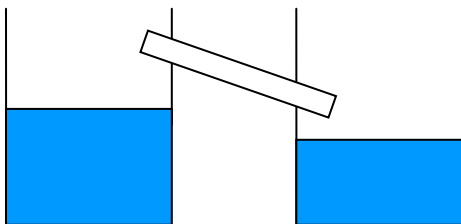


Figure 6: Inlet/outlet pipe over the tank levels

A detailed description of this situation would require modeling the mixture of liquid and air and taking into account that the mixture is not homogenous in a pipe. Since this would complicate the modeling of tanks, pipes and other components considerably, different approximate solutions have been evaluated and verified by a suite of test models.

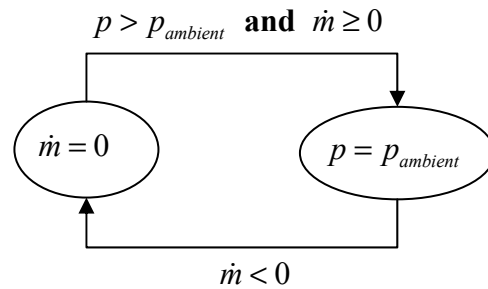
In the tank, there are now two solutions implemented that can be selected in the “Advanced” menu. The

default solution operates in the following way when the fluid level is below a port:

A large pressure loss factor (default: $\zeta_{\text{out}} = 10^5$), is used when the fluid flows out of the port and a small pressure loss factor (default: $\zeta_{\text{in}} = 0.01$) is used when the fluid flows into the port. The quadratic characteristic around zero mass flow rate is regularized with the Modelica_Fluid.Utilities.regSquare2(...) function, in order to guarantee that at zero mass flow rate the derivative of the pressure drop equation is neither zero nor infinity. Note, when the level is above the pipe port, we have $\zeta_{\text{out}} = 1$ and $\zeta_{\text{in}} = 0$. Numerically, this means that the pressure drop equation of the Bernoulli equation is a strict monotonically rising characteristic so that a non-linear solver can handle system equations that contain such an equation, if the step size is selected small enough. Physically, this means that a small leakage mass flow rate appears that depends essentially on the value of ζ_{out} . All test models work very reliably with this solution, see, e.g., the models in Modelica_Fluid.Examples.Tanks.

Alternatively, a second solution can be selected that is based on the following considerations:

If the level is below the port and fluid flows from the port into the tank, the port pressure is identical to the ambient pressure. If the fluid tries to flow out of the tank, the mass flow rate is explicitly set to zero. The solution is now to switch between these two equations using the following simple state machine:



This state machine is implemented in Modelica as:

```
m_flow_out = pre (m_flow_out) and
  not port.p > p_ambient or
  port.m_flow < -1e-6;
0 = if pre(m_flow_out) then m_flow
  else p-p_ambient;
```

This solution has the advantage that no leakage flow occurs and it works reliably in many situations. It may fail in cases where a non-linear system of equations is no longer well defined, e.g., due to the equation $m_{\text{flow}} = 0$ in one branch of the if statement (it fails for example in the complicated situation of the example Modelica_Fluid.Examples.Tanks.Tanks.WithEmptyingPipe2).

A third solution variant would be to take the same basic equations as above, but use a declarative formulation to switch between the two equations based on a parameterized curve description (similarly to the description of ideal electrical switches and to friction in the Modelica standard library). This yields the equations:

```
m_flow_out = s < 0;
port.m_flow = if m_flow_out then 0
               else s;
port.p-p_ambient = if m_flow_out then s
                   else 0;
```

Numerical experiments with the test cases show that this third variant is not reliable and often fails. The reason is that this formulation leads to *non-linear mixed* systems of equations having both Real and Boolean variables (m_flow_out) as unknowns and the algorithms in Dymola seem to not work well in such a situation. Contrary, the parameterized curve descriptions of, e.g., electrical switches, in the Modelica standard library lead to *linear mixed* systems of equations that are solved very well in Dymola.

We have now two sets of equations, one that is used if the level is above the port level and one when it is below. The question arises, how to switch between these two equation sets. In Modelica one could use a simple if clause:

```
if level > portLevel[i] then
  // Bernoulli equation
else
  // equation to handle empty port
end if;
```

since the relation will automatically trigger a state event and therefore the switching is performed numerically in a reliable way.

However, in some situations, especially, when a tank is empty and all fluid that flows into the tank is at once flowing out at the bottom port, a numerical solver will switch extremely often between the two branches of the if clause, leading to a very small step size so that the simulation is practically stopped, i.e., chattering occurs.

Chattering can always be reduced by introducing an appropriate hysteresis. For a tank this is rather straightforward and also makes physically sense, because the exact time instant when to switch between the two equations is not well defined: A port at the side of the tank has a certain diameter. When the level is above the upper part of the pipe, there is clearly fluid in the port. If the level is below the lower part of the pipe, there is clearly no fluid in the port, but in between the situation is not well defined.

In the tank model the hysteresis is formulated in the following way, using the fraction k of the port diameter as hysteresis distance:

```
aboveLevel[i] =
  level >= (portLevel[i] + diameter[i]/k)
  or pre(aboveLevel[i]) and
  level >= (portLevel[i] - diameter[i]/k)
levelAbovePort[i] = if aboveLevel[i] then
                    level - portLevel[i] else 0;
```

Example `Modelica_Fluid.Examples.Tanks.OneTank` demonstrates the behavior. See the model setup in figure 7. It consists of a tank where a constant mass flow rate is entering at the top inlet and the fluid flows out at the bottom port with a higher mass flow rate.

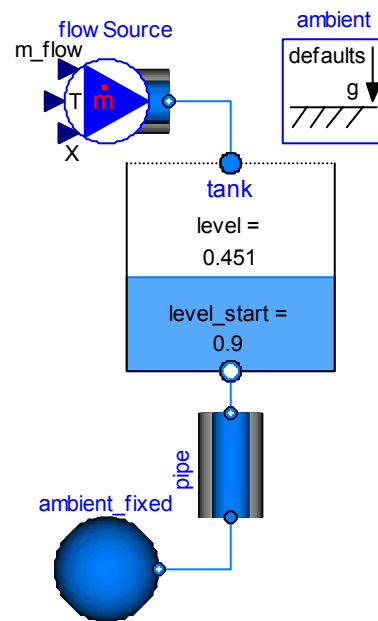


Figure 7: Fluid flows faster out of the tank bottom outlet as it flows in by the top inlet leading to chattering.

A plot of the level is shown in figure 8. The chattering after about 50 s when the tank is empty is clearly visible. Due to the introduced hysteresis, the chattering influences the simulation speed only marginally.

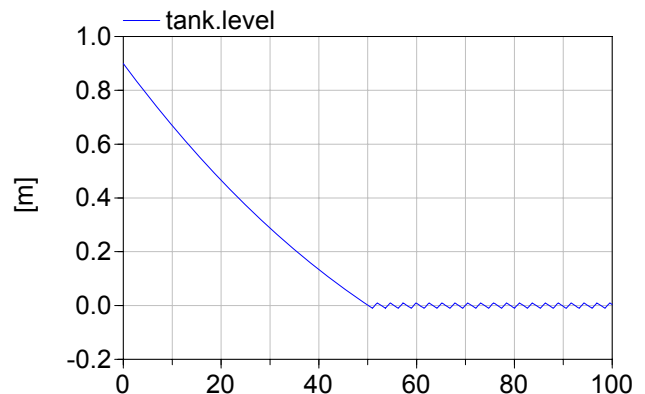


Figure 8: Level of tank from figure 7 over time.

3.2 Evaporator and condenser model

The experimental batch plant contains an evaporator and a condenser, which have to be modeled. In order to simplify the task, the following simplifications are made:

- evaporator and condenser are modeled in one component.
- condensed water is saturated at the end of the condenser.
- no hold up in the condenser.

The evaporator/condenser model is an extension of the generic tank model. A heat port is added to the tank model which extends the heat balance. Furthermore a single fluid port is added which symbolizes the end of the condenser. The icon of the model is giving in 9.

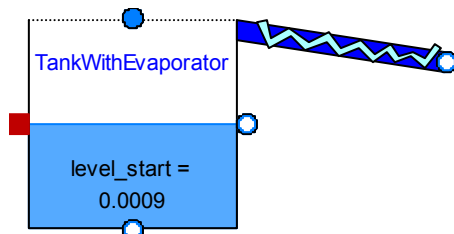


Figure 9: Screen shot of evaporator/condenser model.

The energy balance of the generic tank is changed to

```
der(U) = sum(H_flow_bottomPorts) +
          sum(H_flow_sidePorts) +
          sum(H_flow_topPorts) +
          heatPort.Q_flow +
          Condensed.m_flow*hv -
          p_ambient*der(V);
```

with the enthalpy of the saturated vapor h_v . Because the assumption is made that the condensed water is saturated, the energy from the cooling water in the condenser is equal to the condensing energy. So only the evaporator needs to be balanced.

The mass flow rate of the condensed water m_{con} for a binary system is calculated by

```
m_con = if p > p_sat then 0 else
          - heatPort.Q_flow /
          (hv-hl+Xi(1)*(cp*dT_dX+dh1_dX));
```

with the supplied heat $heatPort.Q_flow$, the specific enthalpy of the saturated liquid h_l , the saturated vapour h_v , the heat capacity c_p , the partial derivative of the boiling temperature with respect to the mass fraction dT_dX , and the partial derivative of the specific enthalpy h_l with respect to the mass fraction dh_l_dX . This calculation assumes that only water evaporates and that all the energy is used for evaporation, once the saturation pressure p_{sat} is reached. This can lead to chattering if cold fluid flows into the

tank. To avoid the chattering the variable Q_{up} is introduced, that gives the amount of energy used to rise the temperature in the tank up to the boiling point. These considerations lead to the following formulation that avoids chattering:

```
if Q_up > heatPort.Q_flow then
  m_con = 0;
else
  m_con = -(heatPort.Q_flow - Q_up) /
           (hv-hl+Xi(1)*(cp*dT_dX+dh1_dX));
end if;
```

3.3 Other Component Models

Additionally, some simpler component models have been implemented, especially:

- Model class **Ambient** to define default ambient conditions (such as default ambient pressure) with one inner ambient instance and access the data via inner/outer from other elements.
- Model class **ValveDiscrete** to open and close a simple valve model by a Boolean input signal.
- Model class **WallFriction** is renamed to **WallFrictionAndGravity** and an additional term $\Delta p = height_ab * d * g$ for the calculation of the pressure drop due to gravity is added, given the relative height $height_ab$ of port_b over port_a.

4 A medium model for mixture of water and sodium-chloride

To model the batch plant it is necessary to construct a model for the system water-sodium chloride. This type of medium is not yet available in the Modelica.Media library and therefore a superclass for two phase models of a mixture was introduced. The subclass relationship of this extension is displayed in figure 10.

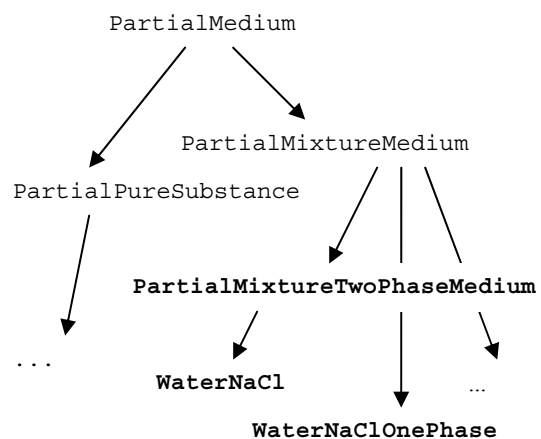


Figure 10: Hierarchical structure of Modelica.Media

For the model of the batch plant the following fluid characteristics are important and therefore included in the medium model:

The density ρ , the specific enthalpy h and the dynamic viscosity η are described as a function of temperature T , pressure p , and composition x_i . To model the evaporation, the saturation pressure p_{sat} is needed which is a function of temperature T and composition x_i .

For the implementation of the medium model the interpolation of Chou [2] is used, as well as the data for the dynamic viscosity of DECHEMA [3]. The resulting terms are as follows:

Density ρ is calculated by

$$\frac{1}{\rho(T, w, p)} = F_1(T) - p \cdot F_2(T) - p^2 \cdot F_3(T) \\ + w \cdot F_4(T) + w^2 \cdot F_5(T) - w \cdot p \cdot F_6(T) \\ - w^2 \cdot p \cdot F_7(T) - \frac{1}{2} w \cdot p^2 \cdot F_8(T)$$

with the pressure p , the temperature T , the mass fraction of sodium chloride w and the terms $F_1 - F_8$ which are provided in the paper of Chou [2].

The specific enthalpy h is calculated by

$$h = h_0 + \int_{T_0}^T c_p \cdot dT$$

where the specific enthalpy h_0 under standard condition is defined by

$$h_0(w) = E_1 \cdot (1 - w) + E_2 \cdot w^{1.5} \\ + E_3 \cdot w^2 + E_4 \cdot w^{2.5} + E_5 \cdot w^3$$

with the mass fraction w and the parameters $E_1 - E_5$ [2]. The heat capacity c_p is calculated by

$$c_p(x, T) = C_1(x) - C_2(x) \cdot T + C_3(x) \cdot T^2$$

with the mole fraction of sodium chloride x , and the terms $C_1 - C_3$ [2].

To calculate the saturation pressure p_s the following equation is used:

$$\ln(p_s(x, T)) = (1 - G_1 \cdot x + G_2 \cdot x^2) \cdot \\ \left(G_3 - \frac{G_4}{T} - G_5 \cdot \ln(T) + G_6 \cdot T \right) \\ - x \cdot (G_7 - G_8 \cdot x + G_9 \cdot x^2)$$

with parameters $G_1 - G_9$ from [2].

The viscosity η , which is needed for the calculation of the pressure drop within a pipe, is calculated by

$$\eta(T, w) = \exp$$

$$(c_7 + c_1 \cdot T^3 + c_2 \cdot T^2 + c_3 \cdot T + c_4 \cdot w^3 + c_5 \cdot w^2 + c_6 \cdot w)$$

with the parameters

$$c_1 = -6.83241E-07 \frac{1}{K^3} \\ c_2 = 0.000765676 \frac{1}{K^2} \\ c_3 = -0.297018665 \frac{1}{K} \\ c_4 = -0.299730079 \\ c_5 = 2.065267182 \\ c_6 = 1.546491257 \\ c_7 = 31.57571595$$

For the vapor-liquid-equilibrium also the specific enthalpy of the water vapor is needed. This is taken from the medium model `StandardWater`.

5 Controller

The controller that regulates the valves, the heating and the cooling is modeled with the Modelica.StateGraph library, i.e., using basically a “sequential function chart” with additional equations. The flow chart with steps and transitions is shown in figure 11. The left triangle is an input connector containing all sensor signals whereas the triangle on the right side is an output connector containing all actuator signals.

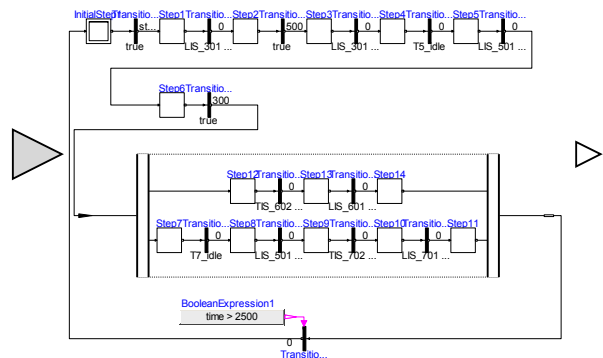


Figure 11: StateGraph model of the controller

In the first step the valve V8 is open so that liquid flows from tank B1 to tank B3 till a minimum level of 0,13 m is reached in B3 (Transition2). In step2 valve V9 opens till a desired concentration is measured in tank B3 (Transition3). Step3 leads the fluid to tank B4 by opening valve V11 till the level in tank B3 is below 0,01m. In Step4 T5_idle shows whether tank B5 can be filled.

```
T5_idle =
  LIS_501 < 0.01 and not V15.open;
```

If Step5 is active, valve V12 opens which leads the liquid to tank B5 till the level T5_batch_level is reached in tank B5. Step6 starts the heating in tank B5 till the desired concentration is reached. Since the evaporated water and the concentrated solution are split up, the controlling has to be split as well which is realized by parallel branches. The upper branch is for the condensed water. Step12 starts the cooling in tank B6 till the temperature is below 20°C. After that the valves V20, V24, V25, V5 and V6 have to be opened so that the pump P2 can pump the water to tank B2. The lower branch controls the concentrated solution. If tank B7 can be filled (T7_idle = true), in Step8 the valve V15 is open till the tank B5 is empty. Then the cooling in tank B7 is switched on till the temperature is low enough. To pump the liquid back to tank B1 with pump P1 the valves V18, V23, V22, V1 and V3 have to be opened. Once both branches are finished, and time > 2500 s, the cycle starts from the beginning.

The controller input connector “sensors” contains all measured values such as the level of a tank, the temperature in a tank or the concentration. The measured values are named in analogue to their names in the flow sheet of the batch plant (see figure 2). e.g.

```
sensors.LIS_301 = B3.level;
```

The Boolean outputs are written in the output connector “actuators”. E.g.

```
actuator.V9 = Step2.active;
```

where Step2.active becomes true when Step1 has been active and the transition Transistion2

```
LIS_301 >= 0.13;
```

becomes true.

6 Model of the Batch Plant

With the new components, the new media model, the Modelica_Fluid and the Modelica.StateGraph library, a model of the batch plant can be assembled. The tanks B1, B2, B3 and B4 are modeled with the generic tank model described in chapter 3.1. The tank B5 is modeled with the model for an evaporator/condenser as explained in chapter 3.2. The tanks B6 and B7 are modeled by an extended generic tank model that includes a heat port to model the cooling. The pumps are modeled with the pump model from the Modelica_Fluid library [4]. To model the pressure drops and the variable heights of the tanks the pipes are modeled with the model of a non-

horizontal pipe. The valves are modeled such that they are either open or closed. They are controlled by the Boolean input signal open to indicate if the valve is open. Two models have been setup: One using the StandardWater medium model, i.e., modeling only two phase water flow and one model that uses the medium model for the water–sodium chloride mixture which has been discussed in chapter 4. The controller sketched in chapter 5 is used to model the controller of the batch plant. The Modelica model of the batch plant is shown in figure 12.

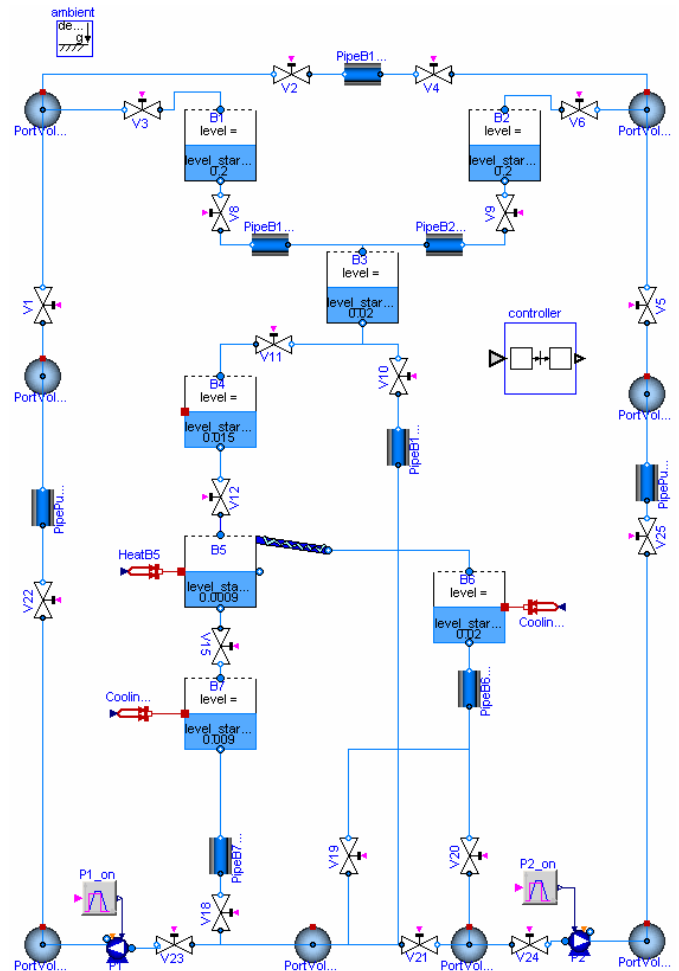


figure 12: Modelica model of the AST batch plant

Modeling this system by pressure drop components only is not possible, since otherwise equations for pressure, composition and specific enthalpy at connectors are missing when two or more connected valves are closed. For this reason, appropriate volumes are introduced at connection points.

The connections from the central controller to the sensors and actuators are not performed graphically because too many signals would need to be connected. Instead, the connections are stated in the textual level with equations: Each valve gets its input if it is open or closed from the controller by e.g.

```
V1.open = controller.sensors.V1;
```

and equally for the other valves. The variables for heating and cooling are Real and not Boolean variables, so the equation looks different to those for the valves. This is shown exemplarily for the heating in tank 5:

```
HeatB5.Q_flow =
  if controller.actuators.T5_Heater
  then 20000 else 0;
```

7 Simulation

It is possible to simulate the batch plant model with the StandardWater medium model completely. Simulation results of the tank levels are shown in figure 13.

For the mixture of water and sodium chloride the simulation stops when one of the pumps should start. Till this unwanted ending the results are plausible and discussed below.

In Figure 14 the fluid levels of all tanks are shown. At the beginning, the level of e.g. tank B5 (black line) stays constant till valve V12 opens. Till this valve closes the level rises quite steeply due to the inflow from tank B4 (magenta line), which shows the same changing just in the opposite way. Afterwards the level in tank B5 keeps rising but slower

due to the expansion of the fluid because of the heating. Once some water evaporates, the level of the tank falls as the level of tank B6 (blue crossed line) rises. As soon as the desired concentration is reached the concentrated solution is flowing out of tank B5 into tank B7 (red crossed line) so the level of tank B5 falls steeper. Because of the different diameters of the two tanks B5 and B7 these two curves show different gradients. These results are as expected.

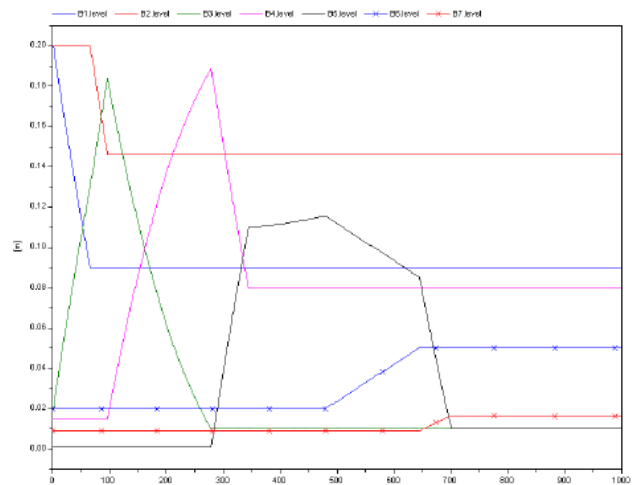


Figure 14: Tank levels of batch plant with water- sodium chloride mixture.

Figure 15 shows in the upper graph the temporal

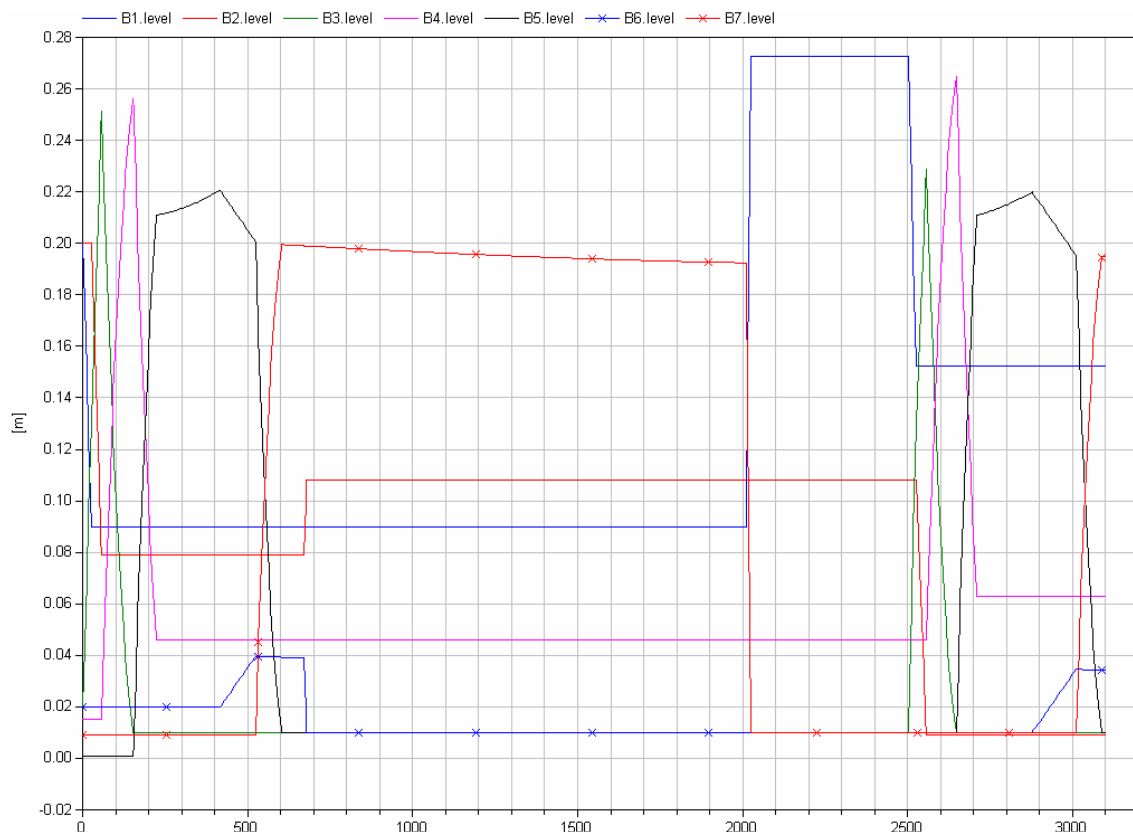


Figure 13: Tank levels of batch plant with StandardWater.

process of the temperature in tank B5 and in the lower image the supplied heat energy is present. When the heat energy changes from 0 to 2000, the temperature starts rising due to the heating. Once the saturation temperature is reached the temperature stays at the boiling temperature which rises only slightly with rising concentration of salt. The supplied energy is then only used for evaporation but not for heating up to higher temperatures than boiling point. This result confirms with reality.

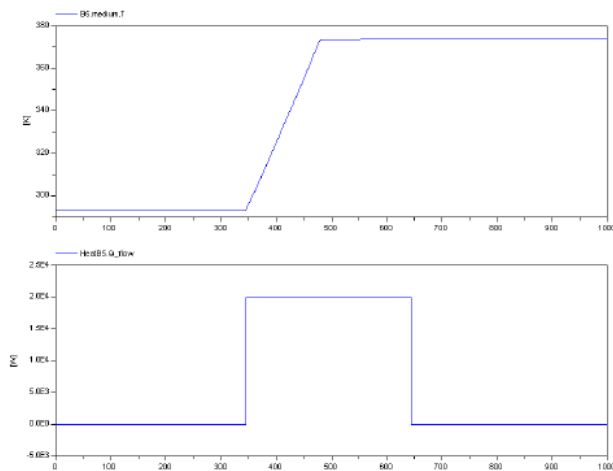


Figure 15: Temporal process of the temperature and the heat flow in tank B5

In Figure 16, the concentration of sodium chloride in tank B3 (blue) and tank B5 (red) are shown over time. For tank B3 the concentration changes when pure water from tank B2 flows into this tank. Once the fluid flow stops, the concentration remains constant. In tank B5 the concentration changes when water is evaporating till the desired concentration is reached and stays constant afterwards.

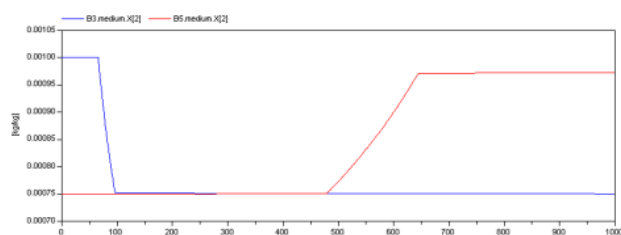


Figure 16: Temporal process of the concentration of NaCl in tanks B3 and B5

All results show qualitatively the expected characteristics.

8 Conclusion and Outlook

The experimental batch plant at AST has been modeled in Modelica. Needed components not available in the Modelica_Fluid library have been imple-

mented. The batch plant model with the Standard-Water medium model, as well as most of the developed components, have been included in version 1.0 Beta 1 of the Modelica_Fluid library and are therefore freely available in open source.

Besides getting the model to work for the whole process duration with the water- sodium chloride mixture, several improvements are desirable: The roughly estimated plant parameters should be more carefully identified and validated with measurement data from the batch plant. The simulation results should be compared with measurement data. The controller should be improved to include exceptional situations such as sensor and actuator failures, as well as stop and shut-down operations. The controller implementation should be made more transparent, which might require extensions to Modelica and tool improvements. Furthermore, it is planned to provide the plant as a benchmark system for tool integration within WP3 of the Network of Excellence HYCON, <http://wp3.hycon.bci.uni-dortmund.de/1.0.html>.

Acknowledgements

This work was partially funded by the Network of Excellence HYCON, contract number FP6-IST-511368.

References

- [1] Kowalewski S., Stursberg O., and Bauer N. (2001): *An Experimental Batch Plant as a Test Case for the Verification of Hybrid Systems*. European Journal of Control 0:1–16.
- [2] Chou J.C.S., and Rowe, A.M. (1969): *Enthalpies of aqueous sodium chloride solutions from 32 to 350°F*, Desalination 6: 105-115.
- [3] Barthel J., Neueder R., and Meier R. (1998): *Electrolyte data collection Vol.XII Part 3c*, DECHEMA.
- [4] Casella F., Otter M., Proells K., Richter C., and Tummeseit H. (2006): *The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks*. Proceedings of the Modelica'2006 conference.
- [5] Poschlad K. (2006): *Modellierung einer Batchanlage mit Ablaufsteuerung in Modelica* (2006). Diplomarbeit, Universität Dortmund, Lehrstuhl für Anlagensteuerungstechnik, Prof. Engell.

Integral Analysis for Thermo-Fluid Applications with Modelica

John J. Batteh

Ford Motor Company, Research and Advanced Engineering

jbatteh@ford.com

Abstract

Integral analysis is a common technique used in the solution of many thermo-fluid problems. While previous applications of integral analysis techniques focused on the solution of boundary layer problems, the techniques are applicable to a wider range of analyses. This paper describes a formulation in Modelica for integral analyses in thermo-fluid applications. Following a brief overview of integral analysis, a sample Modelica implementation is discussed along with numerical simulations that illustrate the model usage in thermo-fluid applications. Additional applications and usage scenarios of the integral analysis formulation are briefly discussed.

Keywords: integral analysis; thermal; fluid

1 Introduction

Heat transfer and fluid flow phenomena are governed by partial differential equations (PDEs) for conservation of mass, momentum, and energy. There are no general, closed-form solutions for these coupled, nonlinear PDEs. While analytic solutions are available for a small class of problems with special boundary conditions, more general analyses with PDEs require numerical approaches, such as discretization and differencing, to resolve the spatial and temporal dynamics.

In contrast to the PDE approach, lumped parameter modeling is formulated by conservation of spatially-integrated quantities. Thus, the temporal dynamics in the relevant conservation equations are retained resulting in ordinary differential equations (ODEs) for the conservation laws of a lumped control volume. It is still possible to obtain spatial dynamics with lumped parameter models by explicitly introducing networks of control volumes. Lumped parameter models typically are relatively easy to formulate, computationally-efficient, and can use standard ODE numerical solvers for simulation. The Modelica Thermal library [1] is a lumped parameter formulation for heat transfer, and Modelica has

been used extensively in thermo-fluid applications (*c.f.* [2]-[5]).

In applications where spatial resolution is important, a hybrid modeling approach that combines aspects of the PDE and lumped parameter formulations may be appropriate. Integral analysis is one such technique that has been used to solve a variety of thermal and fluid problems [6]. The integral analysis technique was first proposed by von Karman [7] and Polhausen [8] for boundary layer solutions and is sometimes referred to as the Von Karman-Polhausen method. This method has been used extensively for boundary layer problems [9] and provides reasonably accurate results when compared with the exact solutions of the governing PDEs with considerably less computational expense and complexity. Integral analysis typically involves the following steps:

- Casting the governing PDEs in integrated form
- Choosing representative spatial profiles with unknown coefficients for the appropriate variables (usually temperature and velocity in thermo-fluid problems)
- Solving for the coefficients as a function of the relevant boundary conditions

This technique can be used for both transient and steady state problems and provides the benefits of a lumped parameter formulation with some additional information regarding spatial distribution without resorting to the solution of PDEs.

This paper describes the implementation of an integral analysis formulation for thermo-fluid applications in Modelica. To illustrate the steps for deriving an integral analysis formulation, sample formulations for various geometries are shown along with example problems illustrating the approach in numerical simulations for simple thermal and thermo-fluid problems with conduction and convection heat transfer. More advanced applications of the resulting models for combined heat and mass transfer simulations are illustrated via a model of heat exchanger fouling.

2 Integral Analysis Formulation

This section outlines an integral analysis formulation for thermal problems. It should be noted that the implementations are geometry-specific due to the functional form of the spatial distribution and that different basis functions could be chosen for a given geometry. The implementations described in this section are meant to serve as sample formulations to illustrate the use of integral analysis techniques in Modelica models and should not be considered universal, *de facto* implementations.

2.1 Planar Geometry

As mentioned previously, integral analysis formulations are most relevant for simulations where additional information regarding spatial distribution is important. Consider the schematic in Figure 1 showing a wall with planar geometry. The wall could be modeled in Modelica with the lumped parameter formulations shown in Figure 2. The two representations reflect different modeling philosophies regarding the placement of the capacitance and flow elements. Each formulation yields a lumped estimate for the temperature distribution in the wall via the mass-averaged temperature computed in the capacitance conservation of energy. Depending on the boundary conditions, geometry and properties of the wall, transient nature of the boundary conditions, and number of capacitances used, the lumped parameter formulation may give a good representation of the actual temperature distribution in the wall. In lumped parameter formulations, higher fidelity representations of the temperature distribution can be realized by networks of lumped capacitances. Rather than adding additional capacitances to provide a better representation of the spatial temperature distribution, an integral analysis formulation can be used. This section gives an overview of the derivation of an integral analysis formulation for planar geometry.

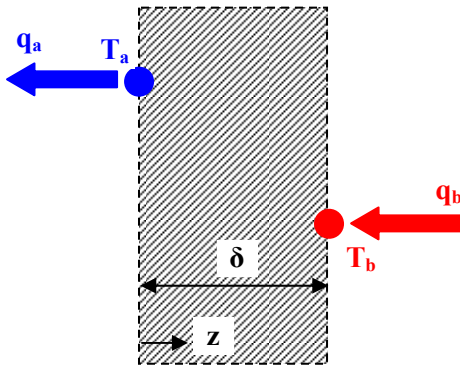


Figure 1. Planar geometry schematic

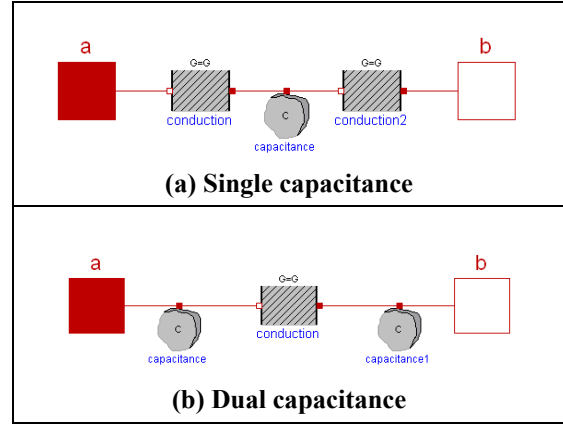


Figure 2. Lumped formulations, planar wall

Again consider the geometry in Figure 1. The first step in the derivation is to assume the form of the temperature profile. Assuming the distribution is one dimensional in the z direction, consider the following distribution:

$$T(z, t) = T_{ss}(z, t) + T_1(z, t) \quad (1)$$

where T_{ss} is the steady-state temperature distribution for a conductive planar wall and T_1 is some assumed temperature deviation:

$$T_{ss}(z, t) = T_a(t) + \frac{z}{\delta(t)} [T_b(t) - T_a(t)] \quad (2)$$

$$T_1(z, t) = a_0(t) + a_1(t)z + a_2(t)z^2 + a_3(t)z^3 \quad (3)$$

Note that a polynomial form has been assumed for the temperature deviation but that other forms are possible.

The next step in the derivation is to determine the unknown coefficients in the temperature profile in terms of the appropriate boundary conditions. Applying the temperature and heat flow boundary conditions at $z=0$ and $z=\delta(t)$ and using the properties of the steady-state solution yields:

$$T_1(z, t) = \frac{q_a(t) - q_b(t)}{kA} \left(-\frac{z^2}{2\delta(t)} + \frac{z^3}{\delta(t)^2} \right) \quad (4)$$

Consistent with the initial construction of the temperature distribution, there is no contribution from the T_1 term at steady state (*i.e.* when $q_a = q_b$).

Though the derivation is in terms of the boundary temperatures, consider a typical interface problem where one of the boundary temperatures is unknown. By combining the integral formulation above with an overall energy balance for the volume, the boundary temperature can be determined in addition to the mean temperature. The overall energy balance for the volume based on the mass-averaged mean temperature T_m yields:

$$\frac{dU}{dt} = q_b(t) - q_a(t) \quad (5)$$

where

$$U = mc_v T_m \quad (6)$$

An equation for T_m can be derived from the definition of a mass-averaged temperature:

$$\rho A \delta(t) T_m(t) = \rho A \int_0^\delta T(z, t) dz \quad (7)$$

Using the assumed temperature distribution and performing the integration leads to the following equation:

$$T_m(t) = \frac{T_a(t) + T_b(t)}{2} + \frac{q_a(t) - q_b(t)}{kA} \frac{\delta(t)}{12} \quad (8)$$

The first term in the equation above is the mean temperature from the steady-state profile while the second term is the transient deviation from steady-state. Note that the deviation term contributes only in non-steady situations and becomes more important when the thickness of the volume increases and the thermal conductivity and surface area decrease, essentially the conditions which contribute to a measurable temperature gradient across the wall.

Figure 3 shows a Modelica implementation of a planar volume based on the integral analysis derivation presented above. The model uses the standard thermal connectors from the Modelica Thermal library. The integral formulation differs somewhat from the generic HeatCapacitor model in the Modelica Thermal library in that the model is not a volume element in the standard sense. The HeatCapacitor model is a single port volume model that is connected to heat flow elements and provides a temperature on its lone connector from its internal conservation of energy equation. The multiport integral planar volume is a combined volume and flow element since it contains a conservation equation that determines a connector temperature similar to a volume element in addition to providing a connector heat flow like a heat flow element based on the heat conduction through the volume that follows from the assumed temperature distribution. A conceptual representation of the integral analysis control volume is shown in Figure 3c to illustrate visually the directionality of this component. This schematic shows how the planar volume from the integral formulation is connected to a flow element on one side (connector b) and a capacitance element on the other (connector a). Because of its directional nature, it is important to connect the integral component carefully to surrounding components to ensure a consistent, complete model.

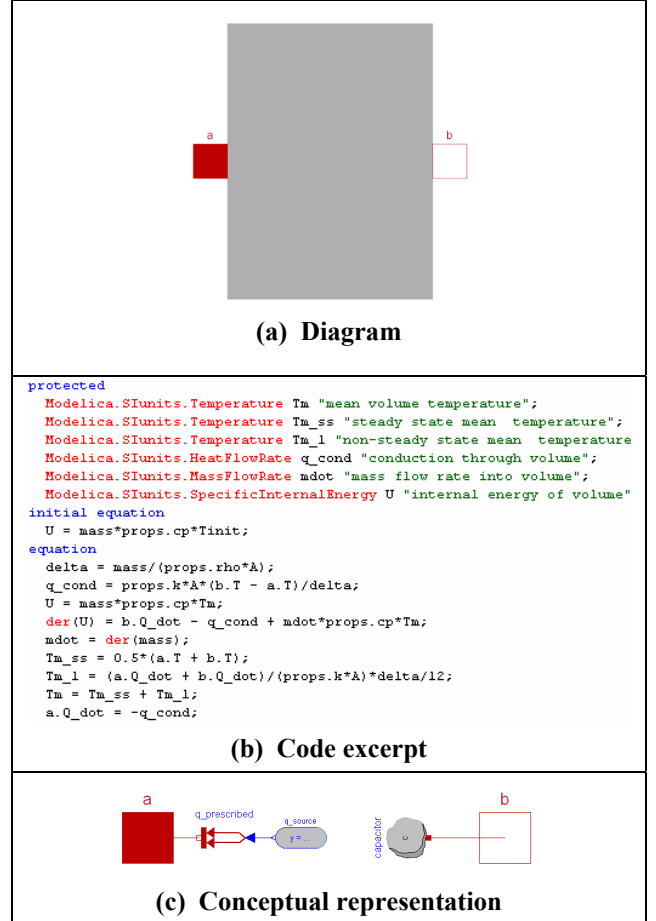


Figure 3. Modelica implementation of planar volume with integral analysis formulation

2.2 Cylindrical Geometry

The general approach used in the previous section can be applied to other geometries given appropriate choices for the temperature distributions. For cylindrical geometry with a one-dimensional temperature distribution in the radial direction, assume the following temperature distribution:

$$T(r, t) = T_{ss}(r, t) + T_1(r, t) \quad (9)$$

where the steady state temperature distribution is given by:

$$T_{ss}(z, t) = T_a(t) + \frac{[T_b(t) - T_a(t)]}{\ln\left(\frac{r_1}{r_2}\right)} \ln\left(\frac{r}{r_2}\right) \quad (10)$$

The choice of the deviation profile is again user and problem specific but could be assumed to have the same form as Eq. (3) in the radial direction. The final form of the integral analysis for cylindrical coordinates follows from the same process outlined for planar geometry: apply BCs to determine the coefficients in the deviation profile, use lumped conservation of energy in conjunction with mass-averaged

temperature distribution, *etc.* The details of the derivation are omitted as they are similar to that in the planar analysis but with some slightly messier algebra due to the radial geometry.

2.3 Comments

The integral analysis formulation results in an approximate method for solution of the geometry-specific PDEs. Integral conservation of energy is maintained as in the lumped parameter formulation. However, some pieces of information used in the solution of the original PDEs are no longer applied, such as the slope boundary conditions at the interfaces, since the temperature distribution is assumed. While this method allows the modeler additional flexibility in terms of the temperature distributions and the specifics of the applications of the integral analysis principles, some care is required to ensure that the physics of the problem are consistent with the assumed distributions and the resulting formulation. The assumption of a temperature distribution throughout the layer leads to some practical limitations on model usage. For example, simulations dominated by an evolving thermal penetration depth would require reformulation since the temperature distribution does not extend through the thickness of the material and thus would not satisfy the assumed temperature distribution. The thermal penetration depth δ_t scales based on the following equation:

$$\delta_t \sim \sqrt{\alpha t} \quad (11)$$

where α is the thermal diffusivity of the material. Setting $\delta_t = \delta$ yields a relationship between the geometry, material properties, and simulation time scale from which the suitability of an integral analysis formulation can be evaluated. In general, the integral analysis formulation becomes increasingly applicable as the material thickness decreases, thermal diffusivity increases, and relevant time scale over which the boundary conditions change increases relative to the penetration depth time scale.

3 Simulation Results

Having developed some sample integral analysis formulations, this section provides some numerical examples to exercise the formulations. All simulations are performed with Dymola [10].

3.1 Planar Wall

Consider the simple test shown in Figure 4 for a planar wall of constant thickness. The wall is sub-

jected to a constant temperature on one side and a constant heat flow on the other side. This test, while extremely simple, provides a nice example to illustrate the utility of the integral analysis formulation. Figure 5 shows the mean and boundary temperatures from simulations of a constant thickness wall with properties of iron and stainless steel [6] at a fixed boundary temperature of 293K. Note that the temperatures in the iron simulation are only slightly higher than the prescribed wall temperature due to the high thermal conductivity of iron. Since the thermal conductivity of stainless steel is about 1/5th that of iron, there is a larger temperature difference across the wall. In addition, the iron wall reaches steady state much more quickly than the stainless steel wall. While the deviation between the mean and boundary temperatures is modest in these sample simulations, the boundary temperature becomes increasingly elevated relative to the mean temperature as the material becomes less conductive. To accurately resolve these potentially-large temperature differences across the wall with a lumped parameter formulation, a network of lumped volumes would be required. Similar behavior is achieved with only a single capacitance using the integral analysis formulation. While this simple example is interesting, applications where the heat and/or mass flow are sensitive to the boundary temperature showcase more clearly the utility of the integral analysis formulation and will be presented in the next sections.

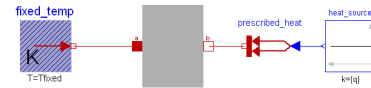


Figure 4. Planar wall, fixed heat flow and temperature boundary conditions

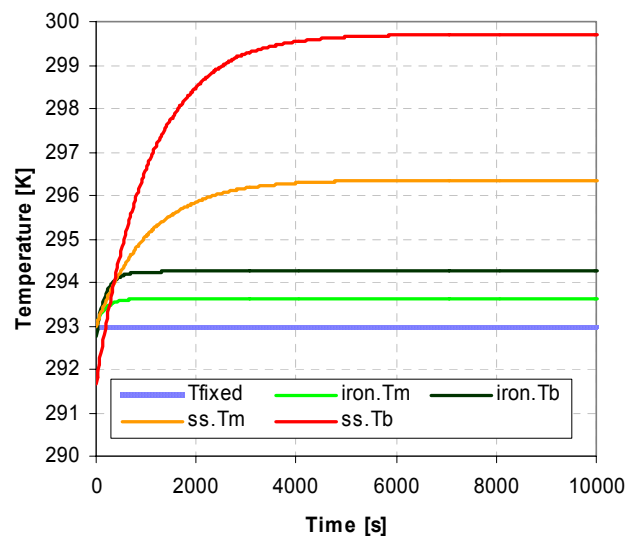


Figure 5. Simulation results, iron and stainless steel

3.2 Planar Wall with Convection

Figure 6 is a simple test similar to that in Figure 4 but with convection heat transfer calculated from the wall boundary temperature. The planar wall, modeled as stainless steel [6], is again subjected to a fixed temperature on one side. The other side is exposed to airflow at a constant velocity with a dynamic temperature given by the `ExpSine` component in the Modelica `Blocks` library [1].

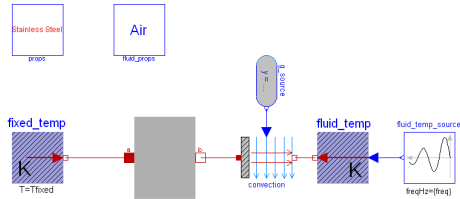


Figure 6. Planar wall with transient convection

Figure 7-Figure 8 show results from the transient convection simulations with varying frequency for the prescribed fluid temperature fluctuations, 0.001 and 0.01 Hz respectively. The top plot in each figure shows the input fluid temperature, and the bottom plot gives the fixed, mean, and interface metal temperatures from the integral analysis. As the wall is heated by the warmer fluid, a temperature gradient is again established across the wall. Unlike the fixed heat flow simulations in the previous section, the convective heat flow boundary condition is calculated from the interface temperature and, along with the fluctuating fluid temperature, adds additional transient dynamics. It is interesting to note the way in which the fluid temperature fluctuations are reflected in the mean and interface temperatures. While the mean temperature also exhibits some characteristics of the input temperature oscillations, they are damped out by the capacitance of the wall. The interface temperature, however, is significantly more oscillatory, as is to be expected. As the frequency of the oscillations increases in Figure 8, the mean temperature hardly reflects any of the oscillations despite the high frequency oscillations in the interface temperature. By introducing some notion of the temperature distribution within the volume, the integral analysis formulation provides a mechanism for a higher fidelity representation of both the spatial distribution in the volume (*i.e.* the difference between the mean and interface temperatures) and the differences in transient response (*i.e.* fast response at the interface and damped response in the mean).

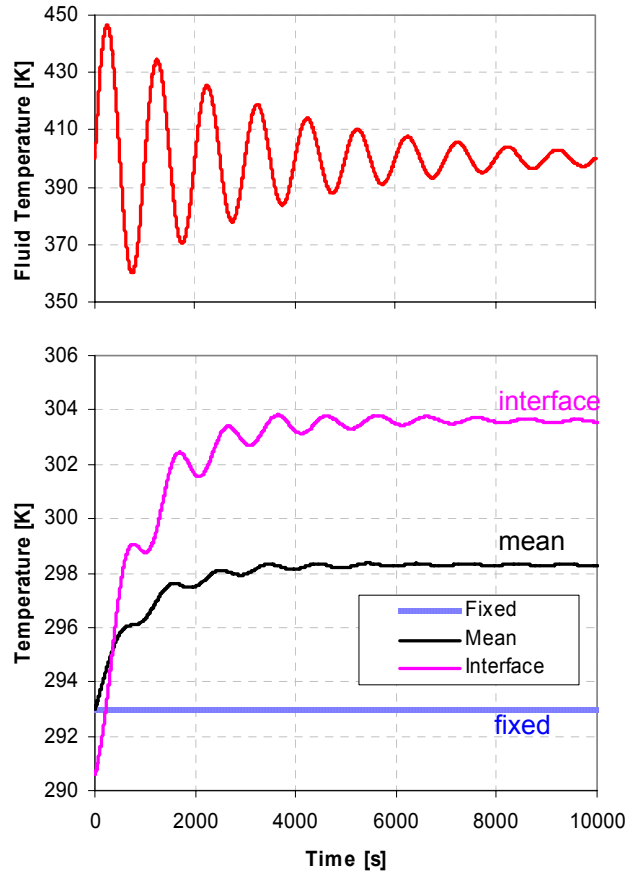


Figure 7. Simulation results, frequency=0.001Hz

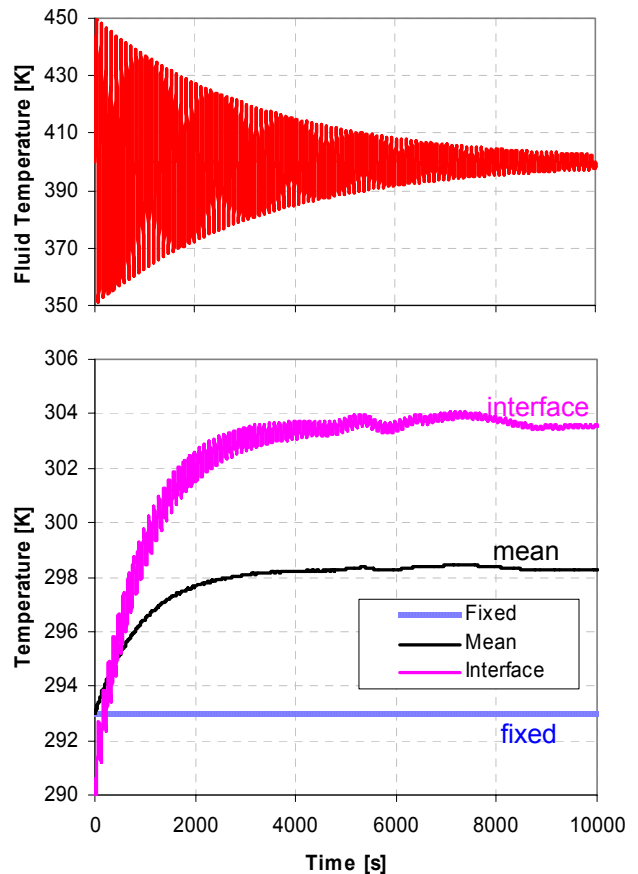


Figure 8. Simulation results, frequency=0.01Hz

3.3 Simulation with Mass/Heat Transfer

While the previous examples contained only heat transfer physics for a layer of constant thickness, the original derivation allowed for dynamic geometry resulting from heat and mass transfer. The example in this section simulates the fouling layer build-up and subsequent thermal degradation common in industrial heat exchangers. Since the fouling caused by deposition of particulates and condensation of vapor phase materials is highly-sensitive to the interface temperature between the fouling layer and the gas [11], an integral analysis formulation is used to represent the dynamics of the fouling layer.

Figure 9 shows a simple test model for simulating the effects of a deposition layer in a pipe. The inner pipe wall is assumed to be at a constant temperature, and the deposition layer is modeled using the cylindrical integral analysis formulation outlined in Section 2.2. The model consists of standard convective heat transfer based on the Sieder-Tate Nusselt number correlation for pipes [6] and a boundary layer formulation for thermally-induced particulate mass transfer [12]. Material properties for air are used for the flowing medium in the pipe, and the deposition layer is assumed to have the material properties of asphalt [6].

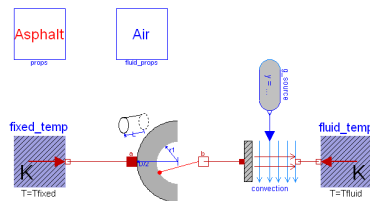


Figure 9. Pipe deposition layer with heat and mass transfer

Figure 10 shows simulation results for the pipe deposition model. The model was simulated with a fixed mass flow rate and inlet gas temperature. The simulations show the slow degradation in heat transfer performance as the deposition layer grows, thus providing an additional thermal resistance between the hot gas and the cool pipe wall. This degradation is evidenced by the increasing gas exit temperatures and alternatively by the decreasing effectiveness ϵ , defined in the following way for constant mass flow conditions:

$$\epsilon = \frac{T_{gas,in} - T_{gas,out}}{T_{gas,in} - T_{wall}} \quad (12)$$

Note the difference between the mean layer temperature and the interface temperature. Since the flowing medium interacts with the layer at the interface temperature for both the heat and mass transfer dynamics, it is important to capture this elevated interface temperature relative to the mean. Furthermore, since the elevated interface temperature and subsequent thermal degradation result from the dynamic layer thickness, it is also important to capture the transient nature of the layer growth. The integral analysis formulation allows a higher fidelity representation of the spatial temperature distribution in the layer without the need for a network of heat transfer components to represent the layer.

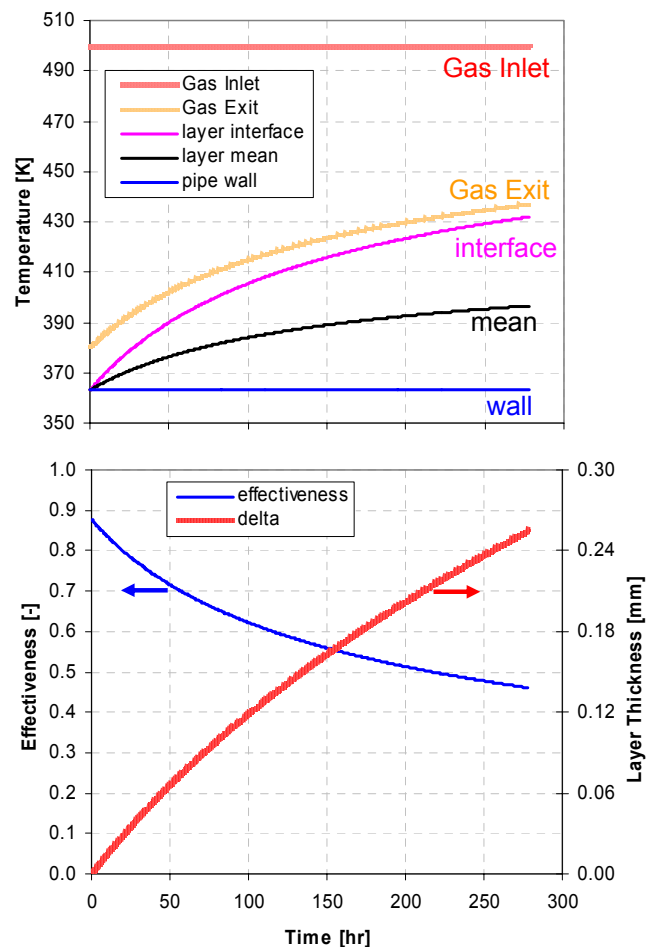


Figure 10. Pipe deposition simulation results

4 Conclusions

This paper illustrates an extension of the application of the Modelica language to thermo-fluid problems using an integral analysis approach. The sample formulations presented here give details regarding the development and implementation of the integral analysis approach in Modelica for different

geometries. This approach allows the computational benefits of a lumped parameter formulation with additional details regarding the temperature distribution across the volume. In particular, the integral analysis formulation is especially useful for problems with interface dynamics, particularly those which require accurate resolution of the interface conditions to obtain the appropriate dynamic physical response.

Acknowledgements

The author gratefully acknowledges Chuck Newman for his valuable discussions regarding this work.

References

- [1] Modelica Association, 2006, "Modelica Language Specifications (Version 2.2.1)", <http://www.modelica.org>
- [2] Elmqvist, H., Tummescheit, H., and Otter, M., 2003, "Object-Oriented Modeling of Thermo-Fluid Systems", *Proceedings of the 3rd International Modelica Conference*, p. 269-286, http://www.modelica.org/events/Conference2003/papers/h40_Elmqvist_fluid.pdf
- [3] Newman, C., Batteh, J., and Tiller, M., 2002, "Spark-Ignited-Engine Cycle Simulation in Modelica", 2nd International Modelica Conference Proceedings, pp. 133-142, http://modelica.org/Conference2002/papers/p17_Newman.pdf
- [4] Batteh, J., Tiller, M., and Newman, C., 2003, "Simulation of Engine Systems in Modelica", 3rd International Modelica Conference Proceedings, pp. 139-148, http://www.modelica.org/events/Conference2003/papers/h34_Batteh.pdf
- [5] Steinmann, W., Buschle, J., 2005, "Analysis of thermal storage systems using Modelica", *Proceedings of 4th International Modelica Conference*, pp. 331-337, http://www.modelica.org/events/Conference2005/online_proceedings/Session4/Session4b1.pdf
- [6] Incropera, F.P. and DeWitt, D.P., 1990, *Fundamentals of Heat and Mass Transfer*, 3rd Edition. John Wiley & Sons.
- [7] Von Karman, T., 1921, "Uber Laminar Und Turbulente Reibung", *Z. Agnew. Math. Mech.*, vol. 1, pp. 233-252.
- [8] Pohlhausen, K., 1921, "Zur Naherungsweise Integration der Differential Gleichung der Laminaren Reibungsschicht", *Z. Agnew. Math. Mech.*, vol. 1, pp. 252-268.
- [9] Schlichting, H., 1968, *Boundary Layer Theory*, 6th Edition. McGraw-Hill.
- [10] Dymola. Dynasim AB, Lund, Sweden, <http://www.dynasim.com>.
- [11] Kittelson, D.B., Ambs, J.L, and Hadjkacem, H., 1990, "Particulate Emissions from Diesel Engines: Influence of In-Cylinder Surface", SAE-900645, Society of Automotive Engineers.
- [12] Suhre, B.R. and Foster, D.E., 1992, "In-Cylinder Soot Deposition Rates Due to Thermophoresis in a Direct Injection Diesel Engine", SAE-921629, Society of Automotive Engineers.

Session 6c

Free and Commercial Libraries 2

Integration of CATIA with Modelica

Partha Bhattacharya Nick-Ange Suyam Welakwe* Ranga Makanaboyina

Adithya Chimalakonda

DaimlerChrysler Research and Technology India

*DaimlerChrysler Research and Technology Germany

Partha.Bhattacharya@DaimlerChrysler.com Nick-Ange.Suyam_Welakwe@DaimlerChrysler.com

Ranga.Makanaboyina@DaimlerChrysler.com Aditya.ch@DaimlerChrysler.com

Abstract

Virtual Prototyping is becoming increasingly popular in the automotive and other industries, especially in the context of concurrent engineering processes. CATIA [1] is a powerful CAD tool for modeling and Modelica based tools are increasingly becoming popular for analysis using multibody simulation. In this work, we developed an interface that integrates Modelica into CATIA to facilitate product data information exchange. This enables automatic extraction of pertinent information from CAD representation of the physical components to Modelica models for multibody simulation. The integration will reduce the design cycle significantly by providing immediate feedback to the designer with minimal intervention of simulation and modeling specialists. We studied three different implementation approaches for integration. The results are presented for a mechanism that opens the car-hood.

Keywords: Multibody Simulation, Concurrent Engineering, Virtual Prototyping, Interface Tools, Joints, CATIA, Integration, Modelica.

1 Introduction

Two important phases in Virtual Prototyping, an integrated part of the modern product development process, are design and analysis. While design deals with form, analysis deals with behavior of the system components. Form can be represented by CAD models (Design) and behavioral models, at different accuracy levels, can be simulated (Analysis) to compare with the product function specifications. Multibody Simulation or motion analysis is an important part of the analysis phase. Generally, design and analysis, are done independently. Designer and Dynamic simulation expert are different persons. But Simulation experts and designers need to exchange data in order to validate their models. This data exchange is made most of the time manually. There is no guarantee that the exchanged data are right and up

to date. So tools are needed to exchange information between these two processes so that designers can take a holistic design approach [2] [3].

The main benefit of the interface we developed is to provide a user friendly framework to support the concurrent engineering process between dynamic simulation experts and designers taking some specific organizational process constraints into account. With these constraints in place, the interface contributes significantly to the improvement of the data exchange process. Here the goal is not to provide an integrated multi body simulation environment in CATIA but to keep CATIA and Modelica separate and just enable the data exchange between CATIA models and corresponding Modelica models.

In fact, most of the commercial MBS softwares provide interface to standard CAD softwares. For example SIMPACK [4] has an interface, CATSIM, to CATIA. SimMechanics [5] provides translator from SolidWorks models, SimDesigner [6] interface enable multi body simulation within CATIA V5.

Modelica is becoming increasingly popular in the automotive industry, at least because of object oriented multidomain modeling and simulation capabilities. A typical application area of Modelica is in modeling of mechatronic systems. In these systems, multi body simulation plays a significant role. The objective of making an interface between CATIA and Modelica is to provide a process and user friendly framework for the exchange of kinematics data.

People have already integrated CAD data into Modelica simulation [7]. MathModelica, one of the popular Modelica tool, offers interface to SolidWorks [8]. Our work is different from the past works in that we are not proposing to automatically translate a CAD model directly which might not be very suitable for MBS, though its one of the options discussed in here (Option I), but we are proposing a semi-automatic data extraction appropriate for MBS.

2 CATIA Data required for MBS simulation in Modelica

A MBS model in Modelica mostly consists of rigid bodies connected by different kind of joints. These bodies interact via Joints and respond to different forces i.e. gravity. Joints define the degrees of freedom between bodies. To define Bodies, three sets of information are required. They are,

- A. **Physical Information:** Mass, centre of mass and inertia tensor of the bodies. CATIA models of the corresponding bodies store mass and inertia of the assembly or individual parts which can be extracted automatically.
- B. **Geometrical Information:** Modelica uses 'Mechanical Pin' or 'frame_a/b' which is nothing but the 'connecting' point of the body. Getting this information automatically is challenging. Fixing a point to define frame_a/b in a 3-D body is difficult. A CAD designer can define points, frame_a/b on CATIA parts which can be chosen interactively. The center of gravity position vector is also required and can be extracted automatically from the CAD model.
- C. **Visualization information:** This data is required for animation. We use BodyShape to represent a body in Modelica. This body can be represented by different available simple shapes or it can refer to CAD data for visual representation. This data can either be in DXF format or STL format. Here we convert CATIA models (CATProduct, CATPart, PartBody) to STL files and use the same in Modelica.

Apart from the bodies, Joints can be defined through a set of points/lines/planes depending on the nature of joints which will be defined by the CAD designers in CATIA components. These points/lines/planes can be chosen by the user and extracted to Modelica.

Component characteristics can be optionally required to simulate some special physical effects. But these characteristics are not usually stored in the CAD model and therefore don't need to be extracted.

3 Implementation Approaches

We have three possible scenarios for integration.

- I. **Translation of CATIA V5 defined Multi-Body structure to Modelica model.** CATIA offers a workbench called DMU Kinematic Simulator in which a mechanism can be defined with different kind of joints. A MBS structure thus constructed

can be directly translated to Modelica. Because, for example, a revolute joint in CATIA can be replaced with a revolute joint in Modelica. This seems to be an easy and obvious choice. But DMU Kinematic Simulator is suitable for kinematic analysis only. In practice, using the same kinematic topology for dynamic analysis is not always possible.

- II. **Development of an add-on to the CATIA interface to define the MBS structure.** To overcome the above drawbacks and to replace the function of DMU Kinematic Simulator, we can develop an add-on to the CATIA interface to define the MBS structure where a modeler can define a MBS structure using connection diagram in a visual editor (like Dymola [9]) which supports drag and drop. This is done in Dymola model editor. Another example of this kind of editor is DynaFlexPro from Maple [10]. This is going to be heavy on the CAD designer who might not have the MBS fundamentals. The best thing will be to see the mechanism already defined and just relate it to the CATIA model which leads us to the next option.
- III. **Existing Modelica model integrated to its CATIA counterpart.** The MBS modeler makes a Modelica MBS skeleton/structure. He also sits with the CATIA designer to define a configuration / mapping file. This mapping file related BodyShapes and bodies to CATParts or CATProducts. It also has information about the joints in terms of lines, planes and points. If additional points are required CATIA designer defines them. With the mapping file and the CATIA model, automatic extraction of appropriate data takes place. The Modelica skeleton gets filled with information. Here for the first time we are talking about starting the flow from Modelica. This definitely lacks the flexibility, user interaction on the CATIA side and most importantly visibility.

4 Implementation Concept

Taking into consideration all the advantages and the drawbacks of the above approaches, we have converged to the final option which we demonstrate with a scenario here.

1. To simulate a mechanism, a kinematic skeleton is constructed with Modelica. The geometric models of the components in this skeleton exist in CATIA.

2. The CAD model of the components is opened in CATIA. A toolbar is created with CATIA Product Design workbench. Using one of the buttons in this the user imports the corresponding Modelica file to CATIA workspace.
3. Using another button the user invokes a graphic user interface (GUI) where all the details of the Modelica components (BodyShape and Joints) are shown.
4. User selects the appropriate parts from CATIA interface to relate it to the proper objects in the diagram.
5. All the data are then extracted and assigned to proper objects. The extracted data is exported back to Modelica script.

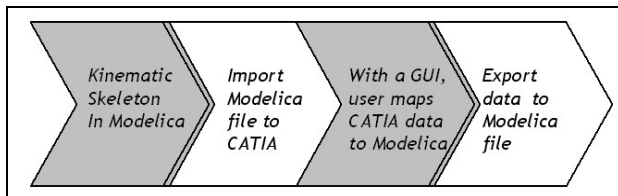


Figure 1: Process flow for the Implementation

Figure 1 gives an overview about the Steps described above.

5 Implementation Results

CATIA allows a set of C++ Application Programming Interfaces (API) as part of Component Application Architecture (CAA) to access functionalities of CATIA models. We have chosen CAA as our development platform. To implement the solution we need core CATIA functionalities with graphical user interface programming. CAA has ability to program graphic interfaces but it's limited. So we used CAA for accessing CATIA's core functionalities. The implementation consists of the following steps:

5.1 Design the kinematic skeleton

The product is decomposed into body shapes P1, P2, P3 and P4, and prismatic joint PJ1, revolute joints RJ1, RJ2 and RJ3 as shown in figure 2.

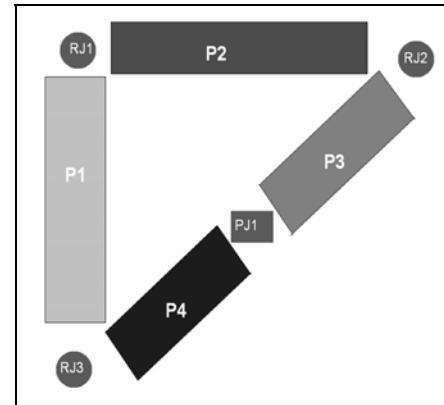


Figure 2: Kinematic skeleton

5.2 Convert the skeleton to Modelica model

Figure 3 shows the corresponding model of the kinematic skeleton in the Dymola editor.

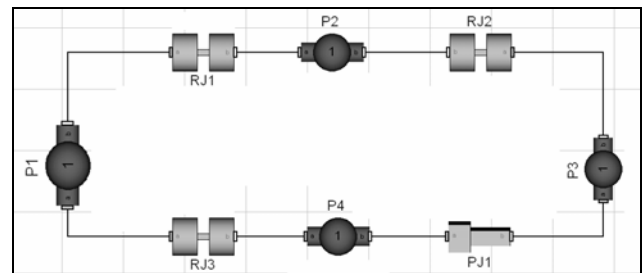


Figure 3: Corresponding Modelica model

5.3 Import Modelica model to CATIA workspace

Our algorithm scans the Modelica script and picks up the relevant components like BodyShape, different Joints etc. and lists them.

5.4 Extract the CATIA data

The next step involves opening the corresponding CATIA model and starting data extraction (figure 4).

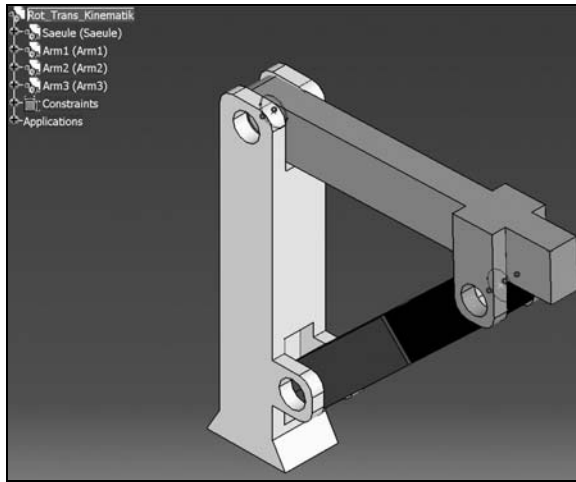


Figure 4: Related CATIA Model

5.4.1 For BodyShapes:

When any BodyShape component is selected in the GUI user can specify a CATPart or a CATProduct to relate it to the BodyShape component. For any BodyShape component we need mass, inertia tensor and centre of mass information. This information is already stored in CATIA. We need to specify the shape information in terms of frame a/b. User can choose points on the component to specify this information (figure 5). Visualization information is exported by converting the CATPart/CATProduct into an STL file.

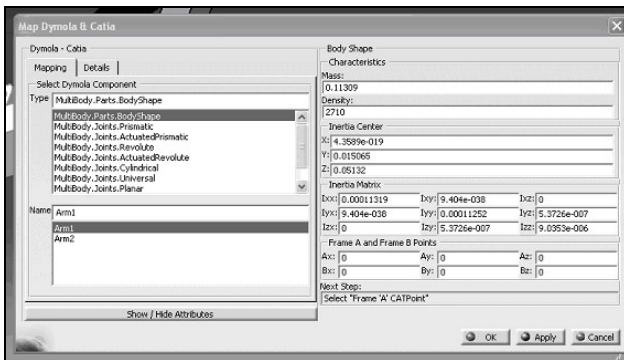


Figure 5: Interface for BodyShape

5.4.2 For Joints:

When a joint is selected in the GUI, details of its connecting frames are shown. User can now choose the required data by choosing the corresponding geometric element in CATIA as shown in figure 6. For example, when a revolute joint is selected in the GUI, details of its connecting frames are shown (figure 6). User can now choose the axis of revolution by choosing a line on the components.

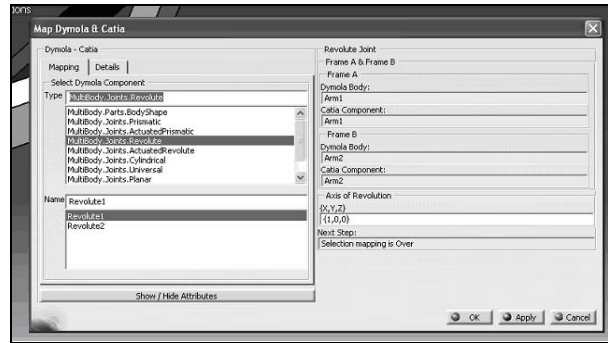


Figure 6: Interface for Revolute Joint

5.5 Update Modelica model

Figure 7 is the Modelica script before the data export. Figure 8 is the Modelica script after the extracted data is appended to the Modelica script for the corresponding components. It may be noted that the mass, centre of mass and inertia tensor values, and geometrical shape values are updated in the Modelica script.

```
World world @;
Revolute Revolute1 @;
ActuatedRevolute ActuatedRevolute1 @;
BodyShape Saeule @;
FixedTranslation FixedTranslation_Saeule @;
BodyShape Part2 @;
BodyShape Part3 @;
BodyShape Part4 @;
Prismatic Prismatic1 @;
Revolute Revolute2 @;
timeTable1(table={0,0; 1,Modelica.Constants
```

Figure 7: Modelica Script of kinematic skeleton

```
Modelica.Mechanics.MultiBody.Parts.BodyShape Part2(m=0.043511,
r={0.000000,0.000000,0.000000},
r_CM={0.000000,0.146040,0.021204},
shapeType="1",
I_11=0.000012,
I_22=0.000002,
I_33=0.000011,
I_21=0.000000,
I_31=0.000000,
I_32=0.000000);
Modelica.Mechanics.MultiBody.Parts.BodyShape Part3(m=0.074444,
r={0.000000,0.000000,0.000000},
r_CM={0.000000,0.174394,0.062528},
shapeType="2",
I_11=0.000042,
I_22=0.000002,
I_33=0.000042,
I_21=0.000000,
I_31=0.000000,
I_32=0.000000);
```

Figure 8: Updated Modelica Script after Data Export

6 Conclusion

We have presented an integrated interface between CATIA V5 and Modelica. The interface supports the exchange of kinematics data between CATIA model and Modelica model. It provides further a framework to support the concurrent engineering process between dynamic simulation experts and designers taking following organizational and OEM process constraints into account:

- Designer and dynamic simulation expert are different roles
- Designer only works with CATIA and the simulation expert with Dymola
- Design process is the master.

The use of the interface requires minimal process changes from the user for data exchange and thus guarantees high process flexibility.

References

- [1] www.3ds.com
- [2] R. Sinha, C.Paredis and P.Khosla, “Integration of mechanical cad and behavioral modeling”, Proceedings of the IEEE/ACM Workshop on Behavioral Modeling and Simulation, 2000.
- [3] V. Engelson, “Tools for design, interactive simulation and visualization for dynamic analysis”, Linkoping Electronic Articles in Computer and Information Science, 5 (2000).
- [4] <http://www.simpack.com>
- [5] <http://www.mathworks.com>.
- [6] <http://www.mscsoftware.com>
- [7] P. Bunus, V. Engelson, and P. Fritzson, “Mechanical models translation, simulation and visualization in Modelica”, in Modelica Workshop 2000 Proceedings, pp. 199–208.
- [8] <http://www.mathcore.com>.
- [9] <http://www.Dynasim.se>
- [10] <http://www.maplesoft.com/>

A Modelica Library for Simulation of Household Refrigeration Appliances Features and Experiences

Carsten Heinrich, Kai Berthold

Institute for Air Conditioning and Refrigeration, Department Refrigeration and Cryogenics
Bertolt-Brecht-Allee 20, 01309 Dresden, Germany

Abstract

Today the vast majority of household refrigerators and freezers work with on-off-control of the compressor. This control method leads to significant dynamic behaviour of the refrigeration cycle and cabinet. Moreover, some phenomena cause energy loss during the off-state. The development of new appliances based on static design and simulation tools neglecting all those dynamic effects will not yield optimal results.

Therefore, a Modelica library for the simulation of household refrigerators and freezers was developed. The modelling of the vapour compression cycle is based on the Modelica Media and Fluid library. The heat exchangers are modelled according to the finite volume method. A media model based on quadratic equations was developed for the refrigerant R600a (Isobutane) to allow fast calculations. The modelling of the refrigeration cycle includes effects like different void fraction models and refrigerant sorption in the compressor lubricant.

The library was validated for a one temperature zone refrigerator and a refrigeration-freezer-combination with evaporators connected in series. Adopted script functions have been written to allow parameter runs with one or two parameters or with different compressors. The library combines model features of recent studies Janssen [2], Philipp [5] and Radermacher [7] with a high degree of flexibility and a parameterization by data which can be obtained easily.

Keywords: *household refrigeration appliances; vapour compression cycle; R600a*

1 Introduction

In Germany 15% of the total CO_2 -emissions are caused by the domestic sector. According to data of the VDEW, about 30% of the electrical energy con-

sumption of the domestic sector are used by refrigerators and freezers. Achieving a better energy efficiency will contribute to a decrease of the CO_2 -emissions in mid term.

Most of the household refrigerators and freezers operates with on-off-control of the compressor. A variable speed compressor can achieve higher energy efficiency (up to 25% less energy consumption). But still nowadays variable speed compressors requires an higher investment from the consumer side.

Due to the on-off-control, the refrigeration cycle and the cabinet show a distinct dynamic behaviour most of the time. Therefore, the use of static design and simulation tools will not lead into optimal results. Particularly, some effects caused by the on-off-control, e.g. the refrigerant migration, are neglected.

Hence, dynamic simulations represent an essential tool to achieve further improvements. The aim of this project was to develop of a Modelica Library for the optimization of household refrigerators and freezers. Importance was given to model parameterization is possible with data which are easy to obtain for the end-user of the library. To get good results void fraction models for the two phase flow and the effect of the refrigerant sorption in the compressor lubricant were implemented.

2 Components and Structure

A schematic drawing of a household refrigerator with the simplest cycle option is shown in Fig. 1.

The main components of the systems are the hermetic piston compressor, the condenser, the non adiabatic capillary tube with the suction line heat exchanger, the evaporator and the cabinet. More complex cycles are realized, e.g. two or more evaporators in series or parallel with magnetic valve. Optional components like stop valves and frame heating are included in the li-

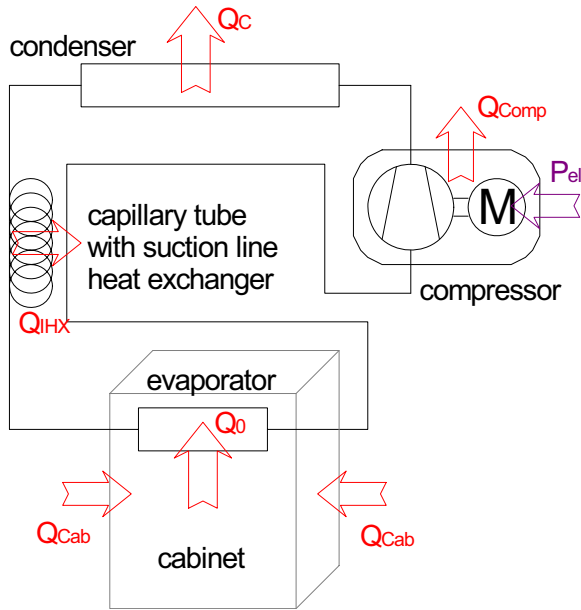


Figure 1: Schematic drawing of a household refrigerator with one temperature zone

brary.

2.1 Hermetic Piston Compressor

Hermetic compressors of household appliances have a dynamic behaviour with considerable larger time constants compared to refrigeration systems in the commercial and industrial sector. Since for a decreasing norm capacity the weight only decreases slightly. The ratio of the heat capacity of compressor to the electrical power consumption

$$\tau_{Therm,Comp} = \frac{C_{p,Comp}}{P_{el}} \quad (1)$$

can be used to compare the dynamic of the thermal behaviour of compressors, approximately. Using the heat flow of the compressed gas and the rejected heat by the motor to the compressor instead of the electrical power consumption would be more exact, but requires more data of the compressors.

For hermetic compressors of household refrigeration appliances $\tau_{Therm,Comp}$ is in the range of 30 to 60 s. Compressors of the commercial sector, e.g. for supermarket refrigeration applications ($\dot{V}_{Gasflow} = 50 \text{ m}^3/\text{h}$) $\tau_{Therm,Comp}$ is about 1 s.

Furthermore the refrigerant charge of the whole system is very small compared to the lubricant charge of the compressor. An average refrigerant charge of an refrigerator with R600a as refrigerant is approxi-



Figure 2: Hermetic piston compressor for household refrigerators with opened shell

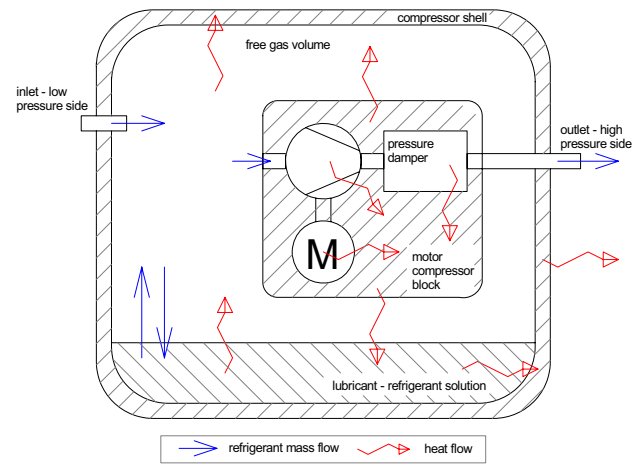


Figure 3: Schematic drawing of a hermetic compressor and existing mass and heat flows

mately 30 g. Lubricant charge of hermetic compressors is about 220 g.

Hence, the dynamic behaviour of the compressor should be taken into account. For this purpose a detailed thermal model of the compressor and its sub-components have been realized as shown in Fig. 3.

2.1.1 Refrigerant Sorption in the Lubricant

The saturation solubility of the refrigerant in the lubricant ζ_{Sat} depends on the temperature T_{Lub} and the pressure p_{Lub} of the lubricant and the type of the lubricant. Mineral and polyolester oils are applied as lubricants. Fig. 4 shows the saturation solubility as function of temperature for different pressures of a polyolester oil. The implementation in Modelica is done by using an approach

$$\zeta_{Sat} = \frac{(c_1 \cdot p_{Lub})^{c_2}}{T_{Lub} + c_3} \quad (2)$$

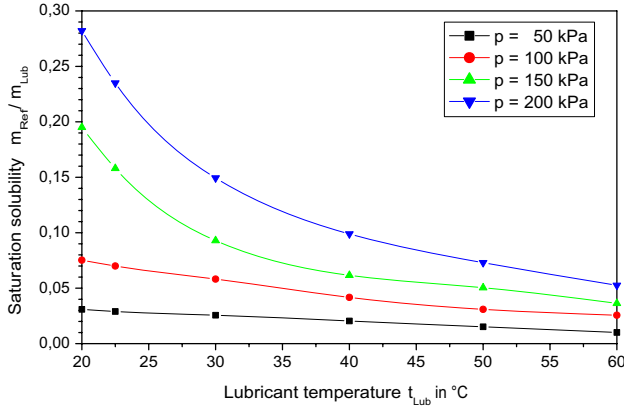


Figure 4: Saturation solubility of a used refrigerant - lubricant mixture

The parameters c_1 , c_2 , c_3 are fitted for applied lubricant - refrigerant systems based on data of [1] and further measurements.

T_{Lub} , p_{Lub} vary distinctly with the on-off-state of the compressor. The absorption and desorption process depends on the difference of the saturation solubility and the absorbed refrigerant.

$$\frac{d\zeta}{d\tau} = \frac{1}{\tau_{Sorp}} (\zeta_{Sat} - \zeta) \quad (3)$$

The circulation of the lubricant varies during on and off states of the compressor. Hence different time constants τ_{Sorp} are applied as described in Philipp [5].

2.1.2 Calculation of Mass Flow Rate and Efficiency

For the parameterization of a detailed volumetric efficiency model based on Eq. 4

$$\dot{m} = \rho_{suc} \cdot V_{Dis} \cdot \lambda(p_C, p_E) \cdot n \quad (4)$$

it is necessary to know about the displacement volume V_{Dis} , the revolution speed n , the exact inlet conditions into the cylinder ρ_{suc} and the characteristic behaviour of the volumetric efficiency λ . The characteristics which can be obtained for hermetic compressors are usually limited to a data set of one or more points of the condensation temperature t_C , the evaporation temperature t_E , the refrigeration capacity \dot{Q}_0 and the electrical power consumption P_{el} .

Hence, for simulation purpose V_{Dis} , λ and n are combined:

$$\Lambda(p_C, p_E) = V_{Dis} \cdot \lambda(p_C, p_E) \cdot n \quad (5)$$



Figure 5: Tube-and-wire condenser

Λ is described by an cubic approach of pressure ratio:

$$\Lambda(p_C, p_E) = \sum_{i=1}^4 c_i \left(\frac{p_C}{p_E} \right)^{i-1} \quad (6)$$

The coefficients are determined during initialization by least square method using the given data. ρ_{suc} must be corrected, owing to the higher temperature in the compressor shell.

The electrical power consumption is calculated by the isentropic power P_{is} and the isentropic, mechanical and electrical efficiencies η_{is} , η_{mech} , η_{el} .

$$P_{el} = \frac{P_{is}}{\eta_{is} \cdot \eta_{mech} \cdot \eta_{el}} \quad (7)$$

For the isentropic power consumption, a simplified approach according to Lippold [3] is used: P_{is} depends on p_C , p_E , \dot{m} , ρ_{suc} and the isentropic exponent κ . The efficiencies are combined for the same reason as above mentioned. For the combined efficiency η a cubic approach in respect to P_{is} is applied. Coefficients are determined during the initialization.

2.2 Heat Exchanger

The main heat exchangers are modelled according to the finite volume method (FVM) used in the ThermoFluid Library [9]. Heat transfer coefficients at the refrigerant side are calculated according to Shah [8]. Heat transfer at air side is separated in convection and radiation. Replaceable components are used to consider different types of evaporator and condenser.

2.2.1 Modelling of Two Phase Flow

The void fraction models of the two phase flow were paid attention due to the great influence on the system

behaviour. Reasons for the high influence are the low mass fluxes, the geometry of the heat exchangers (particularly the evaporator) and the capillary tube as fixed vessel in household refrigeration appliances. Void fraction models according to Hughmark, Lockhardt-Martinelli and Premoli were applied.

The application of void fraction models increases the computation effort significantly. The void fraction model according to Premoli [6] with correction factors of the liquid volume fraction according to Janssen [2] provided the most stable simulations with good results. The liquid fraction depends on the pressure p , the saturation densities ρ' , ρ'' , the dynamic saturation viscosities η' , η'' , the vapour quality x , the hydraulic diameter d_H , the mass flux G and a correction factor depending on geometry c_G .

Based on the vapour volume fraction of the gas phase $\alpha = V''/V$ of homogenous two phase flow a slip factor s is introduced.

$$\alpha = \left(1 + \frac{1-x}{x} \frac{\rho''}{\rho'} \cdot s\right)^{-1} \quad (8)$$

The slip s is calculated by

$$s = 1 + K \sqrt{\frac{y}{1 + Cy}} - Cy \quad (9)$$

$$y = \frac{x}{1-x} \quad (10)$$

The parameters K and C correlates to the flow conditions and are calculated by the Reynolds Number Re and the Weber Number We .

$$Re = \frac{G \cdot d_H}{\eta' + x(\eta'' - \eta')} \quad (11)$$

$$We = \frac{G^2 \cdot d_H}{\sigma} \quad (12)$$

$$K = 1,578 \cdot Re^{-0,19} \left(\frac{\rho'}{\rho''}\right)^{0,22} \quad (13)$$

$$C = 0,0273 \cdot We Re^{-0,51} \left(\frac{\rho'}{\rho''}\right)^{-0,08} \quad (14)$$

To introduce the correction factor for the liquid volume fraction c_{lvf} proposed by Janssen [2] equation (8) has to be rewritten as

$$\alpha = 1 - c_{lvf} \left[1 - \left(1 + \frac{1-x}{x} \frac{\rho''}{\rho'} \cdot s\right)^{-1}\right] \quad (15)$$

The top of Fig. 6 shows the refrigerant charge in the evaporator and suction line calculated by homogenous

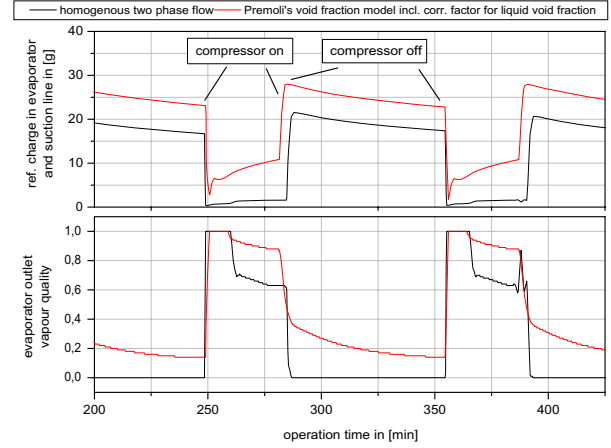


Figure 6: Refrigerant charge in the evaporator and suction line (top) and evaporator outlet conditions (bottom) for different two phase flow models

two phase flow and by void fraction model acc. to Premoli with a correction factor for the liquid fraction for a total refrigerant charge of 36 g. Refrigerant charge calculated by homogenous two phase flow model is only in a range of 10 to 20% of the charge calculated by the void fraction model. A further difference can be identified in the evaporator outlet conditions (Fig. 6-bottom). To get reliable results, a void fraction model has to be applied.

2.3 Capillary Tube with Suction Line HX

The main part of capillary tube is inside the suction line tube and acts as an internal heat exchanger. The internal heat exchanger ensures superheated inlet conditions at the compressor and has a positive effect on the cycle efficiency.

The mass flow rate of the capillary tube \dot{m}_{Cap} is calculated by a function of p_C , p_E , the inlet vapour quality x_{in} or the degree of subcooling Δt_{sc} , the capillary length and the diameter L_{Cap} , d_{Cap} and the heat flow rate between capillary tube and suction line \dot{Q}_{IHX} .

$$\dot{m}_{Cap} = K_{Geo} \cdot K_{sc} \cdot K_{IHX} \cdot [K_1(x_{in}) \Delta p + K_2(x_{in}) \Delta p^2] \quad (16)$$

To calculate the heat transfer \dot{Q}_{IHX} , the capillary tube is modelled according to the finite volume method. As the refrigerant state is in the two phase region, the temperature can be calculated from the pressure. The pressure in each volume is determined by dimensionless pressure distribution correlation according to Philipp [5].

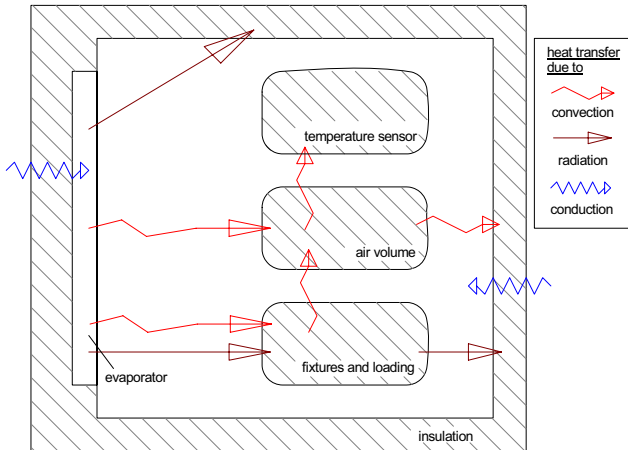


Figure 7: Heat capacities and heat flows in a one temperature zone cabinet

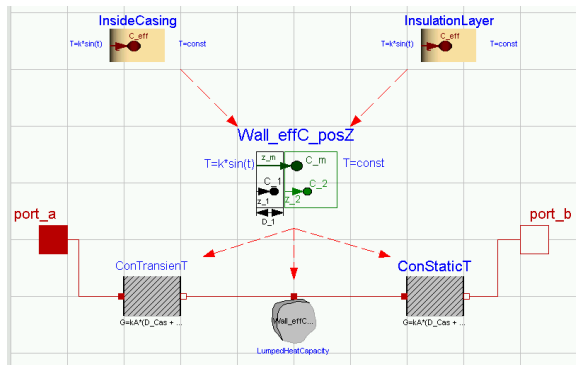


Figure 8: Wall model with calculation of the effective heat capacity

2.4 Cabinet

The cabinet model contains different heat capacity elements which interact via heat transfer due to convection, radiation and conduction. Fig. 7 shows the main heat capacities and heat flows.

For the cabinet wall two models are implemented. In the first model the wall is discretized into a series of heat capacity and conduction elements. The distribution of the parameters has to take the structure of the wall into account, e.g. the inner casing and the insulation.

If the cycling frequency is approximatively known, a simplified model approach can be used. Thus, an effective heat capacity and its position between two heat conduction model is calculated (Fig. 8). By this way the number of state variables can be reduced.

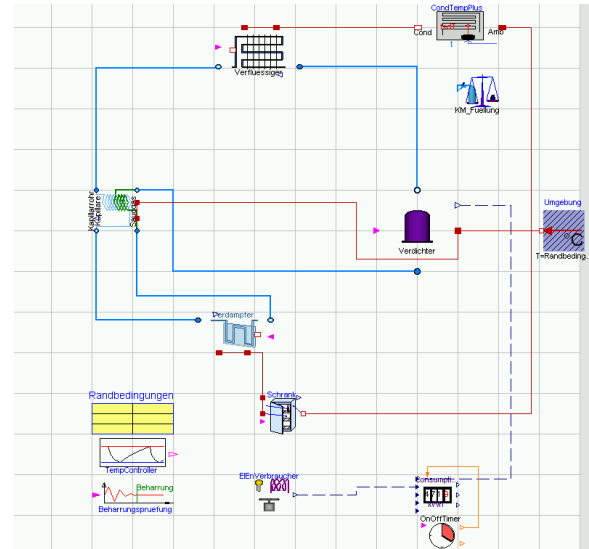


Figure 9: Complete refrigerator model in Dymola

2.5 Refrigerator

All components are connected via fluid ports of the Modelica Fluid Library and heat ports of the Modelica library. Heat ports are separated into convection and radiation heat transfer, if necessary. The cabinet model provides an output signal of the cabinet temperature. Temperature control is done by cabinet and evaporator surface temperature. A complete model is shown in Fig. 9.

2.6 Further Features

To reach a high flexibility, the data inputs for the end-user of each component is put in a replaceable record. The data records are stored as a modification of the basic data record in a top level package outside the main package. This allows fundamental modifications of the components without an impact to the database of the end-user. A data import routine for MS Excel and other databases is under construction.

The component models make high use of the replaceable model feature to allow easy justification of the modelling detail level.

An additional component was developed which allows checking significant variables for steady state. After a selectable number of on-off-cycles in steady state, the component stops the simulation run automatically. Particularly for parameter studies, the computation effort can be decreased efficiently.

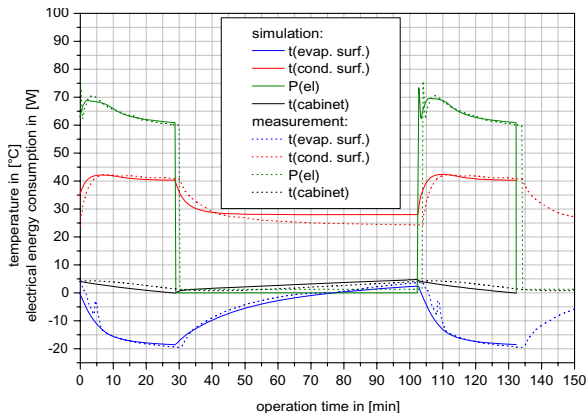


Figure 10: Comparison of a refrigerator system behaviour based on simulation and measurement results

3 Validation

Before the library was used for theoretical investigations, a validation with a one temperature zone refrigerator and a two temperature zone refrigerator - freezer - combination had been performed. Therefore, a series of simulations and measurements with variation of the compressor, the ambient temperatures and the cabinet temperature setting had been done.

Therefore, the ambient temperature where set as a parameter in simulation. The compressor control where done by the temperature output of the cabinet sensor. The parameters of the control hysteresis function of the real refrigerator where applied in simulation to include the influence of thermal inertia of the temperature sensor. For comparison of absolute values (energy consumption, relative compressor operation time, cycling frequency ...) simulation was checked for steady state. Steady state was achieved if all of mentioned values differ less than 0,1 % compared to the previous cycle. For the verification with the measured refrigerator, the average values of three cycles under steady state conditions were applied.

The simulation was in very good agreement in respect of the behaviour of the system and the absolute values like energy consumption and relative compressor operation time. Fig. 10 shows a comparison of simulation and measurement results. Tab. 1 shows the measured and simulated values of energy consumption and relative operation time of the compressor.

The influence of refrigerant sorption in the compressor lubricant increases with decreasing of ambient temperature and of the relative compressor operation time. Fig. 11 shows the dynamic behaviour of saturation temperature in the evaporator and condenser $t_{E,sat}$, $t_{C,sat}$ and the power consumption $P_{E,sat}$ including the

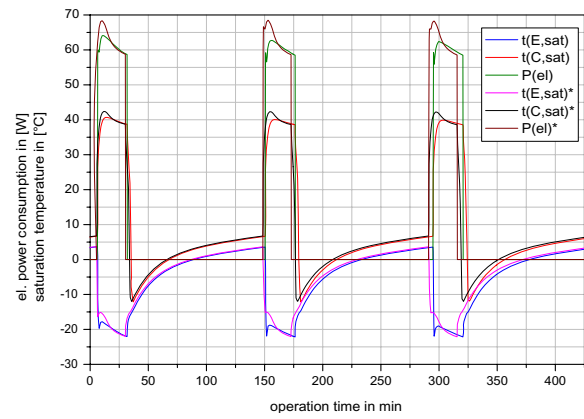


Figure 11: Comparison of simulation including and neglecting the refrigerant sorption

refrigerant sorption and $t_{E,sat}^*$, $t_{C,sat}^*$ and $P_{E,sat}^*$ neglecting refrigerant sorption.

4 Experiences

To reproduce the behaviour in the compressor on-state as well as in the compressor off-state adequately precisely the models have to be very detailed. Some of the physical principles only show an influence on one of the both states and can be neglected in the contrary state.

Usually Dassl and Radau Ila were used as integrators. The Dassl integrator shows the best performance with a relative tolerance between $5 \cdot 10^{-6}$ and $1 \cdot 10^{-5}$. Radau Ila, with a relative tolerance between $1 \cdot 10^{-6}$ and $2 \cdot 10^{-6}$, often provides a faster calculation but simulations lead more often into a stiff system. This is generally a common problem. Further decrease of the relative tolerance does not protect against stiff systems. A high computation effort always happens during the change of the compressor state. Fig. 12 shows the calculation time of Dassl and Radau Ila with Dymola.

The proposed state-depending model structure presented by Nytsch-Geusen [4] can be used to have different models depending of the compressor on and off-state. In the compressor on-state, mass flow depends generally on the compressor and the capillary tube characteristic. In the compressor off-state mass flow rate is caused by thermal influences.

As one example: The heat transfer in the capillary tube - suction line heat exchanger can be neglected in the compressor off-state. The finite volume structure, which is necessary in the on-state, can be simplified to only one mass flow - pressure difference correla-

Table 1: Comparison of absolute values of measurement and simulation results

Compressor	tAmbient	tCabinet	Energy Consumption in kWh/d			Relative Compressor Operation Time		
	in °C	in °C	Simulation	Measurement	Δ (rel.)	Simulation	Measurement	Δ (rel.)
Type 1	39	1,0	0,927	1,002	-7,5%	0,536	0,565	-5,1%
Type 1	25	1,0	0,446	0,480	-7,1%	0,288	0,291	-1,0%
Type 1	25	3,5	0,385	0,407	-5,4%	0,244	0,233	4,7%
Type 2	32	1,0	0,643	0,619	3,9%	0,434	0,409	6,1%
Type 2	25	1,0	0,437	0,413	5,8%	0,314	0,293	7,2%
Type 2	25	3,5	0,349	0,333	4,8%	0,251	0,238	5,5%

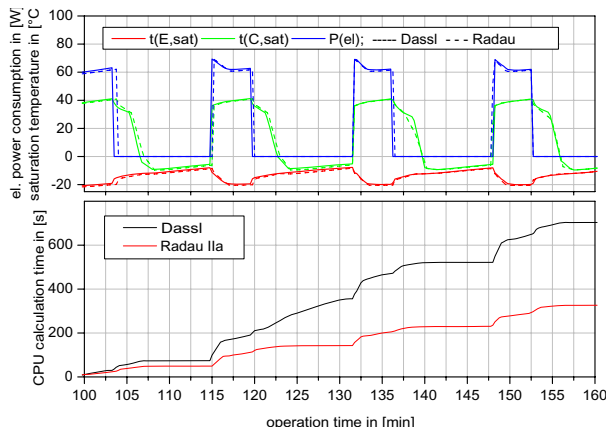


Figure 12: Electrical power consumption, evaporation and condensation temperature and computation time with DASSL and Radau Ila as integrators

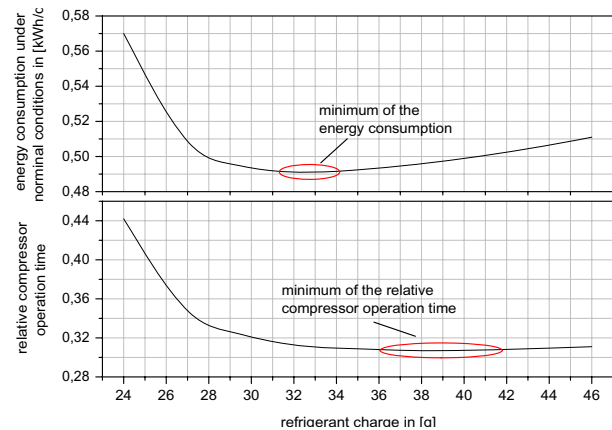


Figure 13: Influence of the refrigerant charge on the energy consumption and the relative compressor operation time

tion with one volume. Hence, calculation effort can be decreased and simulation stability can be increased significantly in the off-state.

5 Issues of Optimization

Effects on different cycle variants, alternative expansion devices and the utilization of stop valves have been investigated with the validated model of the library. Further on, there are some optimization problems which are useful for all refrigerator and freezer models. Getting these results by experimental investigations, means a high effort. Simulation can help to limit that effort efficiently. As examples two optimization problems are presented here. The calculations have been performed on the validated one temperature zone refrigerator.

5.1 Optimal Refrigerant Charge

For a given configuration the energy efficiency varies distinctly from the refrigerant charge. By using a parametric study with Dymola, the optimal refrigerant

charge can be found for known ambient and cabinet setting conditions. Fig. 13 shows a significant minimum of the energy consumption for the optimal refrigerant charge in the range of 33 g. The relative compressor operation time shows a wide minimum which can be found for a higher refrigerant charge. Neglecting the refrigerant sorption leads to a calculation results of optimal charge which is between 3 and 10 g lower, depending mainly on ambient temperature and relative compressor operation time.

5.2 Optimal Compressor

There is a wide choice of hermetic compressors for household refrigerators on the market. The compressor with the best COP under nominal conditions is not necessarily the best under real operation conditions. A specially written script function allows the user to compare a set of different compressors. Fig. 14 shows the energy consumption and the relative compressor operation time for a set of different compressors simulated in one refrigerator. The graph identifies one compressor as the optimal solution for the system. The best

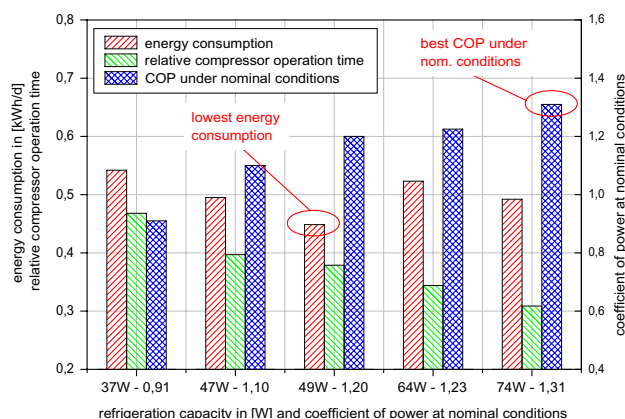


Figure 14: Comparison of energy consumption and relative compressor operation time for a set of compressors

one in the system (COP=1,20) is not the one with the highest COP (1,31).

6 Conclusion and Future Work

A Modelica library for the dynamic simulation of household refrigerators and freezers has been developed. Effects as refrigerant sorption in the compressor lubricant and different void fraction models have been implemented, due to the high influence on the system. The models have been successfully validated and are used for general investigations of the cycle structure as well as for the basic optimization process.

Future work is to be concentrated on the simulation stability, particularly for systems with more temperature zones, as well as for further heat exchanger designs. To obtain a higher stability, the state depended model structure is one issue to work on.

References

- [1] Institut für Luft-und Kältetechnik Dresden. *Stabilität von Kohlenwasserstoffen im Kältemittelkreislauf*. Forschungsrat Kältetechnik e.V., 1999.
- [2] M. Janssen, J. de Wit, and L. Kuijpers. Cycling losses in domestic appliances: an experimental and theoretical analysis. *Int. J. Refrig.*, 15(3):152–158, 1992.
- [3] H. Lippold. Zum Isentropenexponent von Kältemitteln. *Luft- und Kältetechnik*, 12(6):311–313, 1976.
- [4] C. Nytsch-Geusen et al. Mosilab: Development of a modelica base genericc simulation tool supporting model structural dynamics, 2005.
- [5] J. Philipp. *Optimierung von Haushaltskältegeräten mittels numerischer Modellierung*. PhD thesis, TU Dresden, 2002.
- [6] A. Premoli, D. di Francesco, and A. Prina. Una correlazione adimensionale per la determinazione della densità di miscele bifasiche. *La Termodinamica*, 25(1):17–26, 1971.
- [7] R. Radermacher, E. Gercek, V. C. Aute, and Hwang Y. TransRef - Transient Simulation for Refrigeration Systems. Technical report, Center for Environmental Energy Engineering - Univerity of Maryland, 2004.
- [8] K. Stephan. *Wärmeübergang beim Kondensieren und beim Sieden*. Springer Verlag, Berlin, 1988.
- [9] H. Tummescheit, J. Eborn, and F. Wagner. ThermoFluid - A Thermo-Hydraulic Library in Modelica - Documentation. 2001.

A New Energy Building Simulation Library

Juan I. Videla Bernt Lie

Telemark University College

Department of Electrical Engineering, Information Technology, and Cybernetics
Porsgrunn, 3901 Norway

Abstract

Due to the increased interest in saving energy in buildings, new dynamic building models that describe transient response in more flexible modeling languages become necessary. Following some ideas from a previous building thermal behavior library written in Modelica, a new library for modeling building thermal response is presented. The library is meant for system level simulation and its basic elements consist of lumped and one-dimensional distributed parameter models. Object-oriented features like class parameters, inheritance, and aggregation are extensively used to improve the library structure making it easy to read and use. Complex building topologies can be built-up from component models that can be efficiently simulated and studied in any Modelica simulation environment. A comparative test using BESTEST (Building Energy Simulation Test) is performed for inter-program comparison with traditional whole building energy simulation environments.

Keywords: building modeling; building energy simulation; inter-program comparison

1 Introduction

Buildings are complex nonlinear dynamic systems that involve many physical aspects – heat conduction, convective flow, radiation, mass flows, etc. – with hundreds of variables and parameters that must be properly addressed. Due to the complexity and highly coupled nature of these phenomena, simulation seems to be the most cost effective way to study how to improve the energy efficiency in buildings [1]. Although there are many building energy simulation tools available [11], [2], the majority of them are not convenient to easily test and develop advanced controllers and supervisory control techniques.

The hierarchical structure of buildings allows the definitions of different levels of detail naturally. So a building can be divided into different levels going from physical phenomena components to more complex structures like the building itself. A basic elements sublibrary for lumped and distributed parameter models can be defined to represent the underlying physical phenomena. More complex elements are naturally derived from these basic components. Depending on the complexity and the requirements of the problem the models can be adapted and several building topologies can be easily assembled.

The building library, HITLib, is evaluated using BESTEST comparative testing procedure [6] with a special emphasis in the models required to represent the test cases. HITLib is also capable to represent inter-zone convective energy flows in multi-zone buildings using mass and energy balances for the lumped control volumes which represent rooms, and steady-state hydraulic fluid assumption for the bidirectional mass flow components like cracks, doors, ventilations, etc. In the next section, a general overview of the library structure is given. Then, the main building models used in the test cases are developed. The BESTEST inter-program comparison results are presented in section 4. To sum up with the conclusions and future work in the last section.

2 Library overview

The library is organized in 6 main packages (see figure 1)

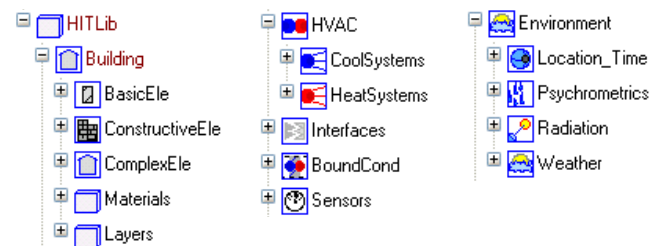


Figure 1: HITLib general library structure.

- Building – Basic, Constructive, and Complex elements; material properties and layers geometry sublibraries
- Environment – Location_Time, Psychrometric, Solar radiation, and Weather sublibraries
- HVAC – Heating, Ventilating and Cooling Systems sublibraries
- Interfaces – all connectors and components used in interconnections.
- BoundCond – Variable and fixed sources and sinks according to the connectors.
- Sensors – Sensor models for the different external interfaces (connectors)

Basic models generally described elementary models like the air heat and mass storage in a control volume, the orifice equation, single and multilayer conductive models, convective models, and radiation exchange models. Many of the

basic model formulations found in HITLib are similar to the ones found in ATPlus library [4], while other ones give a more detailed description of the underlying phenomena (e.g. longwave radiation exchange in the enclosure model, air mass storage, orifice equation, etc.).

Constructive models like a wall model are built by combining basic models. For instance, a wall component addressing radiation, multi-layer conduction and convection is constructed through aggregation and inheritance of basic models. In a similar way, ventilations, doors, windows, roofs, floors, junctions, cracks, etc. can be defined from the basic models.

Complex elements like a room (see Fig. 5) or a storey are defined by aggregation of constructive elements. They are basically template-like components where the subcomponents constructive element parameters are directly defined through the complex element higher level parameters. In this way, the material used in the walls of a room can be directly defined through the higher level room parameters.

Materials subpackage contains records of different material properties. In general parameters are grouped in partial models to increase readability and reuse in similar components at each level.

Environment subpackage contains models and functions to calculate solar radiation and position, perform psychrometric conversions, estimate sky-temperature, and load and interpolate weather data files in tables.

HVAC subpackage contains models representing electrical heaters, radiators, valves, pipes, boilers, pumps, chillers, heat exchangers, etc.

BoundCond and Sensor libraries defined all possible interactions with the connectors used in the different models. Icons are defined in the Icons package and they are inherited in the library models to increase model readability.

3 Building models description

The thermal response of a building space include the heat transfer from the enclosing walls and roofs through conduction and convection, solar heat gains through windows, heat gains or losses due to infiltration, internal long wave exchange and internal short wave absorption and distribution among the enclosure surfaces, and short wave heat gains from internal sources. Additionally, the exterior wall exchange heat with the environment through convection, long-wave radiation, and shortwave absorption. The air within a room is represented through a lumped control volume connected to the internal convective sides of the inner surfaces of the envelope.

3.1 Air volume model

HITLib has three different models to represent the air volume within a room.

An air volume model formulated using a dynamic mass and internal energy balances within a lumped control volume,

$$\begin{aligned} \frac{dm(t)}{dt} &= \sum_j \dot{m}_j(t) \\ \frac{dU(t)}{dt} &= \sum_j \dot{H}_j(t) + \sum_k \dot{Q}_k(t) \\ H(t) &= U(t) + p(t)V(t) \\ H(t) &= m(t)h(t) \\ U(t) &= m(t)u(t) \end{aligned} \quad (1)$$

where m is the mass within the control volume, \dot{m}_j the mass in/outflows, \dot{H}_j the enthalpy in/outflows, \dot{Q}_k the heat in/outflows, U the internal energy, H the enthalpy, p the pressure, V the volume, and h and u , the specific enthalpy and specific internal energy, respectively. The ideal gas law is used since nominal temperatures, pressures, and densities for buildings are well within the range of validity of this thermodynamic model. The ideal gas law is implemented in a partial model class, and used when necessary in other components through inheritance. The preferred states and their initial conditions are then selected, and model reformulation is automatically done by Dymola using its built-in symbolic math capabilities for dealing with DAE (Differential Algebraic Equations):

$$\begin{aligned} p(t)v(t) &= RT(t) \\ u(t) &= u_0 + c_v(T(t) - T_0) \\ h(t) &= u(t) + p(t)v(t) \end{aligned} \quad (2)$$

where, v is the specific volume, T the temperature, u_0 the specific internal energy at T_0 , and R the gas constant. This air volume model is also extended in the AirVolHum model to account the

A more basic air volume model represented as a simple heat storage unit (the model that has been used in the BESTEST analysis) is

$$\dot{Q}_{\text{air}} = c_{p\text{air}} \rho_{\text{air}} V_{\text{room}} \frac{dT_{\text{air}}}{dt} \quad (3)$$

where $c_{p\text{air}}$ is the air specific heat capacity, ρ_{air} is the air density, V_{room} is the air volume, \dot{Q}_{air} is the net heat flow rate entering the air volume, and T_{air} is the air temperature within the volume.

Additionally, a model that accounts the humidity variation within the air volume formulated from a dynamic mass and internal energy balances is available.

3.2 Exterior and interior wall heat transfer models

The heat transfer that takes place at each exterior surface (or roof) i on the outside of building can be calculated using the following heat balance

$$\dot{Q}_{\text{cond},i} = \dot{Q}_{\text{conv},i}^o + \dot{Q}_{\text{radsol},i}^o + \dot{Q}_{\text{radlw},i}^o \quad (4)$$

where $\dot{Q}_{\text{cond},i}$ is the conduction heat flow rate between the external surface and the inner conductive layers, $\dot{Q}_{\text{conv},i}^o$ is the convective heat transfer between the outdoor air and the wall surface, $\dot{Q}_{\text{radsol},i}^o$ is the heat flow rate that the external surface of a wall or roof absorbs from the total incident solar radiation, $\dot{Q}_{\text{radlw},i}^o$ is the heat flow rate from the long wave radiation that the external surface exchanges with the

surroundings, such as outdoor air, the ground and nearby buildings, and the sky.

With

$$\begin{aligned}\dot{Q}_{\text{conv},i} &= h_o (T_{\text{out}} - T_{\text{sur},i}) \\ \dot{Q}_{\text{rad sol},i} &= A_i \alpha_i (I_{\text{dif},i} + I_{\text{dir},i}) \\ \dot{Q}_{\text{rad lw},i} &= \epsilon_i \sigma \left[F_{\text{air}} (T_{\text{out}}^4 - T_{\text{sur},i}^4) + \right. \\ &\quad \left. F_g (T_g^4 - T_{\text{sur},i}^4) + F_{\text{sky}} (T_{\text{sky}}^4 - T_{\text{sur},i}^4) \right]\end{aligned}\quad (5)$$

where h_o is the convective heat transfer coefficient of the external wall surface, T_{out} is the outside temperature, $T_{\text{sur},i}$ is the wall surface temperature, α_i is the shortwave absorptivity, $(I_{\text{dif},i} + I_{\text{dir},i})$ is the total incident solar radiation (direct and diffuse radiation impinging the external surface), A_i is the area of incidence of the solar radiation, ϵ_i is the outer layer emissivity, σ is the stefan-boltzmann constant, F_{air} is the surface-air view factor, F_g is the surface-ground view factor, T_g is the ground temperature, F_{sky} is the surface-sky view factor, and T_{sky} is the sky temperature.

The one dimensional heat conduction through an homogeneous isotropic solid (e.g. layer of a wall) $\dot{Q}_{\text{cond},i}$, with material properties independent of temperature, can be modeled as a one dimensional PDE of the form:

$$\begin{aligned}\dot{Q}_{\text{cond}} &= -k A_{\text{cond}} \partial T / \partial x \\ \partial T / \partial t &= \frac{1}{\rho c} \partial \dot{Q}_{\text{cond}} / \partial x\end{aligned}\quad (6)$$

where T is the temperature, \dot{Q}_{cond} is the conductive heat flow rate, A_{cond} , k , ρ , and c are conductive area, thermal conductivity, density and specific heat capacity of the material, respectively. This equation can be solved using the method of lines (i.e. spatial discretization), see the following Modelica code fragment from the single layer model **CondM1**:

```
model CondM1
...
equation
for i in 2:Ndis loop
  Area*dx*Material.rho*Material.c*der(T[i]) = Q[i - 1] - Q[i];
end for;
for i in 2:Ndis + 1 loop
  Q[i - 1] = (Material.lambda*Area)*(T[i - 1] - T[i])/dx;
end for;
...
end CondM1;
```

The single layer component **CondM1** can be extended to the multilayer case **CondMc** by declaring it n times, each declared single layer component is interconnected using a for clause and the connect command as shown below:

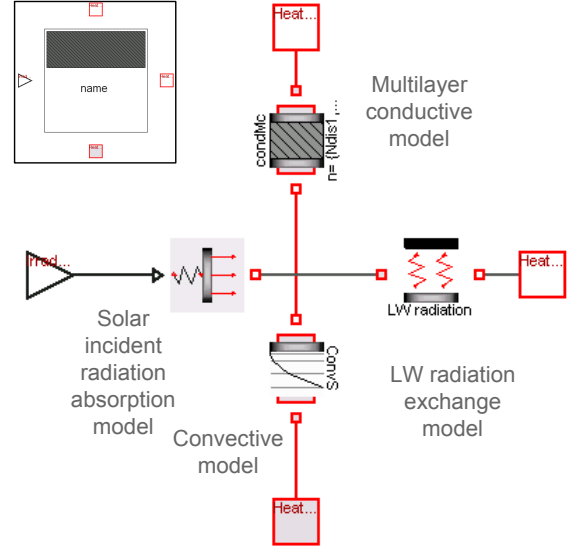


Figure 2: Half exterior roof model icon and diagram layer.

```
model CondMc
...
CondM1 CondM[Nlay](
  Area=Area[1:Nlay],
  Ndis=Ndis[1:Nlay],
  Len=Len[1:Nlay],
  Tini=Tini[1:Nlay],
  Material=Material[1:Nlay]);
equation
connect(CondM[1].Heat_a, Heat_a);
connect(CondM[Nlay].Heat_b, Heat_b);
for i in 1:Nlay - 1 loop
  connect(CondM[i].Heat_b, CondM[i + 1].Heat_a);
end for;
end CondMc;
```

Finally, more complex models (e.g. constructive models like wall, roof, or floor models) can be built aggregating convective models and the multilayer conductive model **CondMc**. As an example see the diagram layer of the half exterior roof model in Fig. 2. The additional models represent the heat flow rate $\dot{Q}_{\text{rad sol},i}$, that the external surface of the roof absorbs from the total incident solar radiation impinging the exterior surface, and the long wave radiation heat flow rate $\dot{Q}_{\text{rad lw},i}$, that the external surface exchanges with the sky for this particular model. This exterior half roof model represents the heat balance at the exterior side of a building roof. This model has azimuth and tilt angle parameters to properly define the total solar radiation. Similarly, other constructive models can be defined by aggregation of elementary models and class parameters of the inner models.

A similar heat balance can be formulated for each building wall (or roof) interior surface i within the building

$$\dot{Q}_{\text{cond},i} + \dot{Q}_{\text{conv},i} + \dot{Q}_{\text{rad sw},i} + \dot{Q}_{\text{rad lw},i} = 0 \quad (7)$$

where $\dot{Q}_{\text{cond},i}$ is the conductive heat transfer between the wall inner layers and the internal wall surface of the room,

$\dot{Q}_{\text{conv},i}^j$ is the convective heat transfer between the internal air and the wall surface, $\dot{Q}_{\text{radsw},i}^j$ is the heat flow rate that the internal outer surface of a wall or roof would absorb from any short wave source (e.g. incoming solar radiation from the window model or short wave radiation from a bulb model), and $\dot{Q}_{\text{radlw},i}^j$ is the net heat flow rate due to the long wave radiation exchange among the internal surfaces.

$$\begin{aligned}
\dot{Q}_{\text{conv},i} &= h_i (T_{\text{int}} - T_{\text{sur},i}) \\
\dot{Q}_{\text{radsw},i} &= Fa_i \dot{Q}_{\text{wind}} \\
\dot{Q}_{\text{radlw},i} &= A_{\text{sur},i} \frac{\epsilon_i}{1 - \epsilon_i} \left(\sigma T_{\text{sur},i}^4 - J_{\text{sur},i} \right) \\
\dot{Q}_{\text{radlw},i} &= A_{\text{sur},i} \sum_{j=1}^N F_{i,j} (J_{\text{sur},i} - J_{\text{sur},j})
\end{aligned} \tag{8}$$

where h_i is the convective heat transfer coefficient of the internal wall surface i , T_{int} is the air volume temperature, Fa_i is the internal solar distribution fraction (for details see Kreith and Bohn (1993)), \dot{Q}_{wind} is the heat flow rate from the incoming solar radiation into the enclosure through the windows, $J_{\text{sur},i}$ is the radiosity of surface i , $J_{\text{sur},j}$ is the radiosity of surface j , $F_{i,j}$ is the view factor between surfaces i and j , and N is the number of enclosure surfaces.

The long wave radiation exchange among the surfaces of the enclosure has been implemented in the **enclosure** model. The conventional radiation heat transfer network analysis method [10] has been used assuming plain gray diffuse surfaces. The **enclosure** model is included in the **room42_B** model and each radiating surface is separately connected to the enclosure model using a connector array. The view factors depend on the geometry, position, and surface emissivity of each of the inner surfaces of the enclosure, they can be precalculated and loaded or an area weighted approach can be directly calculated in the component.

The shortwave radiation distribution is implemented with a model that uses precalculated solar fractions according to the geometry and absorptivity of the internal surfaces.

3.3 Windows heat transfer model

The window model (see Fig. 3) is also built from basic models by aggregation, the heat transfer through a window can be approximated by three parts,

- a conductive-convective part due to the temperature difference between the exterior and interior of the building,
- a solar heat gain model due to the solar radiation,
- and a convective energy flow due to infiltration.

The Window model used to evaluate HITLib with BESTEST do not include any infiltration or ventilation component driven by pressure differences between zones. The building infiltration is directly represented using a simpler ASHRAE model in the room model.

The conductive-convective heat transfer part due to the temperature difference between the exterior and interior of the building is modeled using two convective models for the external surfaces at both sides of the windows, and a conductive model representing the glass pane. This part of the Window model represents the rate of heat transfer through

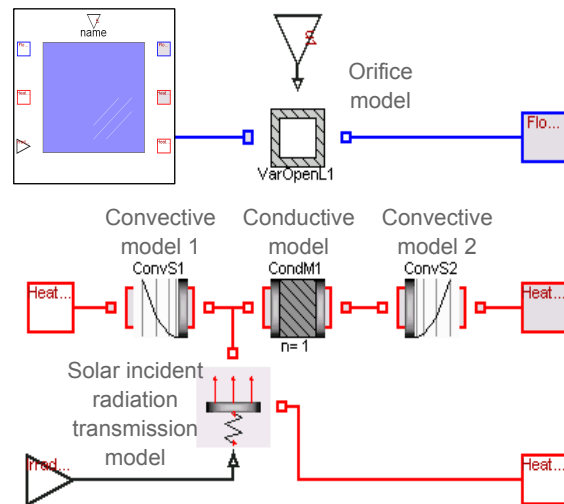


Figure 3: Window model icon and diagram layer.

the windows in the absence of sunlight and air infiltration. Multi-glass windows can be approximated by this simplified model or otherwise, the model can be easily extended to account for a multi-pane window. The total solar heat flow rate through the glass is calculated with the solar incident radiation transmission model.

3.4 Infiltration heat transfer model

The infiltration heat transfer model represents the building's sensible thermal load due to air exchange by infiltrations. The incoming air must be heated or cooled depending on the temperature difference between the outdoor and indoor air. The heat flow rate due to this sensible heating or cooling is given by

$$\dot{Q}_{\text{inf}} = I_{\text{ACH}} V_{\text{room}} \rho_{\text{air}} c_{p\text{air}} (T_{\text{out}} - T_{\text{int}}) \quad (9)$$

where \dot{Q}_{inf} is the sensible heat load, ρ_{air} is the air density, $c_{p\text{air}}$ is the specific heat of air, V_{room} is the room volume, I_{ACH} is the air exchange rate also called air changes per hour (ACH), T_{out} is the outside temperature, and T_{int} is the indoor temperature. The infiltration model can be seen in the diagram layer of the **room42 B** model in Fig. 5

3.5 Environment model

The environment model calculates the sun position and solar radiation with respect to and over each exterior building surface (for a detailed description of the following equations see [5], [3], [9], and [8]). The environment model diagram layer used with the BESTEST simulations can be seen in Fig. 4. Subcomponents (1), (2), (3), and (4) are used to calculate the solar time t_{sol} , the hour angle ω , and the solar declination angle δ (angle between the earth's equatorial plane and the sun beam in the selected location),

$$t_{sol} \triangleq t_{std} + \frac{L_{std} - L_{loc}}{15^\circ/h} + \frac{E_t}{60 \text{ min/h}} \quad (10)$$

$$\omega = \frac{\pi(t_{sol} - 12 \text{ h})360^\circ}{24 \text{ h}180^\circ} \quad (11)$$

$$E_t = 9.87 \sin(2a) - 7.53 \cos(a) - 1.5 \sin(a) \quad (12)$$

$$a = \frac{360^\circ(n-81)}{364}$$

$$\sin(\delta) = -\sin\left(\frac{23.45^\circ\pi}{180}\right) \cos\left(2\pi\frac{n+10}{365.25}\right) \quad (13)$$

where t_{sol} is the solar time, t_{std} is the standard time, L_{std} is the standard longitude, L_{loc} is the local longitude, E_t is the equation of time, and n is the day of the year.

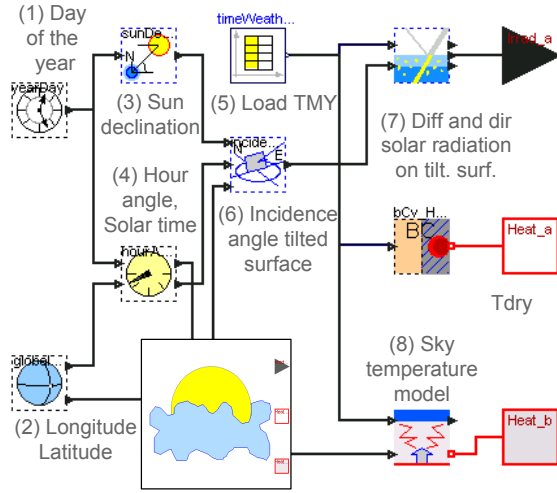


Figure 4: Environment model used with BESTEST analysis.

Subcomponent (6) in Fig. 4 calculates the solar incidence angles θ_i for arbitrary planes i in terms of the tilt θ_{sur} and the azimuth ϕ_{sur} angles of the surface normal,

$$\begin{aligned} \cos\theta_i &= \cos\theta_{sur}(\cos\delta\cos\omega\cos\lambda_{lat} + \sin\delta\sin\lambda_{lat}) \\ &\quad + \sin\theta_{sur}\sin\phi_{sur}\cos\delta\sin\omega \\ &\quad + \sin\theta_{sur}\cos\phi_{sur}(\cos\delta\cos\omega\sin\lambda_{lat} \\ &\quad - \sin\delta\cos\lambda_{lat}) \end{aligned} \quad (14)$$

where λ_{lat} is the building latitude (obtained from subcomponent (2) in Fig. 4). In the current implementation the incidence angle θ_i is calculated for the following surface tilt angles $\theta_{sur} = (0^\circ, 45^\circ, 90^\circ)$ that correspond to horizontal surface, 45° tilted surface, and a vertical surface; and for the following surface azimuth angles $\phi_{sur} = (180^\circ, 135^\circ, 90^\circ, 45^\circ, 0^\circ, -45^\circ, -90^\circ, -135^\circ)$ that correspond to North, North-east, East, South-East, South, South-west, West, and North-West directions. In this way 17 different incidence angles are obtained for the 24 different combinations of surface tilt and azimuth angles¹.

The Load TMY model (see subcomponent (5) in Fig. 4) loads the hourly TMY weather data file into a table. The

¹for $\theta_{sur} = 0^\circ$, any ϕ_{sur} will have the same θ_i value.

table entries include the dry temperature T_{dry} , the dew point temperature T_{dp} , the atmospheric pressure p , the relative humidity H_{rel} , the wind velocity W_{vel} , the wind direction W_{angle} , the global horizontal radiation $H_{glo,hor}$, the diffuse horizontal radiation $H_{dif,hor}$, the total sky cover n_{to} , the opaque sky cover n_{op} and the ceiling height h .

For every incidence angle θ_i , the diffuse solar radiation over a tilted surface $H_{dif,til,i}$ and the direct solar radiation over a tilted surfaces $H_{dir,til,i}$ are calculated from the hourly global horizontal radiation $H_{glo,hor}$ and the hourly diffuse horizontal radiation $H_{dif,hor}$,

$$H_{dir,hor} = H_{glo,hor} - H_{dif,hor} \quad (15)$$

$$H_{dir,til,i} = \frac{H_{dir,hor}}{\cos\theta_s} \cos\theta_i \quad (16)$$

$$\begin{aligned} H_{dif,til,i} &= H_{dif,hor} \frac{1 + \cos\theta_{sur}}{2} \\ &\quad + H_{glo,hor} \rho_{gro} \frac{1 - \cos\theta_{sur}}{2} \end{aligned} \quad (17)$$

where ρ_{gro} is the ground reflectance.

Finally the sky temperature model (see subcomponent (8) in Fig. 4) based on [7] model estimates the sky temperature T_{sky} . First the clear sky emissivity ϵ_{clear} is calculated from the dew point temperature T_{dp} , the solar time t_{sol} , the atmospheric pressure P as

$$\begin{aligned} \epsilon_{clear} &= 0.711 + 0.56 \left(\frac{T_{dp}}{100}\right) + 0.73 \left(\frac{T_{dp}}{100}\right)^2 + \\ &\quad 0.013 \cos\left(\frac{2\pi t_{sol}}{24}\right) + 0.00012(P - 1000) \end{aligned} \quad (18)$$

the sky emissivity ϵ is calculated from the clear sky emissivity and the cloud amount C , which is estimated using the following equations,

$$\epsilon = \epsilon_{clear} + (1 - \epsilon_{clear})C \quad (19)$$

$$C = \sum n_i \epsilon_{c,i} \Gamma_i \quad (20)$$

$$\ln \Gamma_i = -h_i/h_o \quad (21)$$

where n_i is the sky cover fractional area for each cloud layer i , $\epsilon_{c,i}$ is the hemispherical cloud emissivity for each cloud layer i , Γ_i is a factor that depends on the cloud base temperature for each cloud layer i and can be approximated by equation 21 where h_i is the cloud layer base height and h_o is equal to 8200m. Finally, the black-body sky temperature is computed as

$$T_{sky} = \epsilon^{\frac{1}{4}} T_{db} \quad (22)$$

3.6 Room model

The **room42_B** model used to represent the BESTEST cases can be seen in Fig. 5. Several of the already presented components are interconnected in the room model. The central component is the air control volume component, modeled as a simple heat storage model. The half interior walls, roof, and floor model convective sides are connected to the air volume. The longwave radiation exchange

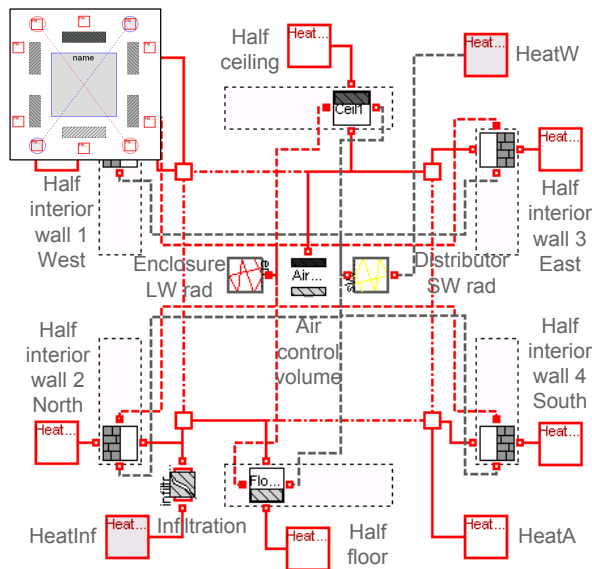


Figure 5: Diagram layer of room42_B model.

heat flow rate is calculated in the Enclosure LW component. The heat flow rate due to the solar radiation transmitted by the windows enters into the room model through the special purpose heat connector HeatW and it is distributed in the enclosure by the Distributor SW rad model. The room model also has an infiltration subcomponent connected to the special purpose heat connector HeatInf.

4 BESTEST approach

Comparative testing is used to compare the performance of the HITLib library with the standard building simulation tools. The BESTEST approach [6] consists of a set of carefully specified cases (36 cases) for software-to-software comparisons, and program diagnostics that can be easily defined over a variety of detailed and simplified whole-building energy simulation programs. The cases go from simple to complex, and only 14 of them are for qualification. Qualification cases 600 to 650 and 900 to 990 represent a set of lightweight and heavyweight buildings that are relatively realistic with respect to their thermal characteristics. The basic geometry of the test case building is a simple rectangular single zone with no interior partitions (see Fig. 6). These test cases are implemented and evaluated with the Modelica building library and the corresponding results are compared with those of the reference whole building energy simulation tools (i.e. BLAST, DOE2, ESP, SERIRES, S3PAS, SUNCODE, TASE, and TRNSYS).

Since non shading device has been modeled cases 610, 630, 910, and 930 can not be properly modeled. Cases 650, 950 were not tested since no scheduled ventilation model was implemented and cases 960 and 990 were neither evaluated. The tested cases (see table 1) allow us to analyze the performance of the library in the aspects of basic heat transfer (case 600), east and west incidence and/or transmittance so-

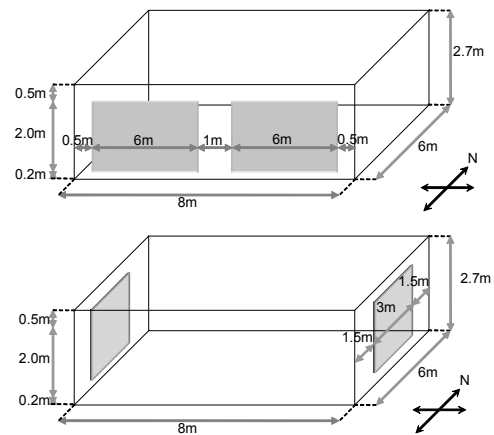


Figure 6: Basic geometry of BESTEST building with south windows (above) and with east and west windows (below).

lar radiation (620 & 620-600) and setback control (640 & 640-600) for the light weight cases, and mass and/or south solar interaction (900 & 900-600), mass and/or east-west solar interaction (920 & 920-900), and mass and/or setback interaction (940 & 940-900) for the heavy weight cases.

A typical meteorological year (TMY) weather data file is provided to perform the tests, the ground modeling is not generally good, therefore a highly insulated floor has been defined in the test cases to effectively decouple the floor thermally from the ground. A ground temperature of 10 °C is applied through a heat boundary condition component (see BC1 subcomponent in Fig. 7). The infiltration component automatically correct its value according to the atmospheric pressure. The internal generated heat gains from equipment, lights, people, animals, etc. that are not related to HVAC correspond to sensible heat (no latent) and they have a value of 200W for the test cases (60% radiative, 40% convective, 100% sensible, 0% latent). The convective and radiative part of the internal gain have been implemented with heat boundary condition components (see BC2 and BC3 subcomponents in Fig. 7), where the first is directly connected with the air mass volume (inside the room model) while the second has been connected to a shortwave distributor model (inside the room model). For the shortwave internal solar distribution fractions the values

Case	Mass	Thermostat
600	Light	Deadband or 20,27
620	Light	Deadband or 20,27
640	Light	Setback
600FF	Light	free-float
900	Heavy	Deadband or 20,27
920	Heavy	Deadband or 20,27
940	Heavy	Setback
900FF	Heavy	free-float

Table 1: BESTEST qualifying cases tested

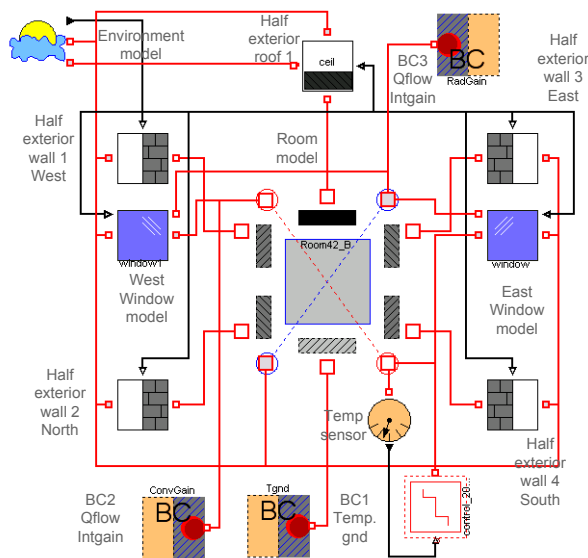


Figure 7: Dymola diagram layout of BESTEST cases 620 and 920 with east/west windows.

Deadband	Heat=on if temp<20C
or 20,27	Cool=on if temp>27C
	23:00<time<07:00
	heat=on if temp <10C
Setback	07:00<time<23:00
	heat=on if temp <10C
	cool=on if temp>27C

Table 2: BESTEST heating and cooling thermostat control system

provided by BESTEST are used.

The heating and cooling systems are 100% convective, the thermostat is sensing only the air temperature and it is assumed that there are no latent loads. Both heating and cooling capacities are of 1000kW and they have an effective efficiency of 100%. Two thermostat control strategies has been implemented (see Table 2),

4.1 Results

There are two types of outputs, the annual outputs defined over the whole reference year and daily hourly outputs defined over specific days of the year for certain tests.

Annual type outputs

- Annual heating and cooling loads (MWh) for all non free-float cases, see Fig. 8.
- Annual hourly peak heating and cooling loads (kW) for all non free-float cases, see Fig. 9
- Difference among annual thermal loads among the non free-float cases, see Fig. 10.
- Difference among annual hourly peak thermal loads among the non free-float cases, see Fig. 11.

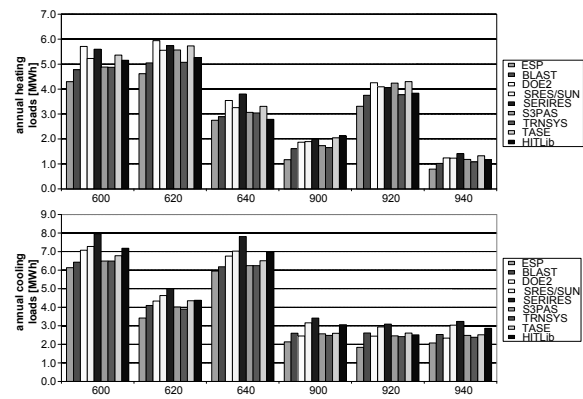


Figure 8: Annual heating loads for low mass buildings - cases 600, 620, and 640- and high mass buildings - cases 900, 920, 940- (above). Annual cooling loads for the same cases (below).

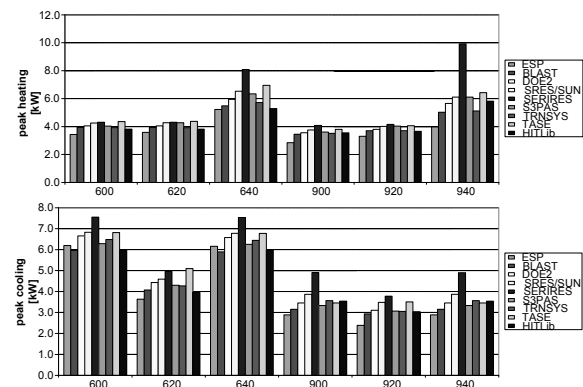


Figure 9: Annual peak heating loads for low mass buildings - cases 600, 620, and 640- and high mass buildings - cases 900, 920, 940-(above). Annual peak cooling loads for the same cases (below).

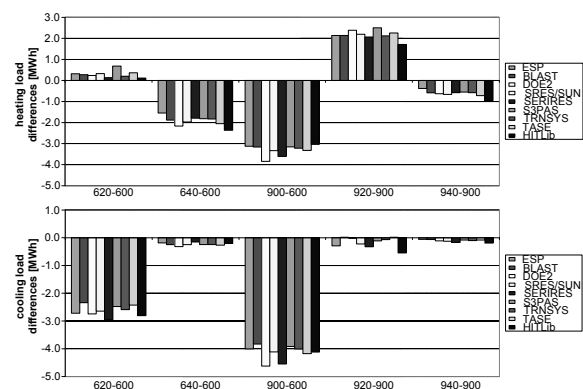


Figure 10: Annual heating load differences among the cases (above). Annual peak cooling load differences among the same cases (below).

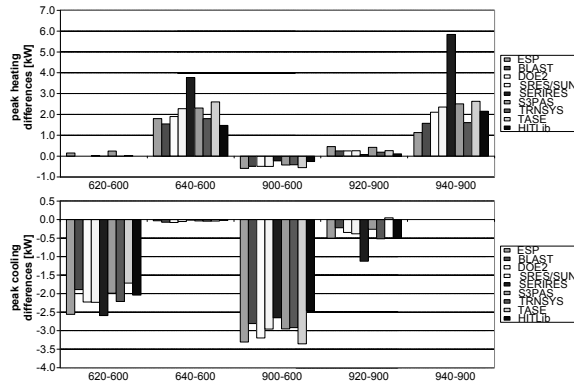


Figure 11: Annual peak heating load differences among the cases (above). Annual peak cooling load differences among the same cases (below).

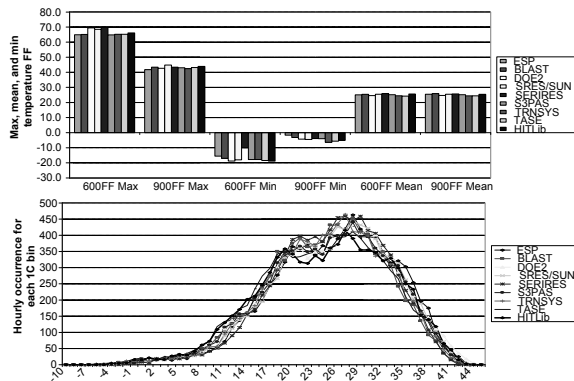


Figure 12: Maximum, mean, and minimum temperatures for the free floating cases 600FF and 900FF (above). Case 900FF temperature histogram with bins of 1C (below).

- Annual hourly-integrated maximum, minimum and average indoor air temperature for all free-float cases and annual hourly-integrated indoor air temperature histogram for free-float case 900FF, see Fig. 12.

Annual hourly-integrated maximum, minimum and average indoor air temperature for all free-float cases show a good level of coincidence as well as the annual hourly-integrated indoor air temperature histogram for free-float case 900FF.

Annual heating and cooling loads for all non free-float cases show a reasonable agreement between the reference values and the values obtained with HITLib. Annual heating load for case 900 and annual cooling load for case 920 are slightly out of the range of the reference values, this does not pose a significant problem since in general for the high mass cases (series 900) there is an expected higher variability among the reference outputs. The difference among the annual thermal loads among the non free-float cases show also a good level of agreement with the reference values.

Annual hourly peak heating and cooling loads (kW) for all non free-float cases show a good level of agreement with

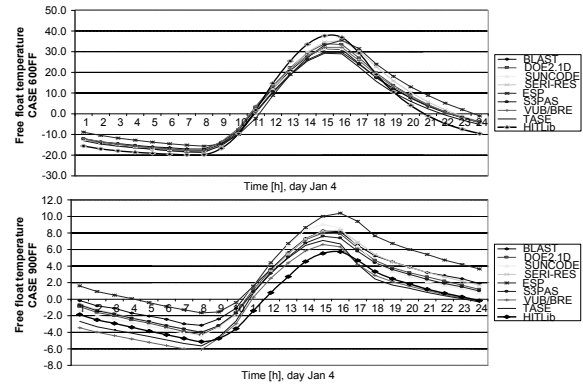


Figure 13: Hourly free float temperatures on January 4 for the case 600FF (above). Hourly free float temperatures on January 4 for the case 900FF (below).

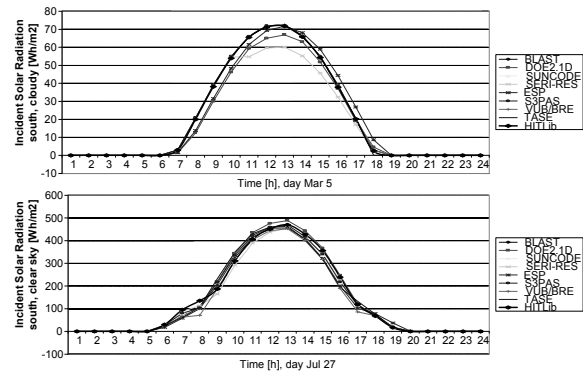


Figure 14: Hourly solar incident radiation over a south facing surface in a cloudy day, March 5 (above). Hourly solar incident radiation over a south facing surface in a clear day, July 27 (below).

the reference values and the time when they occur². Difference among annual hourly peak thermal loads among the non free-float cases also show high coincidence with the reference values.

Daily hourly outputs

- Hourly free-float cases indoor air temperature for 600FF and 900FF (January 4), see Fig. 13.
- Hourly unshaded incident solar radiation on south surfaces in a cloudy day (March 5) and a clear sky day (July 27), see Fig. 14.
- Hourly unshaded incident solar radiation on west surface in a cloudy day (March 5) and a clear sky day (July 27), see Fig. 15.
- Hourly heating and cooling (kWh) for cases 600 and 900 (January 4), see Fig. 16.

The HITLib daily hourly outputs have a high matching with the BESTEST reference values.

²this is not presented here

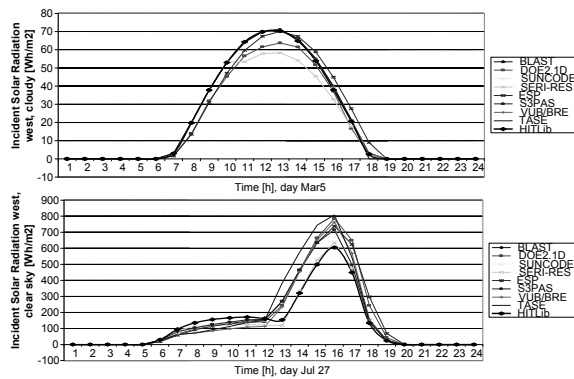


Figure 15: Hourly solar incident radiation over a west facing surface in a cloudy day, March 5 (above). Hourly solar incident radiation over a west facing surface in a clear day, July 27 (below).

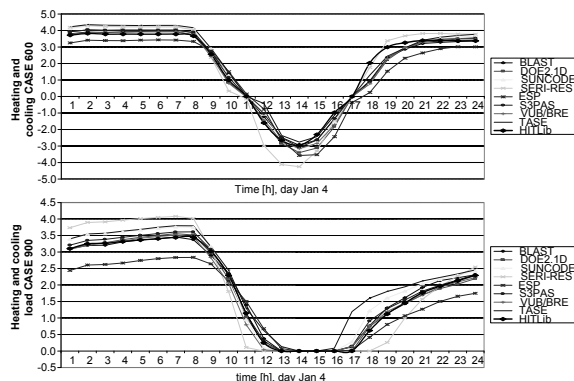


Figure 16: Hourly heating and cooling loads on January 4 for case 600 (above). Hourly heating and cooling loads on January 4 for case 900 (below).

5 Conclusions

The final goal of the library is to give intuitive blocks for modeling the building thermal response to test and develop advanced controllers for system level simulation. Modelica language seems to be an adequate language for this purpose, allowing a simple equation-based formulation of well-known building models.

BESTEST method only compares certain aspects of an energy representation of a building and many models of the new library have not been presented since they can not be evaluated (e.g. interzone convective heat flow). Additionally, not all the BESTEST qualification cases have been tested since some of them are of no interest for the library purpose (e.g. hangout shading). Nonetheless, a quality measurement in the aspects studied in the test cases can be accepted as a good starting point in the validation procedure of the library.

Many of the models used for the BESTEST are still under development but the library seems to lack of any significant modeling error. Particularly, the window model is still un-

der revision and a more refined model addressing the multi-layer glazing cases will be developed.

The library is still under-development and many improvements are foreseen and it will be freely available in the upcoming months.

References

- [1] J. A. Clarke, *Energy Simulation in Building Design*. Butterworth-Heinemann, 2001.
- [2] D. B. Crawley, J. W. Hand, M. Kummert, and B. T. Griffith, "Constrasting the capabilities of building energy performance simulation programs," US Department of Energy, ESRU University of Strathclyde, SEL University of Wisconsin-Madison and National Renewable Energy Laboratory, Tech. Rep., 2005.
- [3] J. A. Duffie and W. A. Beckman, *Solar Engineering of thermal processes*. Jhon Wiley and Sons, Inc., 1991.
- [4] F. Felgner, S. Agustina, R. C. Bohigas, R. Merz, and L. Litz, "Simulation of thermal building behavior in modelica," in *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, March 2002, pp. 147–154.
- [5] M. Iqbal, *An introduction to solar radiation*. Academic press canada, 1983.
- [6] R. Judkoff and J. Neymark, "International energy agency building energy simulation test (bestest) and diagnostic method," National Renewable Energy Laboratory, Tech. Rep., 1995.
- [7] M. Martin and P. Berdhal, "Characteristics of infrared sky radiation in the united states," *Solar Energy*, vol. 33, pp. 321–336, 1984.
- [8] T. Muneer, *Solar Radiation and Daylight Models for the Energy Efficient Design of Buildings*. Architectural press, 1997.
- [9] A. Rabl, *Active Solar Collectors and their applications*. Oxford University press, 1985.
- [10] R. Siegel and J. R. Howell, *Thermal Radiation Heat Transfer*, 3rd ed. Hemisphere Publishing Corporation, 1992.
- [11] C. P. Underwood and F. W. H. Yik, *Modelling Methods for Energy in Buildings*. Blackwell Publishing, 2004.

UnitTesting: A Library for Modelica Unit Testing

Dr. Michael M. Tiller¹
Emmeskay, Inc.
Plymouth, MI USA
mtiller@emmeskay.com

Burit Kittirungsi
Emmeskay, Inc.
Plymouth, MI USA
burit@emmeskay.com

Abstract

One of the challenges when attempting to deploy model based development processes in industry is the “distrust” that many engineers have of models. It is often the case that engineers favor experiments and data over models and simulation results. The typical reason given for this distrust is that engineers often feel they can “trust” information collected from actual hardware during an experiment more than they can trust simulation results.

There is some validity in this notion. In many ways, modeling is as much an “art” as it is a science. The process of developing models involves creativity and many subjective decisions. In order for modeling to be thought of as a “science”, it needs to be viewed as more objective and analytical. The process of developing models must mature from an ad hoc process into a process that includes formal validation, verification and quality assurance processes just like any other “product”.

This paper describes the UnitTesting library which is a commercial Modelica library offered by Emmeskay, Inc. The UnitTesting library supports automated testing of Modelica models to ensure, among other things, the ongoing validity and accuracy of the models.

Keywords: testing, quality, coverage

1 Background

Although not widely recognized or appreciated, model development closely parallels software development. This is particularly true for model development in Modelica because, in addition to the fundamental mathematical constructs required in a modeling language, Modelica has incorporated many software engineering principles into the language design (*e.g.* encapsulation, interface compatibility, *etc.*).

For this reason, many software development methodologies can be applied to model development as well. A recent trend in software development has been the adoption of testing methodologies to improve quality. These methodologies involve codifying requirements and automatically testing those requirements. One example of this is the JUnit library[1] developed by Gamma and Beck which can be used to automatically test Java classes. Most previous efforts at applying testing to model based control system development have focused on embedded code[2] and perhaps a few invariant physical or mathematical properties[3]. In this paper we propose a generic approach that handles testing for all types of Modelica models whether their behavior is continuous, discrete or a mixture of the two.

The purpose of unit testing is to identify problems at their source. This is done by using carefully designed test cases to immediately identify and isolate model errors. One convenient way of identifying errors as they are introduced into existing models is to compare results using the new model versions with baseline results from previous versions.

In order for this kind of testing to be effective, it needs to be automated and focused. For example, lacking the appropriate tools to automate the comparison process some developers may have to rely on visual inspection. In such cases, the testing process becomes tedious and error prone. Furthermore, test cases themselves must be carefully chosen so that failures are meaningful and help to isolate errors. For example, if only a few complex models are used as test cases, it will be difficult to trace the source of the problem. Without this degree of automation and focus, identifying errors is like trying to identify a poorly tuned instrument in a large orchestra. Assuming you are astute enough to notice the problem, simply detecting the problem is not sufficient to identify the source of the problem. If, on the other hand, you were to analyze each individual instrument (or even small clusters of them) you would be able to isolate

¹ All correspondence should be directed to the first author.

the source of the problem much faster. The `UnitTesting` library provides not only a framework for creating test cases but also includes automation and reporting tools.

2 Use Case

In this paper, the use case we consider is that of a library developer who wishes to test a specific library (and only that library). Under these assumptions all models fall in to one of three categories.

The first category is *library components*. These are the components of the library being tested (*i.e.* the components that the user of the library would drag and drop into their models). The next category are *non-library components*. These are component models from libraries other than the library being tested². For example, components from the Modelica standard library would generally be non-library components (unless, of course, you were a Modelica Standard Library developer testing the Modelica Standard Library). The last type of model is a *test case*. The test cases are models that instantiate library and non-library components and exist only to test library components (*i.e.* they are not considered library components).

Before proceeding, we need to describe a few more details about these test case models. The principle behind a test case is that it is a model that is instrumented to compare a simulated result with some *expected result*. How these expectations are represented will be discussed in greater detail later. The important point is that the test case model itself includes information about its expected result. When the test case is simulated it can report whether the expected result was achieved or not. In the next section, we will commonly refer to the case where a test case *fails*. This is what happens when the test case is simulated and fails to match *all* the expected results.

3 Metrics

When it comes to model quality and testing, we cannot convey the state of a model library as a single number. Instead, we depend on a variety of metrics to assess different desirable properties. In this section, we will introduce several different metrics and the issues that they address.

² The same model could be a library component in one circumstance and a non-library component in other (depending on what library was being tested).

For all of the metrics presented in this section, it is possible to do at least some basic calculation of the metric using only the Modelica source code (*i.e.* without the need to run any simulations). However, in many cases the basic metrics and generated reports are enhanced (sometimes considerably) by access to simulation results. If available, the `UnitTesting` tools will use such results to conduct a more thorough analysis.

3.1 Component Coverage

Component coverage measures what percentage of the library being tested is actually “covered” by the test cases for that library. As a consequence of this analysis, library components that are never tested can be identified. The premise behind this metric is that if a component never appears in a test case it will be impossible to identify errors in that component.

The value of this metric is computed as follows. First, we consider each library component in turn and determine whether it appears in any of the test cases. We say that the library component appears in a test case if it is instantiated (in the Modelica sense) as part of the instantiation of that test case.

To evaluate the coverage metric for a library we simply divide the number of library components that appear in a test case by the total number of library components. A result of 100% indicates that the library has 100% component coverage. Said another way, every component in the library appears in at least one test case.

This metric is the most basic metric. Getting 100% component coverage is simply the first step in establishing a robust testing process. In principle, 100% component coverage implies that a mistake in a single component cannot “slip through” but will instead be detected via the failure of at least one test case. However, as we shall shortly see this is a very optimistic assumption.

3.2 State Coverage

Earlier, we introduced the notion of an expected result. When building a test case model, many variables are involved. It is generally not sufficient to validate a model based only on the value of one variable in that test case.

So the question then becomes “what variables should be checked?”. State coverage is one way to address this issue. It is quite common in Modelica to have *alias variables*. These are variables that have a

unique name in the instance hierarchy but are mathematically identical to other variables. In fact, due to the semantics of connections the vast majority of variables in a large Modelica model are typically alias variables. Obviously, there is no sense in confirming the value of all these redundant variables. Instead, we want to focus on the unique signals. But even these are often computed from each other. For example, if we have the equations “ $x=y+5$ ”, we don’t necessarily need to check both x and y .

To minimize the set of signals to check while maximizing the chance of catching an error in the model we should focus on the state variables. For our purposes, we define the state variables in the model as those variables that uniquely define the state of the model (and from which all unique continuous and discrete signals must therefore be calculated).

Precisely determining the state variables in a Modelica model can be quite tricky. In fact, state selection is typically part of the compilation process of a Modelica model and this process is non-trivial. Fortunately, for our purposes we can accept “false positive” results. So we employ a few basic rules that help us to identify all the *potential* states. It may turn out that these states are later eliminated via constraint equations (*i.e.* the states are not, in fact, independent) but that is not so important for our purposes and only results in a slight amount of redundancy in our checking.

We consider a variable a state if it is either assigned a value in a when clause (*i.e.* it is a discrete variable) or the variable appears within the `der(...)` operator. As mentioned previously, such variables are not necessarily actual states but this substantially narrows the set of variables to consider and ensures that we will at least catch all potential states. Said another way, to achieve state coverage it is sufficient to identify the potential state variables but not necessary to identify the actual state variables.

The state coverage of a test case is determined by taking the the number of states for which expected results are provided and dividing that by the total number of states in the model. Since the set of states can be automatically determined *a priori* it is possible to analyze a test case and automatically instrument it with all the necessary expected results in order to achieve 100% state coverage. Note that we take a very conservative position here that the expected results must refer directly to these states (and not to aliases) in order to be considered “covered”³.

3 An alternative approach would be to determine all alias variables of the potential states and see if they

It should be noted that it might also be useful to apply “input coverage” and “output coverage” to a test case as well. That is to say that expected “results” are supplied for all top level input and output signals and all parameters in the model. The combination of input, output and state coverage helps ensure that any issues that arise during testing are truly due to behavioral changes in the library components and not due to changes in the values of test case parameters or simulation conditions. However, the UnitTesting library does not currently address input and output coverage.

3.3 Decision/Condition Coverage

Previously we mentioned the conditions under which 100% component coverage can be achieved. The problem with the component coverage metric is that it assumes that simply instantiating a component is sufficient to test all functions of that component. While this is true of very simple models, it is not true in general. A more complete metric is “condition/decision coverage” [4].

The goal of condition/decision coverage is to verify that logical aspects of the models are completely exercised. To achieve decision coverage we must verify that every decision being made in the code is “exercised” (*i.e.* for each decision both a “true” and “false” result is generated). Coming up with test cases that achieve complete decision coverage is challenging (and in some cases, simply not possible). It should be noted that the UnitTesting library only evaluates coverage *a posteriori* and does not provide functionality to identify how uncovered decisions can be covered.

Decisions can be compound expressions involving many individual boolean values. For example in the statement:

```
if a>10 and b<5 then
```

the *decision* is based on the value of the entire expression “ $a>10$ and $b<5$ ” but this decision contains two individual *conditions*, e.g. “ $a>10$ ” and “ $b<5$ ”. Even if both outcomes of the decision have been covered in a test case, this does not mean that every condition has been verified. For example, imagine b always had a value of 3. In that case all decisions would be covered as long as a had a value both higher and lower than 10. But this does not test whether the condition “ $b<5$ ” was implemented correctly (e.g. “ $b<6$ ” would work just as well).

were being compared against expected results but we do not currently support that.

For this reason, condition coverage expands the notion of decision coverage to include all possible values of every condition. However, it is not necessary to exhaustively test all conditions. Instead, something called “modified condition/decision coverage” is sufficient but the definition of modified condition/decision coverage is beyond the scope of this paper.

In order to evaluate decision coverage it is first necessary to construct a decision graph. This shows all possible outcomes in a model. As an example, consider the very simple controller model shown in Figure 1. Essentially, this controller model provides a constant actuation “force” if the system it is controlling goes outside a specified set of bounds. It also includes the ability to “suppress” this output when needed.

```

model Controller
  import Modelica.Blocks.Interfaces.*;
  RealInput u;
  RealOutput y;
  BooleanInput suppress;
  parameter Real lowerLimit;
  parameter Real upperLimit;
  parameter Real level;
algorithm
  when u>=upperLimit then
    if not suppress then
      y := -level "Outcome 1";
    else
      y := 0 "Outcome 2";
    end if;
  end when;
  when u<=upperLimit then
    y := 0 "Outcome 3";
  end when;
  when u<=lowerLimit then
    if not suppress then
      y := level "Outcome 4";
    else
      y := 0 "Outcome 5";
    end if;
  end when;
  when u>=lowerLimit then
    y := 0 "Outcome 6";
  end when;
  when suppress then
    y := 0 "Outcome 7";
  end when;
end Controller;

```

Figure 1: A Simple Logical Controller Model

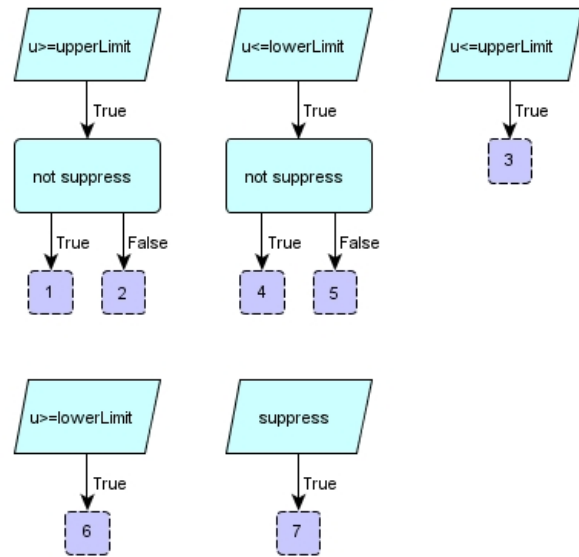


Figure 2: Controller Model Decision Tree

As previously mentioned, the UnitTesting library includes automation and reporting tools. As part of the standard reports, a decision tree is rendered for each model. Figure 2 shows the decision tree for the controller model shown in Figure 1. Each possible outcome is indicated by dark blue squares with dashed outlines. Decisions result from `if`, `when` and `while` statements as well as `if` expressions.

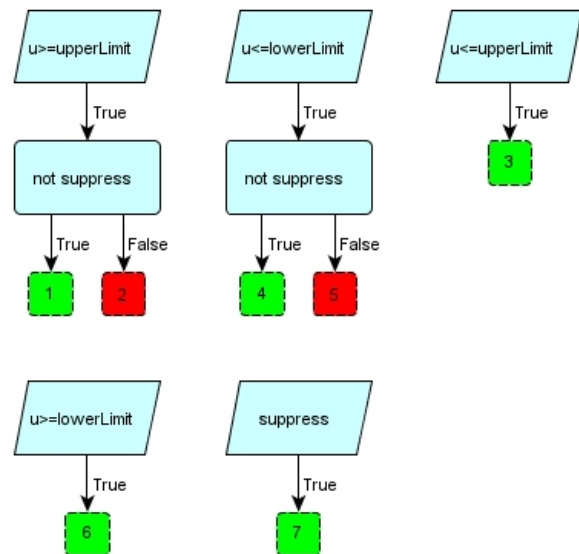


Figure 3: Decision Tree with Coverage Information

To visualize decision coverage it is necessary to use simulation results to assess whether each outcome has been reached. Given this information, we can color the figure, as shown in Figure 3, such that outcomes that have been achieved are green and outcomes that have not are red. To evaluate the deci-

sion coverage, we simply divide the number of green outcomes by the total number of possible outcomes. So in Figure 3, the decision coverage would be 71%.

Along with these decision figures, the `UnitTesting` reports include information about the individual conditional expressions associated with each decision. However, it does not currently visualize the coverage of these conditions (mostly for the sake of visual clarity).

3.4 Pinpoint Metric

The pinpoint metric is a very subtle concept. Having 100% component coverage implies that if a model suddenly stops behaving as expected, one or more test cases will fail. If each test case included only a single library component then it would be very easy to trace a particular failure to a particular model or even to a particular equation/statement.

However, it is not always practical to construct such precise test suites. For example, one library component may depend on the presence of another library component in order to function. It may not be possible to test them independently. The goal of the pinpoint metric is to determine how easy it will be to use test case failure information to identify potential sources for the failure. In other words, for a given set of test cases how easy will it be to identify the component (or set of components) that have errors based on which test cases passed and which failed.

To this end, we would like to compute the probability of an error in a specific component given the fact that exactly one test case has failed. Mathematically speaking, what we are trying to achieve can be expressed in terms of probabilities. For each library component, we would like to compute:

$$P(C_i | F_j \wedge \bar{F}_k \forall k \neq j)$$

where C_i represents an error in the i^{th} library component and F_j represents a failure in the j^{th} (and only the j^{th} test case). Given a means of computing such conditional probabilities, the pinpoint metric, π , is defined as:

$$\pi(C_i) = \max_j P(C_i | F_j \wedge \bar{F}_k \forall k \neq j)$$

The ideal value for the pinpoint metric is 1.0 because that indicates that a failure in one test case definitively points to a specific component as the source of the error. The equations used in computing the pinpoint metric are actually quite involved, especially for

complex test suites, so we will not include them in the paper.

Once we have computed the pinpoint metric for each component, we can visualize them as follows. Consider the test suite shown in Table 1. An X in this figure indicates the presence of a particular component in a particular test case. For example, a failure of all test cases points to component C_2 as the likely source of the error since it is present in all test cases. For simple test suites, this kind of deduction is not difficult but for larger test cases it is nice to automate this analysis. Furthermore, the issue is complicated by the fact that a component might fail in one test case but not in others. For example, it is possible for an error in C_2 to cause test cases 1 and 3 to fail but not test case 2. Such a result might give the misleading impression that C_1 must be the source of the error. This is the main reason that the derivation of the pinpoint metric is based on probabilities.

	C_1	C_2	C_3
T_1	X	X	
T_2		X	X
T_3	X	X	X

Table 1: Sample Test Suite

The main purpose of the pinpoint metric is to determine how well the test suite has been constructed such that component errors lead to outcomes that allow us to easily track down the source of the error. This is analogous to the concept of observability in plant models. The idea here is that given the outputs of the system (*i.e.* whether each test case passed or failed) we can gain some insight into the internal state of the system (*i.e.* whether a given component is working properly or not).

The `UnitTesting` library tools also have the ability to compute the conditional probability that a given component contains an error given not just a single test case failure (as previously discussed) but any arbitrary pattern of test case failures. With this information it is then possible to construct an accountability matrix like the one shown in Table 2.

	T_1	T_2	T_3
T_1	?	?	C_1
T_2	?	?	C_3
T_3	C_1	C_3	?

Table 2: Accountability Matrix

The accountability matrix allows us to assess, given a failure in one or two tests, which component is the most likely to have an error. Each row and column in the accountability matrix represents a test case that has failed. Each cell in the table represents the most likely component responsible (*i.e.* the one with the largest conditional probability). The accountability matrix in Table 2 was constructed based on the information in Table 1. A question mark in the table indicates cases where there is no clear choice for the responsible component.

	C ₁	C ₂	C ₃
T ₁	X		
T ₂		X	
T ₃	X	X	X

Table 3: Sample Test Suite

An ideal accountability matrix would show every component in at least one cell. As we can see from Table 2, failure in only two components can be reliably determined for the test suite in Table 1. On the other hand, if the test suite was constructed as shown in Table 3 we would get the accountability matrix shown in Table 4 where each component appears in at least one cell.

	T ₁	T ₂	T ₃
T ₁	C ₁	?	C ₁
T ₂	?	C ₂	C ₂
T ₃	C ₁	C ₂	C ₃

Table 4: Accountability Matrix

The accountability matrix takes the same information as the pinpoint metric and renders it in a very practical form that allows the developer to use knowledge about test case failures to work backward to identify the most likely source of the error. As mentioned previously, the value of automation is in being able to construct such tables for complex test suites.

3.5 Conservation Checking

In some modeling applications, it is particularly important to make sure that all interactions between components are properly accounted for. Let us start with a very simple example. Consider the EMF model from the Modelica Standard Library shown in Figure 4. This model is a basic DC motor model that relates current and voltage to torque and speed.

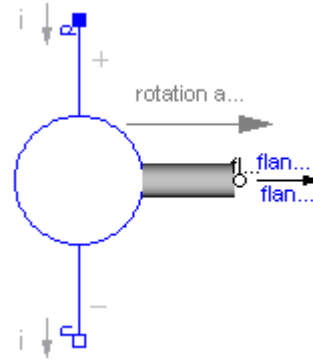


Figure 4: Standard Library EMF Model

To understand the conservation analysis, consider Figure 4 a “free body diagram” for the EMF model. Each connector represents a potential external influence on the EMF model. Generally speaking, the flow variables on the connector represent the flow of a conserved quantity into that component. In the EMF model, we see both electrical and mechanical interactions. If we run the `UnitTesting` conservation analysis on this component, the report includes a balance equation for all conserved quantities:

Quantity	Modelica.Electrical.Analog.Basic.EMF
Angular Momentum	flange_b.tau = 0
Charge	p.i+n.i = 0

Note that one particular conservation equation is rendered in a bold red font. This is to draw our attention to the fact that, based on structural analysis only, this component is unlikely to conserve angular momentum. This is because any exchange of momentum with another component would violate this equation. In some cases, such an equation is OK because the device includes some storage term. To avoid getting erroneous error messages, the `UnitTesting` tools will look for a variable inside the model that includes an annotation of the form:

```
annotation(msk(coverage(storageTerm(quantity="AngularMomentum"))));
```

Such an annotation indicates to the tools that this variable is a storage term and should be included in the balance equations. The ability to specify a storage term means that it should always be possible to avoid a “false positive” for conservation errors. Note that the EMF model does not include any storage term so the `UnitTesting` library is correct when it highlights this as an error in the model.

So far, we have only considered structural analysis but the conservation analysis also has the ability to use simulation results to determine if, in practice, the balance equation has been satisfied. In that case, it will highlight components that failed to properly balance these quantities while running the test cases.

The conservation analysis currently only addresses non-energy quantities. The reason energy is currently excluded is that computing the energy across the boundaries requires, in many cases, computing the derivatives of variables on the connectors. Unfortunately, the simulation may not need to compute (and therefore may not store) the derivatives required to perform this analysis (either because they are not necessary or because it computes the derivative of an alias variable instead). Ideally it will be possible in the future to conduct conservation analysis for energy as well. In that case, the complete conservation analysis process would be:

1. Check to make sure that the sum, over all connectors, of each conserved quantity besides energy (e.g. angular momentum, mass, current, etc) either equals zero or balances the internal storage of that quantity.
2. Check to make sure that the sum (computed either symbolically or numerically based on simulation results) of power flow over all connectors either equals zero or balances the internal storage of energy.

So far, we have focused on conservation within the components themselves. It may also seem necessary to analyze the connections between components to verify that they also conserve energy. However, for any physical connector where power flow can be computed using the variables on the connector (or their derivatives), energy conservation is automatic (e.g. all physical connectors in the Modelica Standard Library).

It may seem like such errors are easy to detect, but it is very easy to overlook important interactions. We already discussed the EMF model in the standard library. It may appear that the only possible error is one where only a single connector is included in the component. But another very common error is the one shown in Figure 5. Note that this model has two ports. So it does not suffer from the obvious error of including only a single port (and no storage term) that the UnitTesting library identified previously in the EMF model. The error in this model is more subtle than the one in the EMF model.

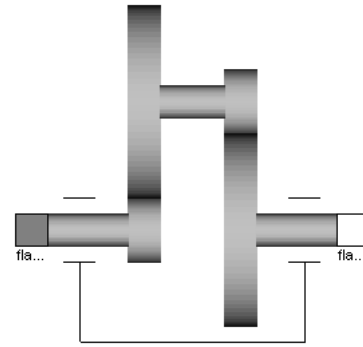


Figure 5: Non-Conserving Gear Model

To understand the nature of the error it is useful to recognize that *torque applied to a component represents a flow of angular momentum either into or out of the component*. This gear model, like the EMF model, does not include a storage term. Students are often taught to formulate the equations for a gear by first assuming a kinematic relationship between the element speeds, i.e. $\omega_a = R \cdot \omega_b$ where R is the gear ratio. We then assume that the power into the component equals the power out, i.e. $\tau_a \omega_a + \tau_b \omega_b = 0$. Using these two relationships it is trivial then to derive the relationship that $\tau_b = -R \cdot \tau_a$.

Like a good magic trick, it appears that nothing suspicious has occurred here. Everything seems logical. But for this component the UnitTesting tools would generate the following report:

Quantity	ErroneousGear
Angular Momentum	$\text{flange_a.tau} + \text{flange_b.tau} = 0$

Based on structural information alone, the UnitTesting tools would not highlight this equation because it cannot determine *a priori* that this equation cannot be satisfied by the component⁴. But when it examines the simulation results it will see that something is not correct.

The source of the problem is that the constitutive equation, $\tau_b = -R \cdot \tau_a$, cannot be correct. This is because the balance equation shown above would then be $\tau_a - R \cdot \tau_a = 0$. Assuming that the gear ratio is not 1, this equation implies that the amount of angular momentum going into the gear is not equal to the amount going out. Since this component has no storage term, angular momentum is not conserved by the component.

So where is the error? The error comes from not properly accounting for reaction torques. The model

⁴ Of course, with sophisticated symbolic processing this would be possible.

is implicitly grounded (*i.e.* it is assumed that the housing does not move). Ironically, this is exactly the same issue that the EMF model has. Torque cannot be generated out of nothing. We cannot forget Newton's Third Law, "For every action there is an **equal** and opposite reaction". For a gear to "multiply" torque something has to hold the gear housing (and absorb the reaction torque). This is exactly why the Modelica Standard Library gear looks like the one shown in Figure 6 (note the presence of the third port).

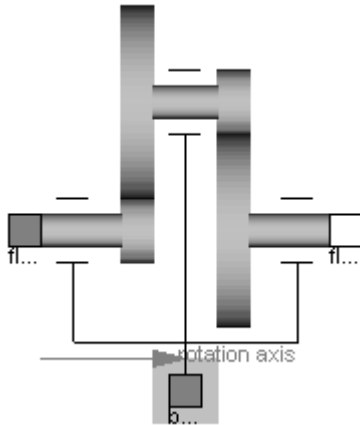


Figure 6: Conservative Gear

3.6 Style Guideline Compliance

A whole family of metrics can be constructed based on style guidelines. It is possible to formulate metrics around guideline adherence to help drive improvements in libraries. While this is beyond the scope of this paper (see [5] for a discussion of this topic), it should be noted that a comprehensive approach to library quality could benefit from attention to these kinds of details since they affect, in some cases significantly, user perception of library quality in subjective ways.

4 Using the Library

This section provides a basic overview of how the library is used in practice.

4.1 Making a Test

The first step in testing is to create a set of test cases. It helps to think of each test case as an experiment to test individual components or small groups of components. The fewer components in a test case, the better the pinpoint metric will be for the library components.

After an appropriate system model has been created, it can be turned into a test case by simply inheriting from the `UnitTesting.TestCase` model. This identifies the model as a test case. The TestCase model requires two parameters to be specified. The first is the name parameter which defines a unique name for identifying the test case⁵. The second parameter, `resultsDir`, indicates what directory any test result information should be stored in⁶. Typically, this results in a model that looks something like:

```
model TestComponentA
  import UnitTesting.TestCase;
  extends TestCase(name="TestComponentA", resultsDir=testResultsDir);
  ...
end TestComponentA;
```

In this case, `testResultsDir` is presumably some package level constant to define where all test results should be placed.

4.2 Adding Results

For the test case to be useful, it should include expected results to validate the model against. Although all kinds of expected results are possible, the most common type of result is one that compares the current simulation results to a previous baseline result. This kind of result is represented by the `ScalarResult` model. A typical declaration of the `ScalarResult` model might look like:

```
ScalarResult result1(name="inertia.phi", y=inertia.phi);
```

where the name parameter should be given a unique value (at least within this test) to distinguish it from other results and the equation for `y` should set it equal to the variable being validated.

As mentioned previously, better state coverage will result in better error detection rates. However, instrumenting the model to check all possible states is tedious because it involves typing in many `ScalarResult` declarations. For this reason, the `UnitTesting` reports include a sample `ScalarResult` declaration (like the one above) for each state in the test case. Copying these declarations

⁵ If, at some point, the Modelica language is expanded to provide a function that returns the fully qualified name for any class then this parameter would no longer be necessary.

⁶ Again, better introspection support in the language could eliminate this parameter as well.

from the report into the test case ensures 100% state coverage.

4.3 Generating Expected Results

As mentioned previously, it is quite common to base the expected results on previously determined simulation results. Such results provide a baseline on which to assess the model over time. It would be quite tedious to manually specify all expected results (e.g. at time=4.32 the value of the signal should be 0.8203, and so on). For this reason, the `UnitTesting` library provides a means to extract expected values from existing simulation results.

For example, the previously mentioned `TestCase` model includes a special `generate` flag. When this flag is set, each `ScalarResult` component extracts the value of the variable it is monitoring and places it in a special file. When the `generate` flag is not set, these same components assume that results have already been generated and attempts to load them from the results directory (specified in the `TestCase` `extends` clause). The default value of the `generate` flag is `false`. So the first time the test case is run it is necessary to set the `generate` flag to generate the baseline results. After that, the `generate` flag should not be set and it will run properly as a check against the baseline results.

It is always a good idea to keep models under version control. In that case, the baseline results should also be version controlled. This is because it will be difficult (if not impossible) to regenerate these baseline results in the future. Note that it is not always necessary to generate baselines results. Some expected result models allow comparisons against analytical solutions. Furthermore, the `UnitTesting` library allows new kinds of expected result models to be created by simply extending from the `UnitTesting.TestResult` base class.

4.4 Running the Test

Once a test case has been created by extending from the `TestCase` base class and adding any expected result components, the test case is ready to be run. Running a test just means running a simulation of the test case model. The `TestCase` base class and expected result components contain all the “code” necessary to automatically determine whether the test case has passed or failed.

There are many things that can cause the failure of a test case. The obvious failure mode is that the computed value for a variable does not match the ex-

pected value (within some pre-specified tolerance). But there are other subtle ways a test can fail. For example, if the expected results occur over the time interval from 2 seconds to 7 seconds but the simulation runs from 0 seconds to 6 seconds, this is a failure. Recall our previous definition for a failure was that a test case did not match all expected results. In this case, the simulation failed to match the expected results between 6 seconds and 7 seconds because it did not simulate that period of time. This is a failure because the simulation ended too soon and it did not check all expected results. A similar failure occurs if the simulation starts too late.

If an expected result does not match, the simulation will terminate immediately and generate a message describing the reason for the failure. In some cases, in order to understand why a test is failing it is useful to run the test case to completion in spite of the failure. In this case, setting the `fatal` flag to `false` prevents the immediate termination of the simulation due to failure. This allows the full solution trajectory to be reviewed which may provide some insights into why the test failed.

4.5 Organizing Test Cases

The `UnitTesting` library identifies test cases based on whether they inherit from the `UnitTesting.TestCase` class. For this reason, it does not care how test cases are organized in the package hierarchy. Some developers may wish to keep the test cases in the same library as the library components while others may chose to maintain a separate library just for test cases. The `UnitTesting` library functions equally well in either scenario.

4.6 Automated Testing

Although it may occasionally be useful to run a single test, complete validation requires that all tests be run. For this reason, the `UnitTesting` library includes several built-in functions that, in conjunction with the “Commands” menu in Dymola, allow all tests to be run with the click of the mouse.

4.7 Reports

In the ideal situation, the test suite provides 100% component coverage, 100% condition/decision coverage and 100% state coverage and running all the tests yields no errors. But it takes some time to get to this point. In the meantime, library developers need assistance to understand what kinds of tests are required or what components are the most likely

source of the test failures that occur. This is where the `UnitTesting` reporting capabilities come in.

In addition to the metrics described previously, the `UnitTesting` reports generate additional information like dependency graphs, accountability matrices, decision trees, *etc.* They also include further analysis (beyond whether a test passed or failed) to highlight conservation issues and other information based on simulation results.

The reports are generated as HTML. This HTML is then embedded, using the `Documentation` annotation in Modelica, in special packages that are automatically inserted into the test case library. These packages include the `DocumentationClass` annotation so, like the Modelica Standard Library User's Guide, they can be immediately identified in the package hierarchy as reference documentation.

5 Conclusions

For library developers, testing can help ensure a quality product. Rather than counting on visual inspection of results from a few complex system models, the `UnitTesting` library and associated automation and reporting tools can be used to create a fine mesh of tests to detect errors or discrepancies at the lowest level and quickly identify the source of the problem.

This formal approach provides a rigorous validation and verification process for the underlying models. The presence of such a process may eliminate barriers preventing library components from playing a more significant role in the product development process.

References

- [1] Husted, T., Massol, V., "JUnit in Action", Manning Publications, ISBN 1930110995.
- [2] Busser, R. D., Blackburn, M. R., Nauman, A. M., "Automated Model Analysis and Test Generation for Flight Guidance Mode Logic", *Proceedings of AIAA/IEEE Digital Avionics Systems Conference*, v 2, 2001.
- [3] Busser, R. D., Boden, L. M., "Extending Simulink Models with Natural Relations to Improve Automated Model-Based Testing", *Proceedings of 29th Annual IEEE/NASA Software Engineering Workshop*, 2005.
- [4] Hayhurst, K. J., Veerhusen, D. S., Chilenski, J. J., Rierson, L. K., "A Pratical Tutorial on Modified Condition/Decision Coverage", NASA Center for AeroSpace Information, NASA/TM-2001-210876.
- [5] Tiller, M., "Parsing and Semantic Analysis of Modelica Code for Non-Simulation Applications", *Proceedings of the 2003 Modelica Conference*.

Session 6d

Multidomain Systems

If we only had used XML...

Ulf Reisenbichler Hansjörg Kapeller Anton Haumer
 Christian Kral Franz Pirker Gert Pascoli
 Arsenal Research
 Giefinggasse 2, 1210 Vienna, Austria

phone +43-50550-6606, fax +43-50550-6595, e-mail: hansjoerg.kapeller@arsenal.ac.at

Abstract

The first part of this paper will provide a general overview over XML technology and will discuss some aspects of designing applications of XML in respect of storing and retrieving data in a standardized and efficient way. This overview will focus on the essential elements of XML needed for data modeling. The second part will introduce a dynamic link library based implementation for XML data handling out of Dymola.

Consequently these considerations should lead into a discussion of how to set up standardized applications of XML for modeling and handling data in various domains of engineering to benefit from the capabilities of XML technology for data exchange in general and specifically for data handling in simulation environments like Dymola.

Keywords: XML; data modeling; data handling; data interchange; XML schema; XSD

1 Introduction

Is there an answer to the recurring question of how to store data for easy and efficient data handling and data interchange? Do we have to continue to pick and poke bits and bytes out of unreadable data chunks, or is it possible to retrieve data much more convenient? Is there a way to design a language to describe data and data structures for our own specialized purposes? Will anybody intuitively understand what we tried to formalize? The response to all of these questions would be: Yes ... if we only had used XML.

In addition to structure and interpretation of data, storing and retrieving data either in plain text or binary formats will always cause explanatory effort of *how* these data are stored – e.g. position of characters or bytes. This fact necessitates the development of individual parsing algorithms for each data format. Us-

ing the standardized structure of XML will place emphasis on the data itself and their coherency. XML documents can be read and intuitively understood by humans without any additional information as well as processed by computers using already existing analysis tools.

The objective of this paper is to motivate – once more – the general use of XML for data handling [5]. The development of ModelicaXML [3] – a representation of Modelica [1] source code using XML – points out that XML is the right concept. A brief theoretic outline will clarify terms and concepts without getting too deep into theoretical computer science or formal syntax and semantics. An interdisciplinary example will demonstrate how to interface XML data sources out of Dymola, and will show: Using XML is easy, intuitive and the best investment for the future.

2 XML

2.1 History

XML (Extensible Markup Language) was derived from the 1986 standardized SGML (Standard Generalized Markup Language) going back to the late 60ies GML (Generalized Markup Language). The XML 1.0 Recommendation was published first on 10th February 1998 by the World Wide Web Consortium (W3C). The most current version XML 1.1 was published on 4th February 2004 [7], [8].

These recommendations were – more or less accurate – realized within different applications of XML. Thereby XML evolved over the past few years into a widespread standard for handling data and documents. Within this evolution various techniques and standards for processing XML documents were developed and implemented which can be easily adapted for specific needs.

2.2 General

The main idea of all markup languages is to separate the content of a document from its structure. Whereas the “structure” can represent the formatting of a document – as in (X)HTML ((Extensible) Hyper-Text Markup Language) – or the informational content of data within a document – as in e.g. MathML (Mathematical Markup Language), an XML based encoding standard describing mathematical notation. The inherent difference between these applications of the concepts and the generalized standards of markup languages is that these standards are metalanguages. A metalanguage like XML provides the opportunity to formulate “new languages” following specific newly defined rules in accordance to the restrictions given by the standardized stipulations.

Defining a “new language” is strictly speaking the only task when creating an application of XML. This language can be very simple or highly complex. All languages must have certain “meaningful” elements – the tokens – and rules which describe the valid sequences and hence restrict the possible combinations of these elements – the syntactic rules or syntax. Mapping the patterns generated by these rules to a structural model and projecting the elements into this derived representation of the model will give a “sentence” of this language.

Within XML the generated “sentence” would be the resulting “file” derived from the freely defined syntax and tokens of the defined markup language, i.e. the application of XML, representing the ruleset for generating and understanding documents following these rules. But XML is not simply the resulting “file”, it is the sum of the concepts and utilization of the surrounding technologies.

2.3 Components

This section will provide a synopsis of the basic concepts and some components within XML technology to demonstrate how they interact one with each other unfolding the power of XML.

2.3.1 Elements

The token elements of XML are represented by the characters enclosed in angle brackets, which mark the characters surrounded by the combination of a start tag `<x>` and the corresponding end tag `</x>` – with `'/'` as symbol for ending the tag – as their content, whereas `x` is the name of the token. The content of an

element may be solely character data or other elements or both of them.

A token without content can be written as `<x/>`, the empty element tag. Tokens can be specified further with attributes of the form `attribute="value"`. Attributes may only appear in start tags and empty element tags, e.g. `<x attribute="value">`, and must be unique.

2.3.2 Documents

An XML document is the data structure which is normally denoted as XML file when saved to a persistent storage device. Combining the tokens to a document only two structural conditions must be kept:

1. Every document must have a root element
2. All tags must be properly nested

These conditions will build the structural model of a tree. This is often stated as being the syntax of XML. But using a tree as structural model is not applying syntactic rules to that model. The model only gives the structure on which the syntax will operate. If a document meets this condition, it is well-formed and therefore an XML document. By definition a *not* well-formed document is *no* XML document.

Beside this tree of tokens an XML document can include other non token elements. Usually an XML document starts with an XML declaration of the form `<?xml version="1.0" encoding="UTF-8"?>`. The attribute `version="1.0"` is mandatory. The optional attribute `encoding="UTF-8"` declares the used character set and should always be used to ensure correct cross locale processing. A comment can be added between `<!--` and `-->` outside the tags anywhere in the document. There are some more elements defined by the W3C recommendation but will not be discussed here.

2.3.3 Parsers

The core piece of machine processing XML documents is an XML parser implementing all the features needed for a specific task. None of the currently available parsers – such as Xerces, libxml, Saxon or Microsoft MSXML – is implementing all of the features recommended by the W3C up to 100% [2]. While parsing mechanisms can be very different, the interface presenting the content of an XML document to an application will follow one of the standard parsing

models, i.e. the Simple API for XML (SAX) [4] or Document Object Model (DOM) [6].

The Simple API for XML is an event driven parsing model. The underlying parsing algorithm informs an application of certain events representing the structure of an XML document through callbacks. These callbacks will indicate the start and end of the document, the start and end of an element, the content of the elements and others. As SAX simply takes the document as a stream to read through without saving any content in memory it is most commonly used in performance critical applications or when parsing large documents. The Document Object Model is the model of choice for analysis and alteration of XML documents. The parser will create a complete representation of the document as a DOM tree in memory. This tree can be traversed to retrieve information or may be transformed in structure or content. DOM is, in contrast to SAX, processing time intensive and memory consuming when parsing the document, but once loaded into memory – which is not possible with SAX as such – all operations will not be very time consuming and the results of these operations can be saved to a persistent storage device.

2.3.4 Evaluation

XML provides two levels to evaluate the structural correctness of documents. A document is well-formed if the structural standard requirements of XML are met, i.e. the document simply must be parseable. To meet the stronger constraint of being “valid” the document must pass a check against additional restriction rules. These additional rules can be defined either as Document Type Definitions (DTD) or as XML Schema Definitions (XSD) each representing constraints upon what elements may appear in a document, their relationships to each other or what types of data are represented by them [12]–[16]. DTD will not be discussed here as XSD is the much more powerful and most recent standardized schema definition.

Figure 1 shows a simple well-formed XML document, with token elements projected onto an XML tree structure. Some of these tokens contain other tokens and some contain values. Intuitively humans will construct a hypothesis of the structural rules, i.e. the syntax, and the “meaning”, i.e. the semantics, of these tokens.

The semantic content of these elements is the assigned value determined by the token. In this example values are only assigned to the tokens which are leaves in relation to the tree structure, i.e. which terminate their branch of the tree. On the other hand the interaction of

```
<?xml version="1.0" encoding="UTF-8"?>
<motor serial="M-453123">
  <type>asynchronous induction machine</type>
  <stator part="s-4583/733">
    <material>IronSheets</material>
    <slots>
      <quantity>48</quantity>
      <slot>
        <dimension>
          <height>23e-3</height>
          <width>7e-3</width>
        </dimension>
        <winding>
          <material>Copper</material>
        </winding>
      </slot>
    </slots>
  </stator>
  <rotor part="r-4877/A72">
    <material>IronSheets</material>
    <slots>
      <quantity>40</quantity>
      <slot>
        <dimension>
          <height>20.6e-3</height>
          <width>2.8e-3</width>
        </dimension>
        <winding>
          <material>Aluminium</material>
        </winding>
      </slot>
    </slots>
  </rotor>
</motor>
```

Figure 1: motor.xml

terminal and non terminal elements is bearing intrinsic semantic information, i.e. this “tokenized” structure reflects the meaning of the syntactically defined branches. To give a precise definition of this restrictions on values and possible syntactic sequences of the tokens – either for humans or machine processing – and to further check this definitions against an XML document an XSD document can be set up (fig. 2), which then will verify the content of this XML document by the validating mechanism of the parser.

The XML Schema definition language opens the possibility to declare XML elements in terms of type definitions. These element declarations, when applied to an actual XML document, represent the abstract concept of type and token. The type of a token will restrict sequences of tokens and define their content. The limited number of components in this example will only sketch the potential of the XML Schema definition language, but illustrates the principles of its conception. The XML element with the name `motor` is a `complexType` which may only contain a sequence of the elements `type`, `stator` and

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:material="http://www.arsenal.ac.at/material"
  xmlns:format="http://www.arsenal.ac.at/formats">
  <xsd:import namespace="http://www.arsenal.ac.at/material"
    schemaLocation="material.xsd"/>
  <xsd:import namespace="http://www.arsenal.ac.at/formats"
    schemaLocation="formats.xsd"/>
  <xsd:element name="motor">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="type"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="stator" type="activepart"
          minOccurs="1" maxOccurs="1"/>
        <xsd:element name="rotor" type="activepart"
          minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="serial" type="format:serial"
        use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="activepart">
    <xsd:sequence>
      <xsd:element name="material" type="material:magnetic"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="slots" type="slots"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="part" type="format:part"
      use="required"/>
  </xsd:complexType>
  <xsd:complexType name="slots">
    <xsd:sequence>
      <xsd:element name="quantity" type="xsd:integer"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="slot" type="slot"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="slot">
    <xsd:sequence>
      <xsd:element name="dimension" type="format:size"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="winding" type="winding"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="winding">
    <xsd:sequence>
      <xsd:element name="material" type="material:conductor"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Figure 2: motor.xsd

rotor, and has an attribute named *serial*. Within the sequence the occurrence of the elements is further restricted by the numbers given in the attributes *minOccurs* and *maxOccurs*, i.e. *type* may, while *stator* and *rotor* must occur exactly one time. These are purely syntactic rules. The attribute *type* now gives the additional information about the content of the elements. The elements *stator* and *rotor* are of the same type *activepart*, which represents on his part a sequence of *material* and *slots*, and the required attribute *part*. This new subordinated structural sequence is also defined purely syntactically, but the succession of elements assigns a meaning to the whole branch, which could be denoted as syntactico-semantic rule. The element named *type* has no at-

tribute *type* and therefore can have any content being a *value* as it is a terminal element lacking any further branching.

Additionally, fig. 2 illustrates the concepts of XML namespaces and qualified names. An XML namespace is defined with the attribute *xmlns* relating an URI reference, e.g. <http://www.w3.org/2001/XMLSchema>, to XML elements, which then build up a collection of names belonging to this namespace. A namespace may be defined in any XML element – not only in XSD schemas – incorporating all subordinate elements into this namespace. To delimit the scope of the specified namespace a prefix, *xmlns:prefix*, may be assigned to qualify only certain elements as belonging to this namespace. The resulting qualified name of the element, *prefix:name*, binds the local name of the element to the specific namespace thus the same name can be used within different namespaces.

An XML Schema may also be compound of different schemas which can be defined as *import* into a schema. The imported schemas must each define their own namespace and may have a given *schemaLocation*. These namespaces and their prefixes are defined in the root element *schema*, here: *xsd:* the namespace of the XML Schema definition itself, *material:* the namespace of a schema defining different “materials” (fig. 3) and *format:* the namespace of a schema defining different number formats (fig. 4).

In fig. 2 the two syntactically defined elements named *material* – one child to *activepart* and the other child to *winding* – are represented by different types, i.e. *material:magnetic* and *material:conductor*, specified in fig. 3. The *xsd:simpleType* named *magnetic* of the base type *xsd:string* underlies the *xsd:restriction* that its content can only be one value out of the *xsd:enumeration* containing *IronSheets* and *CastIron* – correspondingly *material:conductor*. A simple type is always a terminal element, but its value and therefore its semantic content may be restricted.

The values of an element or an attribute may also be restricted using a given pattern like a regular expression or assigning a simple type like one of the built in data types of the XML Schema definition language, e.g. *xsd:integer*, *xsd:string* or *xsd:float*.


```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.arsenal.ac.at/material">
  <xsd:simpleType name="magnetic">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="IronSheets"/>
      <xsd:enumeration value="CastIron"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="conductor">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Copper"/>
      <xsd:enumeration value="Aluminium"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Figure 3: material.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.arsenal.ac.at/formats">
  <xsd:complexType name="size">
    <xsd:sequence>
      <xsd:element name="height" type="xsd:float"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="width" type="xsd:float"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="serial">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[M] [\-] [0-9]{6}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="part">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[r|s] [\-] [0-9]{4} [/] [A-Z,0-9]{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Figure 4: formats.xsd

2.3.5 Retrieving data

Besides others, XPath (XML Path Language) is a standard query language for data retrieval within an XML document [11]. XPath expressions represent a specific node or node set within an XML document. Evaluating an XPath expression against a document, a parser will return all matching nodes for the structural description given. The expression `/motor/rotor/slots/quantity/text()` which defines the complete path to the second `<quantity>` node in fig. 1 will return the value, i.e. 40, of the node using the XPath function `text()`. The expression `//slots/quantity/text()` – with “//” meaning get all nodes matching within the document no matter where they are – will return 48 and 40. The complete XPath language gives much more possibilities for creating query expressions, but these very basic expressions illustrate the intuitive understanding and the simplicity of XPath.

2.3.6 Displaying data

To present data in an actually human readable format, a parser may transform an XML document into a HTML document using the Extensible Stylesheet Language (XSL) [9], [10], [17]. XSL uses XPath to retrieve data out of the XML document, which will be displayed after transforming a template into the resulting HTML representation. Figure 5 shows a minimal stylesheet displaying the `xsl:value-of` of the Xpath expression `/motor/stator/material` in a minimal HTML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <xsl:value-of select="/motor/stator/material"/>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

Figure 5: motor.xsl

All kinds of documents for validating (XSD) and formatting (XSL) can be bound to the XML document within itself (fig. 6). These files may be distributed over the www. If the parser (within a browser) is interpreting all documents and transformations correctly the XML document will be self-testing, self-validating and self-formatting.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="motor.xsl"?>
<motor serial="M-453123" xmlns:xsi="http://
  www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="motor.xsd">
```

Figure 6: Connected documents

2.4 XML-Document Design Guidelines

Designing a complete markup language for a specific application area will be, in any case, a laborious task. The components introduced in the previous sections imply the elementary principles for designing the base units of such an application, i.e. the typed elements represented by tokens when building up a document. Types can either be represented by a terminal element or another structural element. Grouping the structural elements will end up in the complete document.

```

<p>
  text in a
  <table border="1">
    <tr>
      <td bgcolor="red">terminal</td>
    </tr>
  </table>
  non terminal element
</p>

```

Figure 7: Content centric XML

The design of the elements is intimately connected with the specific application of XML. If the main focus of a metalanguage is put on adding information to the “pure” content of a document, as in (X)HTML, this will be a content centric approach.

If the structure and the values of the data themselves will be focused, as if XML is used for data handling, this will be a data centric approach. The XHTML code snippet in fig. 7 illustrates what should be avoided when using XML for data handling. The paragraph element `<p>` contains besides its structural sub element `<table>` other character data which are “values” in a stricter sense. Comparably the attributes `border` and `bgcolor` contain values not specifying their structural information but assigning additional content to their “real value”.

When following the data centric approach a clear distinction of what is or can have a value and how the data are structured must be made when formalizing and hence abstracting the real world. Thus the structural elements, i.e. non terminal tokens, should strictly reflect the conceptual context of the data. Values should exclusively be represented by the content of terminal elements and never within attributes. Attributes may be used to identify elements within the structural relation or specify external references not directly related to concepts within a specific document. This will transform fig. 7 to – for example – fig. 8.

This will lead to a strict structural definition of the values within a document, and will give clearly defined and simple XPath expressions when retrieving data.

3 Dymola XML

3.1 XML Engine

The architecture of the XMLEngine environment (fig. 9) is designed as a scalable “open” system of

```

<p>
  <text>terminal</text>
  <table>
    <format>
      <border>1</border>
    </format>
    <content>
      <tr>
        <td>
          <color>
            <background>red</background>
          <color>
            <text>terminal</text>
          </td>
        </tr>
      </content>
    </table>
  <text>terminal</text>
</p>

```

Figure 8: Data centric XML

components which can be used out of multiple applications. The central unit within this environment is the dynamic link library XMLEngine.dll. It is implemented in C++ and can be used directly by applications dynamically loading the library and calling the interface methods.

The XMLEngine library has no internal parser and therefore must be bound to an external parser. The current implementation uses Xerces 2.6 and its XPath module Xalan 1.9 interfaced by the dynamic link library Xerces.dll binding the XML functionality to the application.

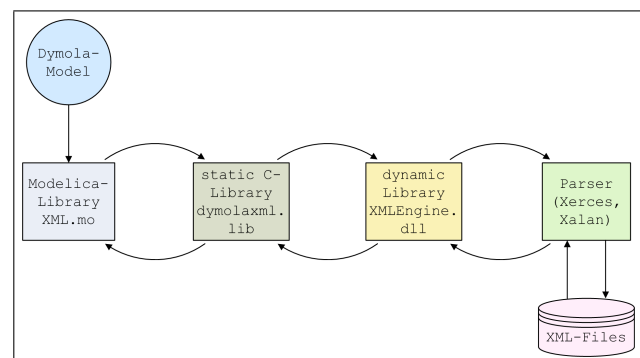


Figure 9: Architecture of the XMLEngine environment

3.2 Dymola

Following the object oriented approach of Modelica, the programming language incorporated into the simulation environment Dymola, the idea of strict separation of code (models) and data (parameters) reflects this attempt for data handling. Using XML documents opens the possibility to store and retrieve primitive data types as well as highly complex data objects in a flexible way.

If a defined application of XML is used as base for data exchange, the import of precalculated data, for e.g. a thermodynamic model, and the export of results, e.g. state variables, for any calculation and simulation tool, following these standardized rules, is guaranteed to work correctly with minimum effort.

Dymola is bound via the static link library `dymolaxml.lib` for MS-Visual C or `libdymolaxml.a` for GNU C to the XMLEngine environment. The XML query language XPath is used to import data from XML documents and to export data into predefined template documents. The integration of the XMLEngine environment into Dymola in cooperation with the scalability and flexibility of XML opens a wide range of possibilities for data handling and data exchange.

3.3 Interdisciplinary Application Example

An interdisciplinary example (containing mechanics, electrical engineering and thermodynamics, fig. 10) taken from drive engineering will demonstrate how to retrieve parameter values from an XML document and how to save results to a file using a predefined XML document template. The single aspects of this example and the corresponding parameter values are reflected by the structure of the elements in the underlying XML document (fig. 11). The different elements are combined via connectors forming the entire model (fig. 10).

- “thermalModel” represents the thermodynamic activity in a permanent magnet DC motor
- the permanent magnet DC motor is as such an electromechanical model
- the electric source represents an electrotechnical application
- the mechanical load is characterized by pure mechanical equations

Having defined the data in an XML document – and not within each single sub model – different aspects

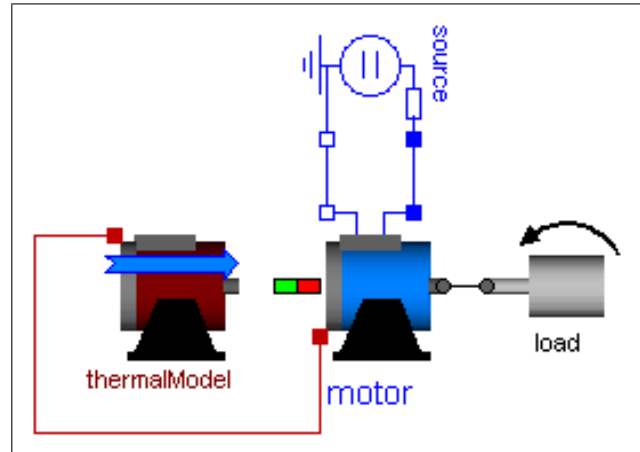


Figure 10: Interdisciplinary application example

of the entire model may be simulated loading the data from – and manipulating the data in – an external data source. By changing the parameter sets – in the now “centralized” data source – it will be possible to analyze for example the thermal stress within the field windings of different motor types (fig. 15). More elegantly different parameter sets may be defined directly in the XML document and selected within Dymola. This approach is illustrated here by the “load_data” element (fig. 11) to operate several workloads of the motor.

The Dymola XML Library is using the Modelica standard external function interface to load and access data from an XML document. For loading data into a model firstly the parser must be initialized to obtain a handle for further processing (fig. 12):

```
domParserHandle := createDOMParser(
  ParserName )
```

The next step is loading the XML document as DOM tree representation into memory, using the handle of the previously created parser:

```
DocumentHandle := loadDocument( domParserHandle,
  FileName )
```

When initializing the parser and loading documents within the same model this must be done in an initial algorithm to ensure strict algorithmic processing order.

Now having a handle to a document retrieving and changing of the data is possible using XPath expressions. In this model the parameters and the initial conditions for the complete characterization of the system of differential equations will be retrieved from the DOM trees residing in memory. The XML document template for storing the results of the simulation will be loaded for later use. To obtain a single result value when evaluating an XPath expression against the

```

<?xml version="1.0" encoding="US-ASCII"?>
<data>
  <source_data>
    .
    .
  </source_data>
  <motor_data>
    <motor name="Standard">
      <electric>
        <VaNominal> 100</VaNominal>
        <IaNominal> 100</IaNominal>
        <rpmNominal>1425</rpmNominal>
        <Ra> 0.05</Ra>
        <Ta> 70</Ta>
        <La>0.0015</La>
        <J> 0.15</J>
      </electric>
      <thermal>
        .
        .
      </thermal>
    </motor>
  </motor_data>
  <load_data>
    <load name="NoLoad">
      .
    </load>
    <load name="Constant">
      .
    </load>
    <load name="Intermittent">
      .
    </load>
  </load_data>
</data>

```

Figure 11: IDE.xml

loaded document the handle of the document and the XPath expression has to be given over to the corresponding function:

```
getReal( DocumentHandle, XPath )
```

This function returns the first result matching the XPathExpression (fig. 13).

All functions are also available for Integer, Boolean and String values. All “get” functions are also available in versions evaluating the second XPath expression starting at the result from a context node’s XPath result: `getRealFromContext(DocumentHandle, ContextNodeXPath, XPath)`

Changing data values in previously loaded documents one of the “set” function will be called specifying the document, the location as XPath expression and the new value:

Modelica definition

```

model Example
  parameter String sourceName = "Standard";
  parameter String motorName = "Standard";
  parameter String loadName = "Intermittent";
  .
  .
  .
  parameter TAmb = 20 "Ambient temperature";
  output TWinding=thermalModel.Winding.T;
  output THousing=thermalModel.Housing.T;
  protected
  parameter Integer domParserHandle(fixed=false);
  parameter Integer parDocumentHandle(fixed=false);
  parameter Integer iniDocumentHandle(fixed=false);
  parameter Integer trmDocumentHandle(fixed=false);
  .
  .
  .
  initial algorithm
    domParserHandle :=createDOMParser("Xerces");
    parDocumentHandle :=loadDocument(domParserHandle,
      "IDE.xml");
    iniDocumentHandle :=loadDocument(domParserHandle,
      "Initial.xml");
    trmDocumentHandle :=loadDocument(domParserHandle,
      "TerminalTemplate.xml");
    if iniDocumentHandle == 0 then
      TWinding:=TAmb;
      THousing:=TAmb;
    else
      TWinding:=getReal(iniDocumentHandle, "//data/
        TWinding");
      THousing:=getReal(iniDocumentHandle, "//data/
        THousing");
    end if;
  .
  .
  .
end Example;

```

Figure 12: Initial algorithm

```
setReal( DocumentHandle, XPath,
  newValue );
```

Finally the changed template document will be saved to disk calling (fig. 14):

```
saveDocument( DocumentHandle,
  newDocumentName );
```

The following simulation results refer to the above mentioned DC machine operated at continuous duty with intermittent periodic loading. The load torque is alternating between no load (0Nm, 24s) and load (80Nm, 36s) periodically. In fig. 15 the temperature rise of the DC-machine field winding and in the housing is depicted. In a new simulation procedure these final values can be used as new initial values.

4 Conclusions

Separating simulation models and data can be very convenient when using XML documents and XPath expression for storing, retrieving and manipulating

Modelica definition

```

model DC_PM "Permanent magnet DC machine"
  parameter Integer parDocumentHandle = 0;
  parameter String motorName = "";
  extends
  .
  .
  .
protected
  parameter Modelica.SIunits.Voltage VaNominal =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      VaNominal/text()" );
  parameter Modelica.SIunits.Current IaNominal =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      IaNominal/text()" );
  parameter Modelica.SIunits.Conversions.
    NonSIunits.AngularVelocity_rpm rpmNominal =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      rpmNominal/text()" );
  parameter Modelica.SIunits.Resistance Ra =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      Ra/text()" );
  parameter Modelica.SIunits.Conversions.
    NonSIunits.Temperature_degC Ta =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      Ta/text()" );
  parameter Modelica.SIunits.Inductance La =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      La/text()" );
  parameter Modelica.SIunits.Inertia J =
    getReal(
      parDocumentHandle, "//data/motor_data/
      motor[@name=\""+motorName+"\"]/electric/
      J/text()" );
public
  .
  .
  .
equation
  .
  .
  .
end DC_PM;

```

Figure 13: Motor specification

Modelica definition

```

model Example
  .
  .
  .
  initial algorithm
  .
  .
  .
equation
  when terminal() then
    setReal(trmDocumentHandle, "//data/TWinding",
      TWinding);
    setReal(trmDocumentHandle, "//data/THousing",
      THousing);
    saveDocument(trmDocumentHandle, "Terminal.xml");
  end when;
  .
  .
  .
end Example;

```

Figure 14: Storing simulation data

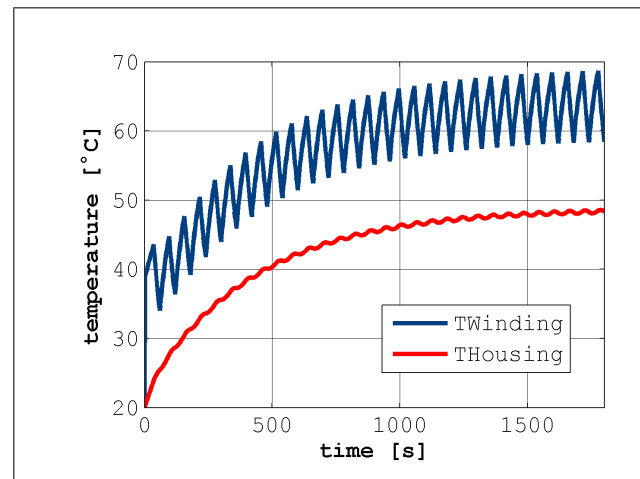


Figure 15: Simulation results

data. Additionally, XML technology provides the opportunity of modeling data in a standardized way. The resulting application of XML, i.e. the specific meta-language, can be designed independently from the use in certain models and therefore provide a consistent basis of terminology and structured data representations for various domains of engineering.

References

- [1] Fritzson P., *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Piscataway, NJ: IEEE Press, 2004,
- [2] Harold, E. R., *SAX Conformance Testing*, XML Europe 2004 Conference,

- http://www.idealliance.org/papers/dx_xml04/papers/03-06-02/03-06-02.pdf
- [3] Pop A., Fritzson P., *ModelicaXML: A Modelica XML Representation with Applications*, Proceedings of the 3rd International Modelica Conference, pp. 419-430, 2003
- [4] The SAX project, home page, <http://www.saxproject.org>
- [5] Tiller M., *Implementation of a Generic Data Retrieval API for Modelica*, Proceedings of the 4th International Modelica Conference, pp. 593–602, 2005
- [6] World Wide Web Consortium (W3C), *Document Object Model (DOM)*, <http://www.w3.org/DOM>
- [7] World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.0* (Third Edition), W3C Recommendation 04 February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204>
- [8] World Wide Web Consortium (W3C), *Extensible Markup Language (XML) 1.1*, W3C Recommendation 04 February 2004, <http://www.w3.org/TR/2004/REC-xml11-20040204>
- [9] World Wide Web Consortium (W3C), *Extensible Stylesheet Language (XSL)*, Version 1.0, W3C Recommendation 15 October 2001, <http://www.w3.org/TR/xsl>
- [10] World Wide Web Consortium (W3C), *The Extensible Stylesheet Language Family (XSL)*, <http://www.w3.org/Style/XSL>
- [11] World Wide Web Consortium (W3C), *XML Path Language (XPath)*, Version 1.0, W3C Recommendation 16 November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>
- [12] World Wide Web Consortium (W3C), *XML Schema Part 0: Primer* Second Edition, W3C Recommendation 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>
- [13] World Wide Web Consortium (W3C), *XML Schema Part 1: Structures* Second Edition, W3C Recommendation 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>
- [14] World Wide Web Consortium (W3C), *XML Schema Part 2: Datatypes* Second Edition, W3C Recommendation 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>
- [15] World Wide Web Consortium (W3C), *XML Schema 1.1 Part 1: Structures*, W3C Working Draft 30 March 2006, <http://www.w3.org/TR/2006/WD-xmlschema11-1-20060330>
- [16] World Wide Web Consortium (W3C), *XML Schema 1.1 Part 2: Datatypes*, W3C Working Draft 17 February 2006, <http://www.w3.org/TR/2006/WD-xmlschema11-2-20060217>
- [17] World Wide Web Consortium (W3C), *XSL Transformations (XSLT)*, Version 1.0, W3C Recommendation 16 November 1999, <http://www.w3.org/TR/1999/REC-xslt-19991116>

Coupled Simulation of Building Structure and Building Services Installations with Modelica

Peter Matthes
Thomas Tschirner

Timo Haase
Dirk Müller

Alexander Hoh

Hermann–Rietschel–Institute, Technical University of Berlin
Marchstr. 4, D–10587 Berlin, www.tu-berlin.de/fak3/hri

peter.matthes@tu-berlin.de timo.haase@tu-berlin.de alexander.hoh@tu-berlin.de
thomas.tschirner@tu-berlin.de dirk.mueller@tu-berlin.de

Abstract

The Hermann–Rietschel–Institute (HRI) uses the ability of Modelica to aggregate different physical domains inside one simulation model. This approach allows for the coupled simulation of thermal effects in building structures and building services installations applying only one software tool.

We show the capabilities of such coupled simulations, taking a room that is equipped with capillary pipes inside the ceiling and being connected to a chiller as an example.

Keywords: Multi domain simulation; dynamic building simulation; simulation of building installations

1 Introduction

In order to investigate the dynamic thermal behavior of buildings in conjunction with hydraulic networks and new kinds of building heating and cooling systems a number of component models have been developed at the Hermann–Rietschel–Institute (HRI).

The main focus of the HRI is to research thermal building behavior and thermal comfort with different approaches of air-conditioning. Comparing the power consumption and analyzing the employed control strategies are also important topics.

One approach for example is room heating or cooling with large area heat exchangers [1]. Due to the increased heat exchanging surface it is possible to reduce the temperature difference that must be applied to sufficiently heat the room. This in return leads to a lower amount of needed exergy for this heating system minimizing primary energy usage. An appliance is a panel heating system which could be powered by the relatively cool return water of a district heating system. This would further increase

the efficiency of the plant providing the heating water and could spare heating costs at the same time since the return water could be bought cheaply. The HRI will focus on this topic in an upcoming study.

Another application is a large area cooling system employing capillary pipe mats. With capillary pipe mats the achievable relative heat flow density is higher than with conventional panel heating and cooling systems. It is possible to even more reduce the surface excess temperature at the same level of cooling power.

This makes it possible to reduce the amount of energy that is needed to cool the room because the efficiency of the chiller increases. This holds true for a heat pump system that is often used for such purposes.

Computer simulations must be carried out to examine the feasibility of new air-conditioning technologies as well as to find ways to optimize standard technologies. The HRI uses Modelica as modeling language to benefit from the possibility to create multi domain simulation models.

To show that multi domain simulation models can be built with the HRI libraries an example simulation setup will be discussed in this paper. The design of several components used in this simulation will be described also.

2 Simulation setup

An exemplary simulation setup is shown in Fig. 1. The graphical representation of the system shows a single room model (1), a panel cooling system (2) that is integrated into the room ceiling, a chiller machine (4) and a heat exchanger (3) that would be used to separate the heat pump circuit from

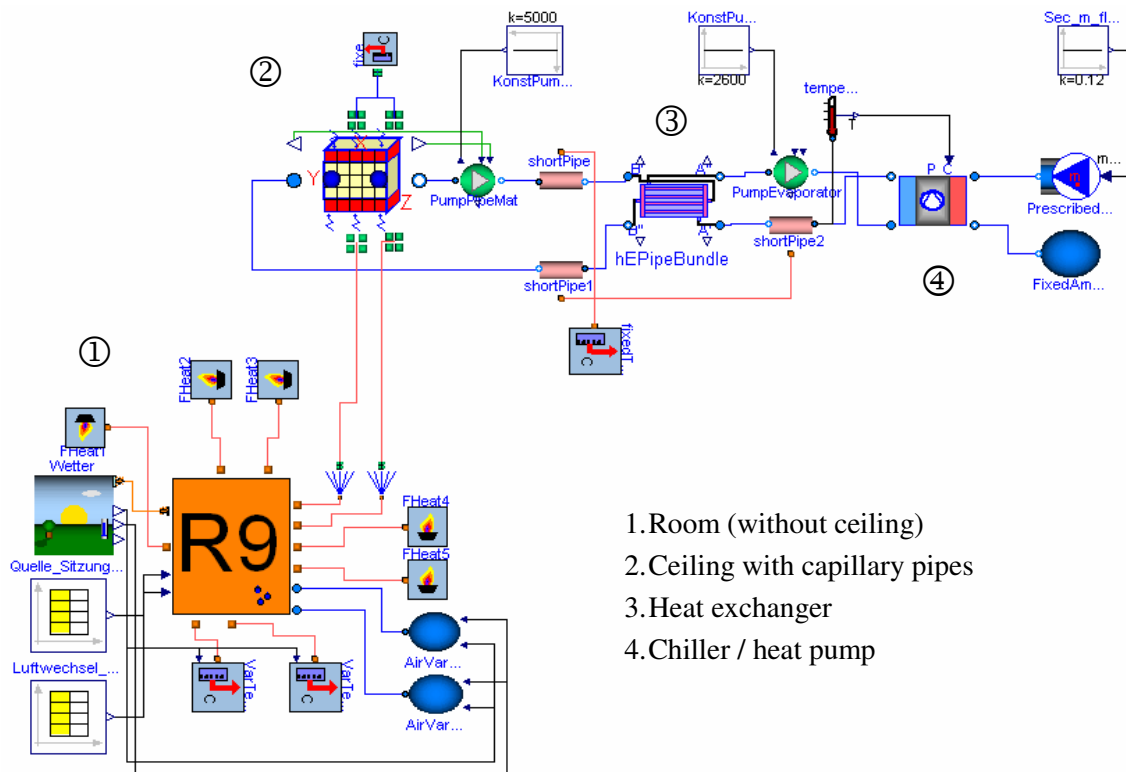


Fig. 1: Graphical representation of the simulation model.

the oxygen enriched capillary pipe mat water system.

Additionally, pipe models connect the mentioned system components and two pumps enforce the necessary fluid flows. The condenser side of the heat pump model is connected to a source with constant medium properties and constant mass flow rate. This would correspond to a supply with ground water.

Room heat loads and air exchange rates are provided by tabulated data. Ambient temperature, air humidity, atmospheric pressure and radiation values are provided by a weather module which uses a test reference year with tabulated values.

Different boundary conditions for the room are applied in this setup. The exterior wall is connected to modules that set the ambient temperature provided by the weather module. Assuming that all adjacent rooms will have the same temperature leads to the boundary condition that no heat flow through inner walls will occur. This is ensured by a second module type.

Temperature and humidity of the outside air will be provided by two reservoirs. The weather module sets the appropriate media properties.

In the following some of the components used in the simulation setup are described in more detail.

3 Simple Room Model

The room model contained in the example represents a cubical room that is enclosed by walls consisting of several layers (see Fig. 2). Three of the four walls are interior walls, the fourth one – an exterior wall – contains a window. Also the floor is modeled with several layers with different material properties.

The model of the ceiling has been deleted from the aggregated room model and was substituted by an external model of a ceiling with embedded capillary pipes (see Fig. 1). These capillary pipes are used to cool the room and act as an interface between the room and the building services installations.

3.1 Radiation Exchange

In addition to basic physical phenomena like heat transport and natural convection, also long wave and short wave radiation interchange are being examined.

Besides the exact approach to calculate long wave radiation exchange between room surfaces basing on angle factors we also implemented a simpler model using an area weighted approach to calculate radiation exchange. This is especially important for assembling rooms that are more complex than a simple

“match box” shaped room. In the latter case it would be a challenging and error prone task to determine all necessary angle factors to describe radiative heat exchange.

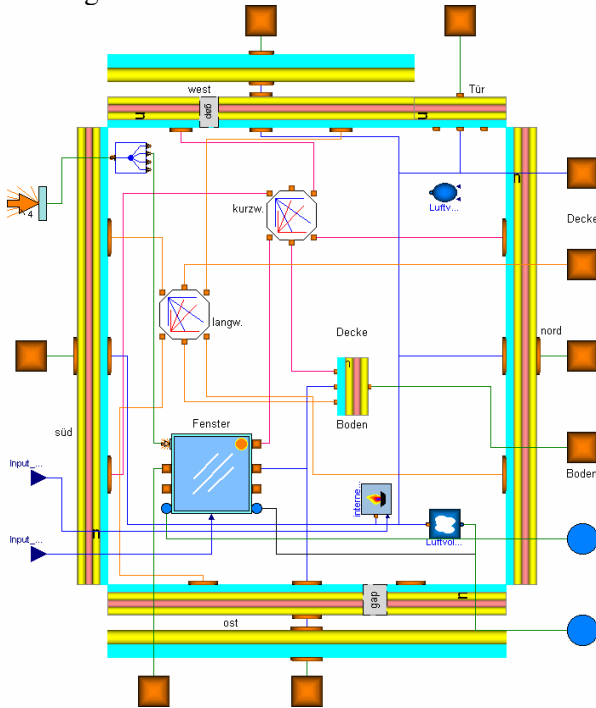


Fig. 2: Graphical representation of the room model.

3.2 Air Volume and Boundary Conditions

The air volume inside the room includes the property of humidity, which is obtained by connecting the model to the *Modelica.Media* library.

In order to observe local condensation on the chilled ceiling surface, we added a sensor that continuously checks the local dew point temperature.

Regarding the interior walls and the floor we assume adiabatic boundary conditions for this simple example – which imply equal temperatures inside the simulated room and the adjacent rooms. So heat transport only takes place through the exterior wall, the window, and the ceiling. A weather model provides the necessary data concerning temperature, humidity, and pressure of the ambient air as well as the solar radiation onto the exterior wall and the window. This is based on measured data called “test reference year”, which is provided by the German weather forecast agency *DWD*.

Finally, the control of the inner heat sources inside the room as well as the air exchange through the window are being realised by connecting external files to the simulation model.

4 Building Services Installations

Most of the hydraulic components are black box models. Due to the fact that design specifications are not available from most component manufacturers it is a superior aim for the HRI to model components based on the provided manufacturer data as input parameters and approximately reproduce the components behavior. Trying to model the component as exactly as possible by modeling each concerned physical phenomenon would transcend the needs of the Hermann–Rietschel–Institute. The latter proceeding would be favorable if the behavior of a specific apparatus must be examined or tested. This would normally include to rewrite a considerable amount of code if a different apparatus design shall be examined.

4.1 Panel Heating and Cooling Model

As stated in [1] a model of a panel heating and cooling system was developed at the HRI. Pipes embedded in a wall layer provide heating or cooling. The hydraulic and thermal behavior of the system is modeled in two components:

- the pipe mat model and
- the wall model.

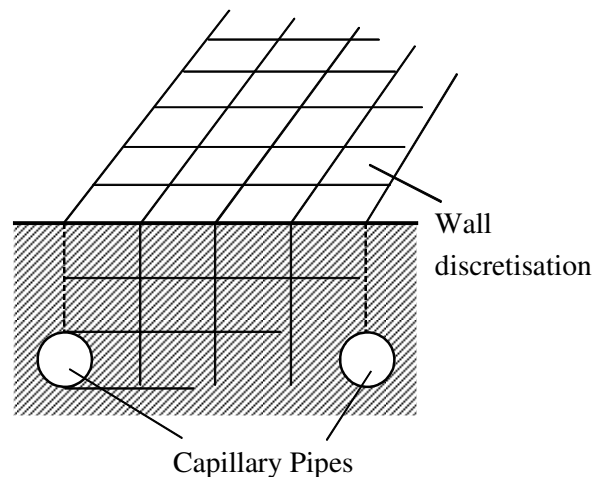


Fig. 3: Illustration of the capillary pipe mat model.

The wall model consists of discrete elements (finite volumes) that conduct and store heat energy allowing for the examination of the temperature distribution in the wall layer. Capillary pipes are built from of a number of small pipe elements connected together to build the large pipe mat model. The pipe model allows heat transfer from the fluid model to the wall model.

The short pipe model contained in the *Modelica_Fluid* library is used to facilitate the pressure loss calculation.

Since the first presentation the panel heating and cooling model has been extended and now provides support for two types of capillary pipe mats (see Fig. 4) and several wall layers to simulate realistic wall designs.

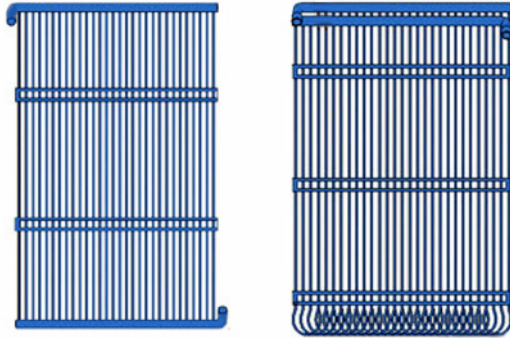


Fig. 4: implemented pipe mat types.

A model of a capillary pipe mat system as described by B. Glück in [3] was replicated and simulation results were compared to the findings of Glück. It could be shown that the replicated model had very small differences in the area-related heat energy storage (Wh/m^2) of below 5% depending on the grid spacing. Some differences between the models remained (unknown material properties of one ceiling layer, modeling of pipes, calculation of heat transfer coefficients at ceiling surface) and will be a reason for the differences.

4.2 Simple Radiator Model

To be able to model a standard European heating system a radiator black box model has been developed. It features radiative and convective heat transfer with the consideration of the influences of ambient air pressure, radiation and air temperature. Parameterization is easy with catalogue data (Mass of steel, water volume, length ...) provided by the manufacturer. Correction factors accounting for the situation of installation, coating and other influences can be provided as *parameters*. Fig. 5 shows the graphical representation of this model.

The calculation of current heat flow is based on the ratio of current heater surface excess temperature and its nominal value:

$$\frac{\dot{Q}}{\dot{Q}_{nom}} = \left(\frac{\Delta T_{log}}{\Delta T_{log,nom}} \right)^{eh}$$

Since the exponent of the heating surface eh is only valid for a small range around nominal conditions an

approach applying correction factors is used in order to obtain a reasonable behavior under part load conditions. B. Glück describes the algorithm in [4].

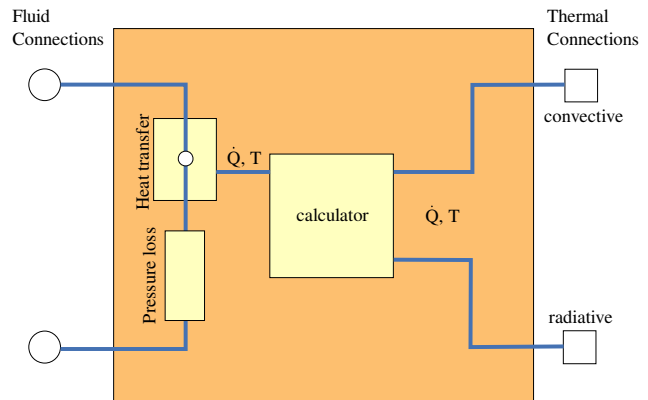


Fig. 5: Illustration of radiator model.

4.3 Simple Heat Exchanger Model

The heat exchanger component used in the described simulation model is a shell and tube heat exchanger with a number of discrete pipe elements to model the heat exchange between shell and tube side fluids.

Compared to the simple pipe model heat convection will now be computed by a variable heat transfer coefficient depending on geometric attributes, fluid velocity and temperature. Algorithms for various kinds of heat convection effects on pipes are based on [2].

Appropriate functions for the heat convection effect can be chosen in the graphical user interface. Because the effect is computed in a separate sub-module it is easy to implement more heat convection effects for other surface geometries.

Currently a one pass heat exchanger with plain tubes is implemented. The structure of the component is depicted in Fig. 6.

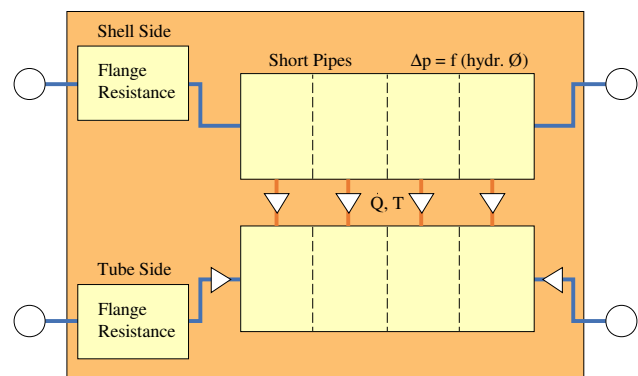


Fig. 6: Illustration of tube and shell heat exchanger.

Thermal effects can be described using an adapted mandrel pipe model [2]. The shell side tube contains a number of inner tubes. Heat transfer between the fluids is influenced by the heat convection coefficients at the tubes – shell side and tube side. The coefficients are calculated using an equation for fluid flow in pipes found by Gnielinski [2]. For shell and tube side the same equation applies in this case except that the equivalent hydraulic diameter must be provided for the shell side fluid flow.

Instead of modeling all tubes directly, only a single tube will be used to reduce model size and computation time. This is feasible under the assumption of equal temperature distribution normal to the flow direction. Total mass flow rate through the pipe must be reduced to reflect the real flow conditions inside a single tube. Additionally, the transferred heat flow towards the tube side fluid must be adjusted to account for the reduced number of tubes.

The heat exchanger is divided into sections alongside the tube direction to solve for the time and location dependent differential equation.

Hydraulic effects are considered for the flange taps and the heat exchanger's internal fluid flow. Shell side pressure loss for longitudinal flow in tube bundles can be calculated as for fluid flow inside a single tube. In this case the hydraulic diameter of the shell side must be used instead of the pipe diameter.

An option is implemented to regard deposits at the tubes surface with a definable thickness.

Currently a new heat exchanger component is being developed at the HRI. It is based on characteristic values of different types of heat exchangers and various flow types according to the theory in [2]. This generic model will allow covering a broader range of heat exchanger designs in simulations.

4.4 Heat Pump Model

Fig. 7 shows the simplified illustration of the black box heat pump model. There are two heat exchangers (condenser and evaporator) which allow exchanging energy between the medium and the heat pump cycle.

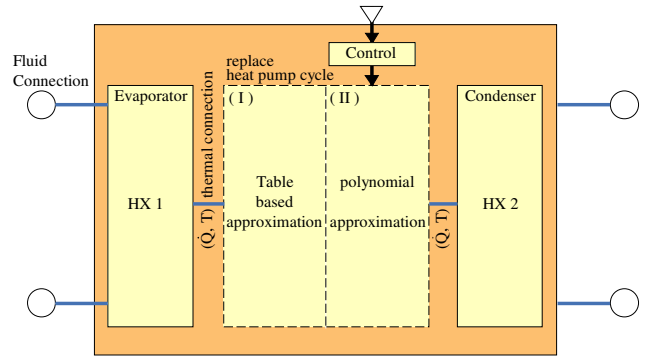


Fig. 7: Illustration of heat pump model.

Two models exist for the inner heat pump cycle. A table based approach offers to easily apply manufacturer data for evaporator power \dot{Q}_0 , condenser power \dot{Q}_c and engine power consumption P_e . Cooling power and engine power will be interpolated by the *CombiTable2D* module contained in the Modelica Standard Library.

A second model applies polynomial approximations to calculate volumetric cooling power

$$\dot{q}_{0W} = \dot{Q}_{0,nom} / \dot{V}_W$$

of a piston compressor as a function of condenser and evaporator temperature, T_c and T_0 respectively. Volumetric cooling power \dot{q}_{0W} depends on the refrigerant used and the construction of the piston compressor. Approximate values for two refrigerants R404 and R134a used in a variety of piston compressors are known [5]. \dot{V}_W may be calculated using geometric data of the piston which is provided by the manufacturer.

The second approach allows to adjust the cooling power of the generic heat pump very easily by adapting piston size. The first model on the other hand will be more accurate because it employs manufacturer data for a specific heat pump construction.

Power regulation of the heat pump is implemented as continuous speed regulation which is modeled by the ratio of current to nominal speed where the speed range can be chosen freely. Since evaporator power depends on the refrigerants volume flow rate which is directly proportional to the machine speed one can scale nominal cooling power linearly with the speed ratio:

$$\dot{Q}_{0,nom} = \dot{V}_{W,nom} \cdot \dot{q}_{0W}$$

and

$$\frac{\dot{V}_W}{\dot{V}_{W,nom}} = \frac{n}{n_{nom}}$$

Power consumption can roughly be estimated using the affinity law:

$$\frac{P}{P_{nom}} = \left(\frac{n}{n_{nom}} \right)^3$$

To better describe the dynamic behavior of a heat pump in our simulations the model features temperature protection implemented by the use of two boundary temperatures. By default an on-off controller will turn the pump off when the controlled temperature T_{target} drops below temperature T_1 and start the machine again when the T_{target} rises above T_2 . When the pump turns off it will come to a halt within a definable time period. It will not start again until a minimum amount of time has passed. Besides those implementations there is no modeling of dynamic behavior of the heat pump.

4.5 Other Components

Several additional components are needed to model hydraulic networks. The *HVAC-HRI* library contains most of these components. The models are designed to work with different fluids – mainly water and air in building installations.

An extended short pipe model based on the model of the *Modelica.Fluid* library allows simulating heat transfer from the fluid through the pipe wall, as well as heat storage in the pipe material. The pipe wall consists of two layers to being able to model an insulation layer. An aggregation of a sufficiently high number of these short pipe models allows for the simulation of long pipes.

Single hydraulic resistor elements can be represented by a model applying friction coefficients as is often done when designing hydraulic networks. Based on friction coefficients a three-way pipe model is also available.

Valve models exist for two-way and three-way valves. Additionally, a thermostat head can be combined with the two-way valve to form a heat valve that would normally be used with the radiator model as described in section 4.2.

Two pump models are currently available: one model defines the pump's hydraulic curve (head is a function of mass flow rate) by an n^{th} grade polynomial. Different pump speeds are modeled by employing the affinity law.

The second pump model uses tabulated values for pump head and volume flow rate. Therefore, the pump curves can be modeled very precisely as well as the electrical power consumption of the pump. Several control strategies are available for both

pump models. For example constant pump speed, constant pump head and constant volume flow rate.

A generic heater and chiller model allows defining a heat source or heat sink in a hydraulic circuit. A derived model allows defining the outflow temperature depending on an input value. This allows for the employment of heater flow temperatures depending on ambient air temperature. Day and night modes are also possible.

4.6 Exemplary Results

An exemplary result of a simulation with the described model is shown in Fig. 8 and Fig. 9. Two very simple control strategies of the heat pump model have been compared to a simulation without cooling.

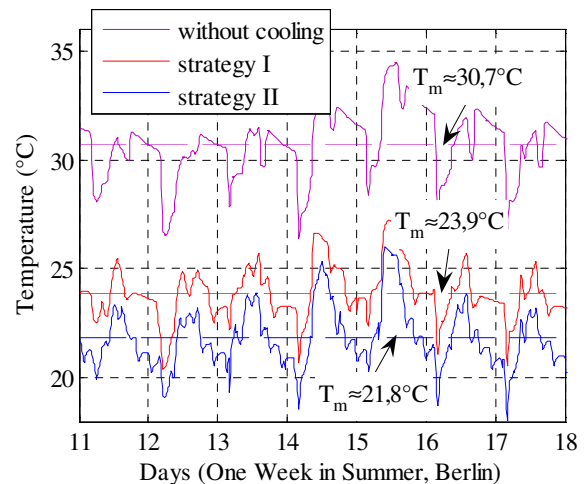


Fig. 8: Room air temperatures of example simulation.

Strategy I uses evaporator outflow temperature to control the heat pump power as displayed in Fig. 1. In strategy II the capillary pipe mat inflow temperature was controlled. The temperatures were lower in this case compared to the values in strategy I.

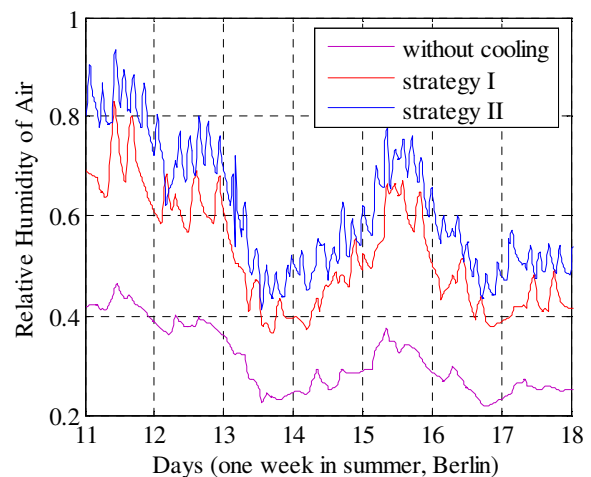


Fig. 9: Humidity of air near room ceiling.

The power control included in the heat pump continuously adapts heat pump power to the input temperature signal. This input signal later can also be provided by a more sophisticated controller algorithm.

One can clearly see the cooling effect of both strategies on room temperature. Because the second strategy yielded lower fluid temperatures in the pipe mat the cooling effect is stronger than with strategy I.

The graphs in Fig. 9 display humidity of air near the cooling panel for both strategies and without cooling. Relative humidity performs as expected according to the mean surface temperature of the ceiling.

The control strategies employed in the described example simulation have not been optimized. Detailed examination of the system behavior considering special requirements (e.g. a maximum humidity level) would allow finding optimized control strategies.

5 Conclusions and Perspectives

The feasibility of thermally and hydraulic coupled dynamic building simulations could be shown. It is possible to test and verify different control strategies of the components. This allows for the research of energetically optimized installations in buildings.

Different approaches to design a model are often useful to account for the specific demands on a model.

Further development will concentrate on the refinement of the models developed at the HRI in combination with larger scale building simulations. New components will also be developed to cover a wider range of building services installations.

Upon completion of new test facilities at the HRI validation of several components and systems will be possible. For example a testing environment with capillary pipe mats for the air-conditioning of two rooms is currently being installed.

List of symbols

eh	exponent of heating surface
n	rotational speed of engine
\dot{Q}	heat flow rate
\dot{q}_{ow}	volumetric evaporator power (W/(m ³ /s))
P	engine Power
T	Temperature
\dot{V}	volume flow rate
Indices	
$l, 2$	minimum, maximum value
$target$	target value
c	condenser
o	evaporator
w	piston working volume
nom	nominal value
log	logarithmic value

References

- [1] A. Hoh, T. Haase, T. Tschirner, D. Müller: A combined thermo-hydraulic approach to simulation of active building components. 4th International Modelica Conference, March 2005, included on the CD-ROM version
- [2] VDI Wärmeatlas, Berechnungsblätter für den Wärmeübergang, Düsseldorf, Germany, 1988
- [3] B. Glück: Thermische Bauteilaktivierung. Heidelberg, Germany, 1999, ISBN: 3-7880-7674-7
- [4] B. Glück: Bausteine der Heizungstechnik, Wärmeabgabe von Raumheizflächen und Rohren. Germany, 1990, ISBN: 3-345-00515-8
- [5] Recknagel, Sprenger, Schramek: Taschenbuch für Heizung und Klimatechnik, Oldenburg Industrieverlag, 2005

MWorks: a Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica

FAN-LI Zhou, LI-PING Chen, YI-ZHONG Wu, JIAN-WAN Ding, JIAN-JUN Zhao, YUN-QING Zhang

CAD Center, Huazhong University of Science and Technology, China

fanli.zhou@gmail.com, chenlp@hustcad.com, cad.wyz@gmail.com, jwdingwh@gmail.com, zhaojj@hustcad.com, zhangyq@hustcad.com

Abstract

MWorks is a platform for modeling and simulation of multi-domain physical systems. It is a modern integrated development environment (IDE) integrating with visual modeling, translator, optimizer, solver and postprocessor. MWorks implements all the syntax and most semantics of Modelica 2.1.

This paper first describes the features of MWorks as a modern IDE, and then gives the detailed descriptions of special Modelica semantic implementation in translator and self-adapting solving strategies in solver of MWorks.

Keywords: IDE; Modeling and Simulation; Multi-domain Physical Systems; Modelica

1 Introduction

MWorks is a general modeling and simulation platform for complex engineering systems which supports visual modeling, automatically translating and solving, as well as convenient postprocessing. The current version is based on Modelica 2.1 and implements all the syntax and most semantics of Modelica.

MWorks has features as follows:

- With modern integrated development environment styles, it provides friendly user interfaces such as syntax high-lighting, code assist etc.;
- Based on object-oriented compiler framework, it perfectly supports almost all the syntax and semantics of Modelica;
- Using self-adapting solving strategies, it can agilely solve differential equations, algebraic equations and discrete equations.

The version 1.0 of MWorks will completely support Modelica 2.2, and the current version is 0.8 which realizes the most semantics of Modelica 2.1.

2 Framework of MWorks

MWorks is a general modeling and simulation platform which consists of studio, translator, optimizer, solver and postprocessor.

The main process is similar to the described in book of Peter Fritzson [1], shown in Figure 1.

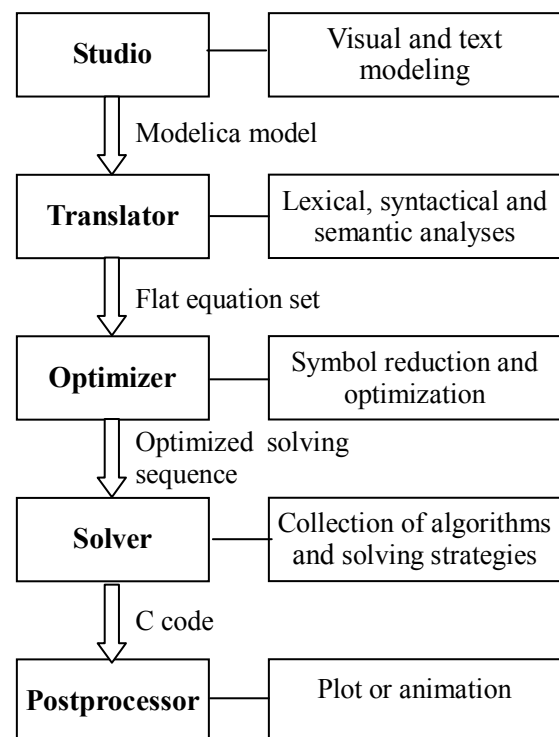


Figure 1. Main process of MWorks

The studio is an integrated development environment which integrates the visual modeling interface with other modules of MWorks. The translator is a compiler of Modelica by which an equation system of model will be generated after lexical, syntactical and semantic analyses. The optimizer completes symbol reduction based on graph theory and gives an

optimized solving sequence. The C code of the model will be emitted after compiling, analyzing and optimizing. The simulation of model is driven by the C code through calling the solver. The solver includes the collection of algorithms for algebraic equations, ordinary differential equations, differential-algebraic equations and discrete equations. Its core is the self-adapting solving strategies. The result of the simulation is displayed on postprocessor in a plot or animation way.

3 MWorks Studio: Visual Modeling Environment and IDE

MWorks Studio is a visual modeling environment which supports drag-drop modeling based on Modelica Standard Library. It is also an integrated development environment integrating with translator, optimizer, solver and postprocessor.

As a developing tool, this studio provides many modern IDE styles to promote the users' conveniences just as Eclipse or Microsoft Visual Studio does, such as real-time syntax highlighting, content assist, code formatting, outlining etc..

The snapshot of MWorks Studio is shown as Figure 2.

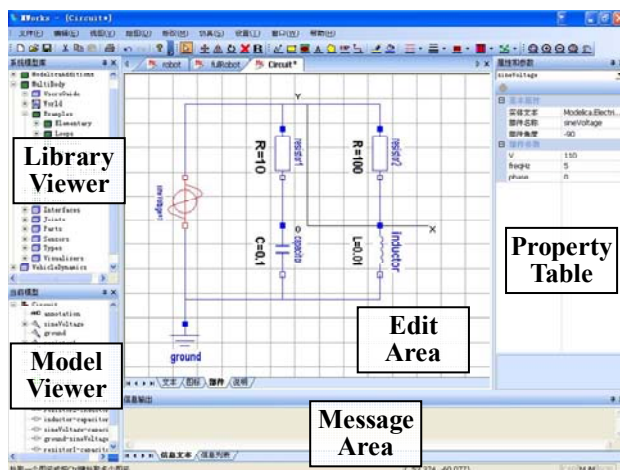


Figure 2. Snapshot of MWorks Studio

The library viewer illustrates all predefined system libraries, all loaded user libraries and other top models in the memory. The model viewer shows all components of the current model. The edit area is visual modeling, text modeling, icon-editing or information area, and the status is chosen by tag. The property table displays all properties of selected element in the model, and the properties can be edited here. The message area displays all messages in the checking, translating, or simulating, including status

and error messages. The error can automatically be located by double clicking error message.

The auxiliary functions of real-time syntax highlighting, content assist, code formatting and outlining are provided in the text modeling status.

4 MWorks Translator: Modelica Compiler and Equations Generator

The tasks of translator are to perform lexical, syntactical and semantic analyses for the model files and generate equation systems for the models. They are accomplished by three time parsing: lexical and syntactical parsing, resolving of model and instantiation of model.

The design and implementation of translator are based on object-oriented framework, which is obtained in the first parsing. The designs of all the semantic mechanisms, which are implemented in the second and third parsing, are also based on the framework to perfectly support Modelica.

4.1 Three Times Parsing

Three times parsing should be done for a complete translating of the main model for simulation in MWorks.

4.1.1 Lexical and Syntactical Analysis

The lexical and syntactical analysis is performed in the first parsing by using ANTLR tool. The ANTLR is a convenient, object-oriented, automotive lexical and syntactical analysis tool [2]. The result of the first parsing is Document Object Model (DOM) tree that is object-oriented container presentation of Abstract Syntax Tree (AST). The class hierarchy of DOM in MWorks is shown as Figure 3.

The class hierarchy of DOM is abstracted from the EBNF description of Modelica language specification (MLS) [3]. Each class in the hierarchy is consistent with the corresponding element in the EBNF. Three main class groups are noticeable in the hierarchy. Element, expression and behavior are respective abstract base class of the three groups. All operations of the translator are based on the DOM, from semantic checking to generating equation system.

4.1.2 Semantic Resolving

The semantic resolving based on DOM tree is the main content of the second parsing, including collecting information for checking types and resolving extends clauses, modifications (general modifica-

tions and redeclarations), outer-inner matches, connect clauses, and so on. The realization of UBD (use

before declaration) is easy in the second parsing based on DOM.

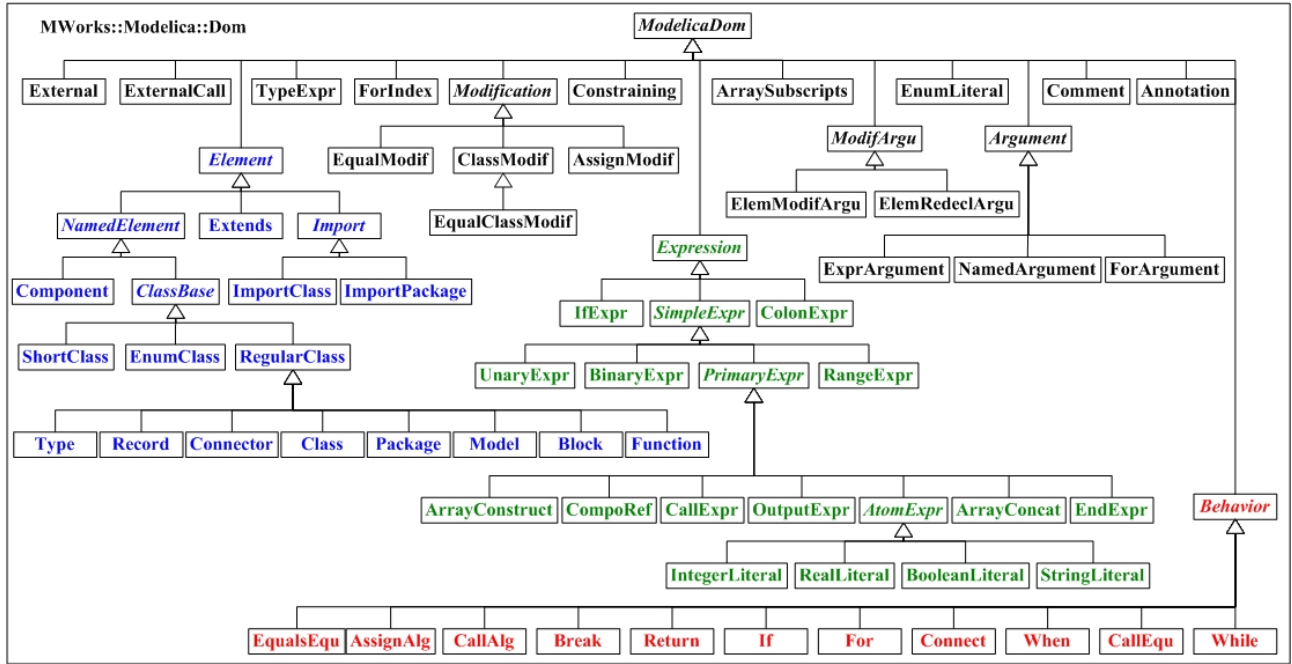


Figure 3. Class hierarchy of DOM

The implementation of semantic resolving is mainly concentrated into three classes in DOM: RegularClass, Component and ShortClass. Their resolving algorithms are as follows.

Algorithm 1: Resolving of Regular Class

1. Checking circular type definition (circular inheritance);
2. Validating type prefixes;
3. Resolving outer-inner (matching inner element for outer element);
4. Resolving extends clauses (looking up the base class);
5. Checking repeated names of the named elements;
6. Resolving member components (calling resolving of component);
7. Checking type restriction;
8. Deducing type prefixes;
9. Resolving modifications (collecting information for modifications, including redeclaratoins);
10. Resolving connections (collecting information for generating connection equations).

The nested classes are not resolved in the resolving of regular class, and they are resolved when they are used as types, such as component types, reference types of short classes or base classes.

Algorithm 2: Resolving of Component

1. Validating component prefixes;

2. Looking up type of the component;
3. Resolving the type if unresolved (calling resolving of type);
4. Checking the validity of the type;
5. Considering redeclaration of the type;
6. Resolving type expression of the component (expanding the type of the component type);
7. Deducing component prefixes;
8. Resolving modifications (collecting information for modifications, including redeclaratoins).

The resolving of component is called when resolving its parent regular class.

Algorithm 3: Resolving of Short Class

1. Checking circular type definition (circular reference);
2. Validating type prefixes;
3. Looking up reference type;
4. Resolving the type if unresolved (calling resolving of type);
5. Considering redeclaration of the type;
6. Resolving type expression of the short class (expanding the type of the short class);
7. Deducing type prefixes;
8. Resolving modifications (collecting information for modifications, including redeclaratoins).

The resolving of short class is called when resolving the element that uses the short class as type of

component, reference type of short class or base class of regular class.

4.1.3 Instantiation

The semantic resolving prepares necessary information for instantiation which is performed in the third parsing. The purpose of instantiation is mainly to generate equation system (continuous equations and discrete events) for the main model.

Similar to the resolving, the implementation of instantiation is concentrated into the same three classes in DOM. The algorithms are as follows.

Algorithm 4: Instantiation of Regular Class

1. *Merging modifications (defining valid modifiers, including redeclarations);*
2. *Instantiating base classes (calling instantiation of the base class);*
3. *Instantiating member components (calling instantiation of the component);*
4. *Instantiating equation clauses to generate equations;*
5. *Instantiating algorithm clauses to generate equations or definition of function (if regular class is function the instantiation should be handled specially).*

According to the regular class being container of elements, the instantiation of regular class is the console of calling all element instantiations. Additionally, it generates the direct equations of the regular class itself.

Algorithm 5: Instantiation of Component

1. *Dealing with component prefixes;*
2. *Looking up redeclared component or type if redeclaration is valid;*
3. *Looking up inner type if the component type is outer;*
4. *Evaluating subscripts of component type if the component is an array;*
5. *Merging modifications (defining valid modifiers, including redeclarations);*
6. *Generating variables for built-in component (considering array) or calling instantiation of type for complex component.*

The instantiation of component is mainly to generate variables for built-in component or to call the instantiation of type for complex component.

Algorithm 6: Instantiation of Short Class

1. *Looking up inner type if the reference type is outer;*

2. *Merging modifications (defining valid modifiers, including redeclarations);*

3. *Instantiating reference type (calling instantiation of the reference type).*

The instantiation of short class is mainly to call the instantiation of the reference type.

4.2 Lookup Mechanism

Lookup is the most basic mechanism for Modelica translating. The type resolving of component declarations, extends clauses and import clauses depends on standard static lookup, and the outer-inner matching depends on dynamic lookup. Besides both of them, the special lookup mechanisms are necessary for supporting modification and array. All of the lookup mechanisms are implemented as appropriate interfaces in DOM class hierarchy.

The static lookup mechanism, which is described in the Modelica language specification, is induced as two virtual interfaces of DOM class NamedElement and one top static interface of the root context class. The two virtual interfaces are lookup_type() and lookup_comp(), and the static interface is lookup_type_from_top(). The interface lookup_type() is used in the lookup of component type, base type or reference type of short class; The interface lookup_comp() is called in the lookup of component in expression or modification; The interface lookup_type_from_top() is specially designed for the lookup of import package or class.

The dynamic lookup, i.e. matching of outer and inner, is very complicated because of the freedom of the situation and kind of the outer element. The outer element may appear anywhere in model, and may be type or component. The implementation of the match depends on stack of both of type and component in the resolving and instantiation of the main model.

Special lookup mechanisms are imported to support the resolving of modification and the evaluation of array subscripts. The imported interface is to lookup replaced type or component for resolving modification and lookup modifier in the modifier container for evaluating value of array subscripts. The modifier container is built in the resolving of modification. Not only array subscripts but also all parameters, their values are evaluated by looking up modifier when the evaluation is necessary.

According to these lookup mechanisms, many Modelica semantics get perfectly implemented, such as circular extends check and complex redeclaration semantics.

4.3 Modelica Semantic Mechanisms

To support modeling and simulation of complex engineering physical systems, Modelica defines complex semantics, and it can be abstracted as the following basic mechanisms: type, extends, general modification (except redeclaration), redeclaration, outer-inner, connect, array and algorithm mechanism.

Some of these mechanisms are very complex, especially, some of them may cause couple effects. For examples, the couple influence of redeclaration and out-inner must be considered everywhere in implementing translator.

The core of the type mechanism is type checking, in which the short class and array should be considered. The type resolving is done in the second parsing and the type checking is finished in the third parsing.

The extends mechanism is easily realized based on lookup mechanism.

The implementation of modification is done by collecting modification to build modifier container in the resolving and merging modifiers to define valid modifiers in the instantiation.

The process of matching of outer-inner is similar to the modification, which is realized by operations in both resolving and instantiation.

It is handled according to the steps described in Modelica specification to translate connect clauses to direct equations.

Array is also very complicated. Comparing with general programming languages, Modelica allows variables as subscripts, and must translate the array to single equation for solving, especially, it allow applying modification to array. If no constraints were set to modification of array, the compiling would be extremely complicated, such as modification to array subscripts. MWorks chooses appropriate simplification just like Dymola does.

Algorithm in model is not same as equation, and the flow analysis should be done for its generating equations. Though function is a class with special algorithm, but generating equations for the algorithms in function is thoroughly different from the algorithms in model. The functions are directly translated into C functions.

Based on DOM class hierarchy, these mechanisms have been dealt with well, and the couple mechanisms are also considered.

5 MWorks Solver: Collection of Algorithms and Console of Solving Strategies

The solver of MWorks includes two primary modules: collection of algorithms and console of solving strategies. In fact, the optimizer shown as Figure.1 should be a part of the solver. Here it is skipped, and its implementation for MWorks is described in the paper [4].

Solver provides different basic algorithm alternatives for users to select appropriate one. For examples, users can select one-step method series or multi-step method series for ODE/DAE problems. The different basic algorithms for differential equation and algebraic equations are collected in solver.

The consistent interface framework is designed for solver to conveniently call different basic algorithms. Each basic algorithm can be easily integrated into the solver by simply encapsulating it according to the template. Now, a series of algorithms for different kinds of equations have been collected in the solver, such as SUNDIALS [5].

It is the task of console of solving strategies to solve continuous-discrete hybrid problem. Solver controls the solution of problem based on basic algorithms according to information collected in translator.

The solving of continuous-discrete hybrid problem in Modelica is according to the principle of synchronous data flow [6]. All the events are collected in the third parsing, and solver will monitor them in the simulation.

6 Example

Here an example is given to show the visual modeling and simulation in MWorks. The visual model of the example is shown as Figure.1, and the text model is as follows (annotations are skipped),

model Circuit

Modelica.Electrical.Analog.Sources.SineVoltage AC(V=110, freqHz=5);

Modelica.Electrical.Analog.Basic.Ground G;

Modelica.Electrical.Analog.Basic.Resistor R1(R=10);

Modelica.Electrical.Analog.Basic.Resistor R2(R=100);

Modelica.Electrical.Analog.Basic.Capacitor C(C=0.01);

Modelica.Electrical.Analog.Basic.Inductor L(L=0.1);

equation

connect(AC.p, R1.p);

connect(R2.p, R1.p);

connect(R2.n, L.p);

```

connect(L.n, C.n);
connect(AC.n, C.n);
connect(G.p, AC.n);
connect(R1.n, C.p);

```

```
end Circuit;
```

The following is the equations generated by the translator. There are 34 variables and 34 equations excluding parameters and constants.

```
model Circuit
```

```

parameter Real AC.offset = 0;
parameter Real AC.startTime = 0;
parameter Integer AC.signalSource.nout = 1;
parameter Integer AC.signalSource.outPort.n =
AC.signalSource.nout;
parameter Real AC.signalSource.amplitude[1] = AC.V;
parameter Real AC.signalSource.freqHz[1] = AC.freqHz;
parameter Real AC.signalSource.phase[1] = AC.phase;
parameter Real AC.signalSource.offset[1] = AC.offset;
parameter Real AC.signalSource.startTime[1] =
AC.startTime;
constant Real AC.signalSource.pi = Modelica.Constants.pi;
parameter Real AC.signalSource.p_amplitude[1] =
AC.signalSource.amplitude[1]*1;
parameter Real AC.signalSource.p_freqHz[1] =
AC.signalSource.freqHz[1]*1;
parameter Real AC.signalSource.p_phase[1] =
AC.signalSource.phase[1]*1;
parameter Real AC.signalSource.p_offset[1] =
AC.signalSource.offset[1]*1;
parameter Real AC.signalSource.p_startTime[1] =
AC.signalSource.startTime[1]*1;
parameter Real AC.V = 110;
parameter Real AC.phase = 0;
parameter Real AC.freqHz = 5;
parameter Real R1.R = 10;
parameter Real R2.R = 100;
parameter Real C.C = 0.01;
parameter Real L.L = 0.1;

```

```

Real AC.v;
Real AC.i;
Real AC.p.v;
Real AC.p.i;
Real AC.n.v;
Real AC.n.i;
Real AC.signalSource.outPort.signal[1];
Real AC.signalSource.y[1];
Real G.p.v;
Real G.p.i;
Real R1.v;
Real R1.i;
Real R1.p.v;
Real R1.p.i;
Real R1.n.v;

```

```

Real R1.n.i;
Real R2.v;
Real R2.i;
Real R2.p.v;
Real R2.p.i;
Real R2.n.v;
Real R2.n.i;
Real C.v;
Real C.i;
Real C.p.v;
Real C.p.i;
Real C.n.v;
Real C.n.i;
Real L.v;
Real L.i;
Real L.p.v;
Real L.p.i;
Real L.n.v;
Real L.n.i;

```

```
equation
```

```

AC.v=AC.p.v-AC.n.v;
0=AC.p.i+AC.n.i;
AC.i=AC.p.i;
AC.signalSource.y[1] = AC.signalSource.outPort.signal[1];
AC.signalSource.outPort.signal[1]=AC.signalSource.p_offset[1
]+(if time<AC.signalSource.p_startTime[1] then 0 else
AC.signalSource.p_amplitude[1]*Modelica.Math.sin(2*AC.sign
alSource.pi*AC.signalSource.p_freqHz[1]*(time-
AC.signalSource.p_startTime[1])+AC.signalSource.p_phase[1])
);
AC.v=AC.signalSource.outPort.signal[1];
G.p.v=0;
R1.v=R1.p.v-R1.n.v;
0=R1.p.i+R1.n.i;
R1.i=R1.p.i;
R1.R*R1.i=R1.v;
R2.v=R2.p.v-R2.n.v;
0=R2.p.i+R2.n.i;
R2.i=R2.p.i;
R2.R*R2.i=R2.v;
C.v=C.p.v-C.n.v;
0=C.p.i+C.n.i;
C.i=C.p.i;
C.i=C.C*der(C.v);
L.v=L.p.v-L.n.v;
0=L.p.i+L.n.i;
L.i=L.p.i;
L.L*der(L.i)=L.v;
R1.p.v = AC.p.v;
R2.p.v = AC.p.v;
R2.n.v = L.p.v;

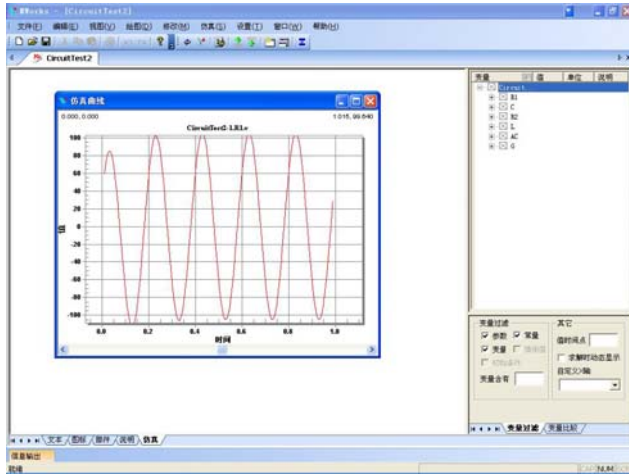
```

```

C.n.v = AC.n.v;
G.p.v = AC.n.v;
L.n.v = AC.n.v;
R1.n.v = C.p.v;
AC.p.i+R1.p.i+R2.p.i = 0;
L.p.i+R2.n.i = 0;
AC.n.i+C.n.i+G.p.i+L.n.i = 0;
C.p.i+R1.n.i = 0;
end Circuit;

```

The result is displayed in the postprocessor, shown as Figure. 4.



7 Conclusions

MWorks is a modern IDE for modeling and simulation of multi-domain physical systems based on Modelica. All the syntax and most semantics of Modelica 2.1 have been implemented. The current version of MWorks can validly deal with some problems based on Modelica Standard Library. The coming version 1.0 of MWorks will completely support Modelica 2.2.

References

- [1] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, 2003.
- [2] <http://www.antlr.org/>
- [3] <http://www.modelica.org/documents/ModelicaSpec21.pdf>
- [4] Ding Jianwan, Chen Liping, Zhou Fanli. A Component-based Debugging Approach for Detecting Structural Inconsistencies in Declarative Equation based Models. Journal of Computer Science & Technology, 2006, 21(3):450-458
- [5] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, C. S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. ACM Transactions on Mathematical Software, 2005, 31(3):363-396.
- [6] Otter M., Elmqvist H., Mattsson S.E.. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99, Hawaii, August 22-27, 1999:151-157.

Domain Library Preprocessing in MWorks - a platform for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica

Wu Yizhong, Zhou Fanli, Chen Liping, Ding Jianwan, Zhao Jianjun
CAD Center, Huazhong University of Science and Technology
Wuhan, Hubei, China: 430074
Cad.wyz@gmail.com, Chenlp@hustcad.com

Abstract

Modeling and simulating with Modelica-based platforms for multi-domain physical systems need support of diversified domain libraries. Modelica Standard Library (MSL) is a free, reusable library released by Modelica Association, which comprises definitions of basic classes belonging to many domains such as mechanical, electrical, hydraulic, control, multi-body, thermodynamics, etc. User can also define or extend his special library. These libraries usually are stored at disk with mo text format (or binary encrypted format). They only record text information of the defined classes, and do not save their structured information. They will be loaded when system starts up, and they can be reused as extends classes or component instances when modeling. When the current main class is being checked or translated, the referenced classes (including extends and component classes) from domain library will be parsed first.

Loading mo format files involves searching of a great many files from the library directory. At the same time, lexical and syntax analysis is performed. So the loading process is time-consuming. Likewise, parsing the referenced classes from domain library when parsing the current main class is also a time-consuming process. So this paper studies a so-called domain library preprocessing technology which can greatly improve loading speed of domain library and parsing speed of the main class. The idea of this approach is that: parsing all the classes within domain library once, building their structured information of document object models, and then saving them to a binary file. When the system starts up, the binary file is automatically serialized to construct structured information of the classes which were defined originally in domain library. This approach is implemented in MWorks – another independent platform for modeling and simulation of multi-domain physical systems based on Modelica language.

Keywords: Modeling and Simulation; Modelica; Domain Library; Preprocessing; Document Object Model

1 Introduction

Modelica language has been developing rapidly during these years[1]. Applications of modeling and simulation based on Modelica have also been emerging in endlessly. However, software implements based on Modelica language can be counted on your fingers. Dymola[2] and MathModelica[3] are currently the most two important and successful commercial software. In addition, other Modelica-based software which are been implemented include OpenModelica[4], MOSILAB[5] and SCICOS[6], etc. Having seen the daylight of Modelica simulation language in future, we have been bending ourselves to implementing an integrated development environment based on Modelica from 2003. And now our work has come into being an alpha stage product – MWorksV1.0a.

MWorks is another Modelica-based platform for modeling and simulation of multi-domain physical systems, which wholly integrating a modeler, a translator, an optimizer, a solver and a postprocessor. Like other platforms based on Modelica such as Dymola, MathModelica, etc., MWorks requires support of domain library including Modelica Standard Library (MSL) and user's special domain library. These libraries can be reused after they are loaded when system starts up. With the development of Modelica language and the technology of multi-domain physical system modeling and simulation, MSL and user's library become more and more complex and larger increasingly. As we know[7], MSL of Modelica2.1 have more than 3,100 defined classes. And now in Modelica2.2 this number has exceeded 4,300. This trend consequentially results in the following two problems:

- 1) Time cost becomes larger during the loading process of domain library. When system starts up, this process is executed automatically. If the library becomes very, very complex and large, the loading process may be unbearable.
- 2) Time cost becomes larger during the translating process of the main simulation class. For loading process does not construct document object model (DOM), translating a model needs building the structured information (i.e. DOM) of the reused classes from domain library at first.

Apparently, simulating a realistic complex model could consume a lot of time for translating the correlative classes from domain library ever built.

To resolve these problems, we put forward and implement the so-called domain library preprocessing technology in MWorks platform. The basic idea of this approach is that: We save all the document object model information of domain libraries through serializing binary DOM files only once when MWorks is set up or starts up for the first time. Afterwards, MWorks will load these DOM files every time during it starts up. Because the DOM file is only one binary file, loading it becomes very quickly. And also because the DOM file includes enough structured information of all the classes within domain library, DOM building processes of extends classes or component referenced classes of the current complex simulation class will be skipped. So through domain library preprocessing, simulating a complex model which reuses a lot of classes from the domain library will become very quickly even at the first time.

The next paragraphs of this paper will expatiate upon this approach in detail.

2 Traditional library loading and class translating processes

2.1 Traditional library loading

Traditional library loading process is a reading and pre-checking process of a series of mo text files according to the directory defined in system register table or MODELICAPATH environment variable. Usually, software based on Modelica loads the basic Modelica Standard Library including Modelica, ModelicaAdditions, MultiBody, etc. The traditional library loading process only performs the lexical and syntactical analyses, and then builds the tree structure of domain classes according to the mo file and directory structure, package and nested classes structure. Because the loading files are text format,

its directory structure is complex and the process needs lexical and syntactical analyses, traditional library loading process is time-consuming. As our statistics, loading all the classes of Modelica2.1 needs more than 5 minutes through using lexical and syntactical analyzing tool ANTLR.

In addition, building tree control of domain library is also a waste-time process, for this process needs generate icons of the loading classes, which correspond to the images of tree nodes. The image of each tree node is created by drawing the icon annotation of the corresponding class. For this process involves creating and drawing geometric entities, as our statistics, building all the tree nodes of Modelica2.1 needs more than 8 seconds. Even if we adapt the method by saving the images to temporary bitmap files, this process still needs about 5 seconds.

2.2 Traditional class translating

As we mentioned above, loading process of domain library only pre-checks the classes but does not construct structured information which are necessary for the main class translating.

Traditional class translating process usually includes three steps: 1) Lexical and syntactical checking. This step usually depends on professional translating tool such as ANTLR. At the same time, this checking process will create abstract syntax tree (AST) of the main class. 2) Semantic resolving based on AST. It is to check types and resolve extends clauses, modifications (including re-declarations), outer or inner matches, connect clauses, and so on. 3) Equation system generating. The semantic resolving prepares necessary information for instantiation which is performed. The purpose of instantiation is mainly to generate equation system (continuous equations and discrete events) for the main class.

Because the main class may include quite a number of components and extends classes, translating it needs checking referenced classes of the components and extends classes, then generating their ASTs recursively. So if the main class includes a great many components or extends which are referenced from domain library, this checking process of the referenced classes could take long time.

3 Document object model (DOM) of Modelica class

As an immediate expression of program language, abstract syntax tree (AST) is adopted abroad to storage

structured information of class. But building AST of Modelica code is a very complicated task, for Modelica language has a complex syntax structure. Moreover, translating operation based on AST is more complex. For example, in Modelica language we can use the keyword “*final*” to constrain element or class being modified further, as the following code:

```
final class Volt
  String quantity = “Voltage”;
  String unit = “Volt”;
  String displayUnit = “V”;
end Volt;
```

For all the class is defined as *final*, its property variables (quantity, unit, displayUnit) are also looked as *final*. Just to say they all can not be modified. The prefix deduction of AST needs traverse all nodes of the class *Volt* to define which elements need be set *final* property. Another more troublesome thing is that we must add nodes to AST or find particular node to modify its properties. So, we put forward and design the so-called ModelicaDOM, i.e. Modelica document object model, which is a type of container structure.

DOM (Document Object Model) is developed from XML (eXtensible Markup Language). DOM is defined formally as [8]: “Document Object Model of a type of document is a platform-independent and language-independent interface. It allows program or script to access or modify the content, structure and style of the document”.

ModelicaDOM is a container in physical structure while stores the tree information of Modelica-based document in logical structure. We design the class ModelicaDOM and implement its construct function and other access functions which are supplied to access the inner data of its objects.

Relative to AST, ModelicaDOM has the following advantages:

- (1) ModelicaDOM is an object-oriented expression style. Its nodes of container could be objects of a class or objects of derived classes of the class.
- (2) Logical structure of the syntax tree with ModelicaDOM is simpler. For Modelica2.1, We can conveniently storage all its grammar information. According to grammar, a regular class can include information of seven types of elements: imports, extends, nested classes, components, equations, algorithms and modification information. So the definition of regular class should comprise container of this elements. Contrasting with AST, the number of classes which need be designed in ModelicaDOM hierarchy decreases about one third.
- (3) Access operation of ModelicaDOM is simpler and more efficient. Finding some element or modifying data of some element is very convenient and quickly. For example, if we want to modify the

data of a component, we can search and get the object pointer of the component class firstly, then modify the corresponding data through interfaces of the class. However in AST, these operations often need traverse the whole tree in order to find and modify the data.

So as a replacer of AST, *ModelicaDOM*, document object model of a defined class includes all information of the class in structured data format. DOM of a class can be built after the checking process of class. And it is the base of generating equation system of the class.

We designed the class hierarchy of ModelicaDOM according to Modelica semantics like fig. 1.

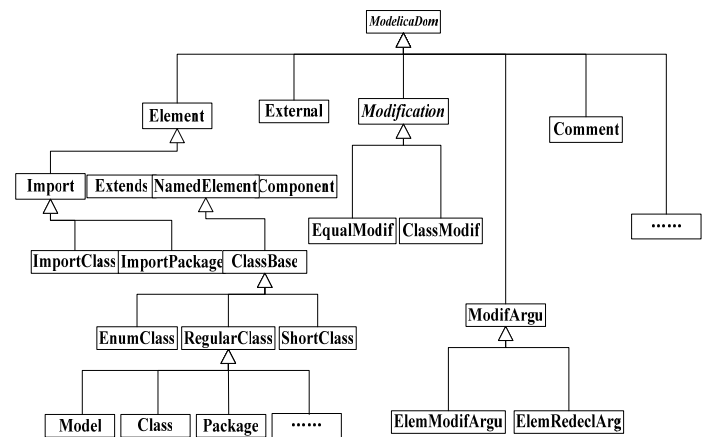


Fig.1 class hierarchy of ModelicaDOM

The *RegularClass* class in the hierarchy is one of the most important classes. *RegularClass* class has the following eight derived classes: Model, Class, Package, Block, Record, Type, Function and Connector classes. Thereinto, the “*RegularClass*” class is a common class which is usually used to define a simulation model. It contains about seven kinds of information: imports, extends, nested classes, components, equations, algorithms and modifications. In MWorks, “*RegularClass*” class is designed as following:

```
class RegularClass :
{
public:
...
//vector pointer of import classes
vector<ImportClass*>* pImpClasses;
//vector pointer of import packages
vector<ImportPackage*>* pImpPackages;
//vector pointer of extends classes
vector<Extends*>* pBases;
//vector pointer of nested classes
vector<ClassBase*>* pNestedClasses;
//vector pointer of components
vector<Component*>* pComponents;
//vector pointer of annotations
vector<Annotation*>* pAnnotations;
```

```

//container pointer of equations
BehaviorContainer* pEquaContainer;
//container pointer of initial equations
BehaviorContainer* pInitEquaContainer;
//container pointer of algorithms
BehaviorContainer* pAlgoContainer;
//container pointer of initial algorithms
BehaviorContainer* pInitAlgoContainer;
//pointer of modifications of this class
ClassModif* pClassModif;
...
};

```

But according to Modelica2.1, there are other classes (enum class and short class) excluding the regular class. So we design the class “Classbase” as the base class of these three classes, which defines all the same properties and same operations of these classes. And Classbase will be the base class of all node objects of the container of ModelicaDOM.

4 Preprocess of domain library in MWorks

When MWorks system starts up, it will execute the OnStartup() function automatically like the following code in C++:

```

//using MFC
void CMainFrame::OnStartup(vector<ClassBase*>
& vClasses;)
{
    /*vClasses is the container of ModelicaDOM, i.e.
    vector of pointer of Classbase*/

    /*getting ModelicaPath, these paths were set by
    user*/
    vector<string> vPaths = GetModelicaLibPaths();
    //foreach path do
    for(int i=0; i<vPath.size(); i++)
    {
        string sPath = vPath[i];
        //search DOM file from work directory
        string sDOMFile = GetDOMFile(sPath);
        //if the Dom file exists
        if(sDOMFile != "")
        {
            /*read the Dom file, construct vector of Classbase
            objects*/
            DOMSerialize(sDOMFile, vClasses, DOM_READ);
        }
    }
}

```

```

/*if DOM file does not exist, execute domain
library preprocessing*/
else {
    /*load mo files in sPath, and build vClasses
    roughly*/
    vClasses = LoadPath(sPath);
    /*rebuild vClasses again, set the pointer value of
    referenced classes*/
    RebuildDOM(vClasses);
    /*generate a DOM file name according to work
    dir*/
    string sDOMFile=GenerateDOMFileName(sPath);
    //save DOM information to sDOMFile
    DOMSerialize(sDOMFile,vClasses,DOM_WRITE);
}
//create icons of domain library tree nodes
...
}

```

From above, *GetModelicaLibPaths()* function gets paths of Modelica library which will be loaded. If there exists MODELICAPATH environment variable, the paths will be extracted from this variable. If not, system will search the registered table of Windows to get paths from the key “Modelica_Lib_Paths” which was set while MWorks installing.

The function *GetDOMFile(sPath)* will get a DOM file according to the const string *sPath* and the work directory of MWorks. For an example, if *sPath* is “C:\MWorks\Library\Modelica” and the work directory is “C:\MWorks\work”, this function will return “C:\MWorks\work\Modelica.DOM”.

The code of creating icons of domain library tree nodes is leaved out. This process is just like the *vClasses* created. When the tree nodes is building according to the *vClasses*, it will search the icon images from a bmps file, if the file does not exists, it will generate an icon from the icon annotation of the class. At the same time saving the icon image to a bmps file. For all icons are saved to one bmps file only if system starts up once, the icons of domain tree nodes will be created very quickly through reading the image from the bmps file, without creating images through constructing geometric entities and drawing them.

From the code above, preprocess of domain library includes three stages: library loading, ModelicaDOM building and serializing. This process is usually performed when MWorks starts up or software is installed. It can also be performed via loading library command. It is performed only once, and results in generating a DOM file.

4.1 Loading of domain library

Like 2.1, MWorks loads all mo files including structured entities and non-structured entities according to the specified library directory through executing the function *LoadPath()*. At the same time, MWorks performs lexical and syntactical analysis for the loaded files.

In addition, MWorks will build the DOM information roughly when checking, that is to say, it build all the nodes of the DOM container and set their main information. For there could exist inner reference relationships among the classes in domain library, the pointer of referenced classes of a class could not be all obtained when the class is constructing. Only after all the classes are constructed, these pointers information can be obtained.

4.2 Rebuilding of DOM

The *LoadPath()* function only created the DOM structure roughly, its detailed data are obtained through *RebuildDOM()* function after loading process.

As mentioned above, DOM of a class includes all structured information of the class. DOM of the domain library contains all structured information of all classes defined in the domain library. In MWorks, they are linked as an object-oriented container structure like the following fig.2 shows.

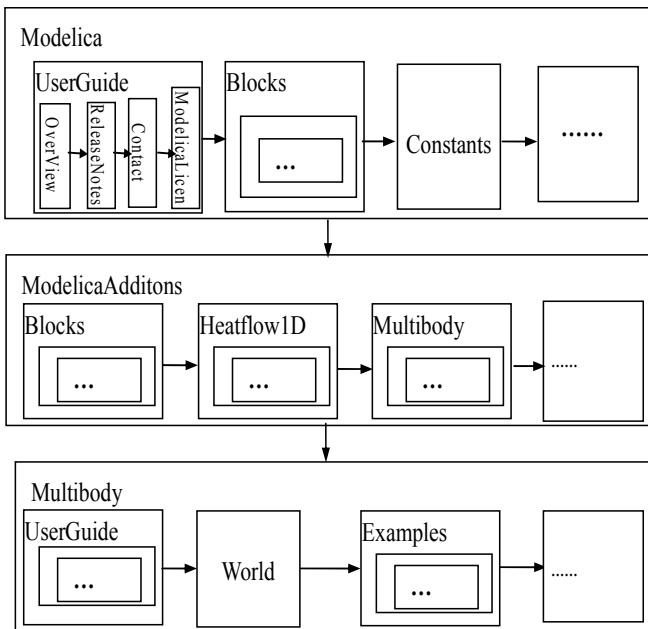


Fig.2 ModelicaDOM of domain library (Modelica2.1)

The container of fig.2 expresses the tree structure of domain classes like fig.3 shows:

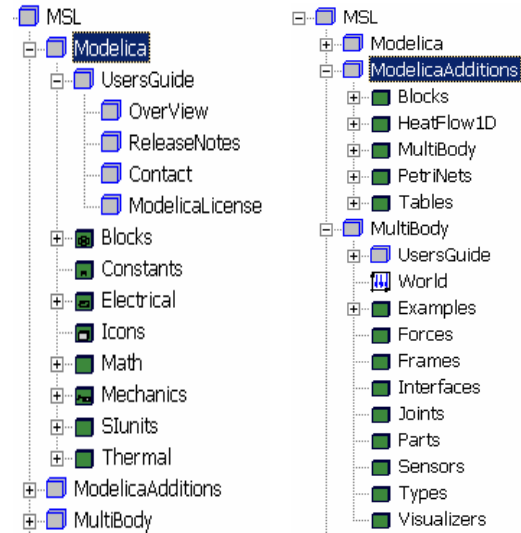


Fig.3 tree structure of domain library (Modelica2.1)

4.3 Serializing of DOM - Saving

Serializing of DOM involves two aspects: saving DOM information into DOM files and loading DOM files to construct DOM data structure. And the former is the main task of preprocessing of domain library. This is realized by the function *DOMSerialize()* whose third parameter *mode* is set *DOM_WRITE*.

After the DOM of one path in domain library has been constructed, they can be serialized into one DOM binary file according to DOM data. This process can be described as the following pseudo-code:

```
function
    DOMSerialize(vector<ClassBase*>vModels,
        string sDOMFileName, int mode =
            DOM_WRITE)
{
    if(mode == DOM_WRITE)
    {
        ofstream fs(sDOMFileName, ios::binary);
        //write the number of saved classes
        fs << vModels->size();
        //write data of each class
        foreach(pModel in vModels) {
            //write the type of class
            fs << pModel->GetType();
            //write data of class in detail
            pModel->WriteToDOMFile(fs);
        }
    }
    else { //DOM_READ
        ...
    }
}
```

The function *WriteToDOMFile()* is virtual function of *ClassBase* class, which is implemented in the same functions of its derived classes *RegularClass*, *ShortClass* and *EnumClass*. From the class hierarchy of ModelicaDOM, this function of *RegularClass* is implemented through its derived classes *Model*, *Class*, *Package*, *Block*, *Record*, *Type* and *Connector*. We give the function code of *RegularClass* as an example:

```
void RegularClass::WriteToDOMFile(ofstream &ofs)
{
    //save import classes
    ofs << pImpClasses->size();
    for(int i=0; i<pImpClasses->size(); i++)
    {
        ofs << (*pImpClasses)[i]->GetFullName();
    }
    .. .. .
    //save extends classes
    ofs << pBases->size();
    for(int i=0; i< pBases->size(); i++)
    {
        (*pBases)[i]->Write(ofs);
    }
    //save nested classes recursively
    ofs << pNestedClasses->size();
    for(int i=0; i<pNestedClasses->size(); i++)
    {
        pNestedClasses[i]->WriteToDOMFile(ofs);
    }
    //save components
    ofs << pComponents->size();
    for(int i=0; i< pComponents->size(); i++)
    {
        (*pComponents)[i]->Write(ofs);
    }
    .. .. .
}
```

5 Library loading, and class translating in MWorks

5.1 Library loading in MWorks

As another aspect of serializing, loading is the reverse operation of saving which is implemented through the function *DOMSerialize()* with the third parameter mode being *DOM_READ*. In MWorks, library loading process can be described as the following pseudo-code:

```
function DOMSerialize(vector<ClassBase*>vModels,
    string sDOMFileName, int mode = DOM_WRITE)
{
```

```
    if(mode == DOM_WRITE)
    {
        .....
    }
    else { //DOM_READ
        ifstream ifs(sDOMFileName, ios::binary);
        //read the number of classes ever saved
        ifs >> num;
        for( i =0; i<num; i++)
        {
            //read type of class
            ifs >> type;
            //Create object of the class
            ClassBase *pModel ;
            if( type == "shortclass")
                pModel = new ShortClass();
            else if(type == "enumclass")
                pModel = new EnumClass();
            else if(type == "class")
                pModel = new Class();
            else if(type == "model")
                pModel = new Model();
            .....
            pModel->ReadFromDOMFile(ifs);
            //append to vModels
            vModels->append(pModel);
        } //end for
    } //end of else
}
```

The function *ReadFromDOMFile()* is executed in the same order with saving like the following code:

```
void RegularClass::ReadFromDOMFile(ifstream &ifs)
{
    //read import classes
    int nImpSize;
    ifs >> nImpSize;
    for(int i=0; i<nImpSize; i++)
    {
        string sImpName;
        ifs >> sImpName;
        ImportClass *pImpClass=
            new ImportClass(sImpName);
        pImpClasses->push_back(pImpClass);
    }
    .. .. .
    //read extends classes
    int nExtendsSize;
    ifs >> nExtendsSize;
    for(int i=0; i< nExtendsSize; i++)
```

```

{
  Extends *pExtends = new Extends();
  pExtends->Read(ifs);
  pBases->push_back(pExtends);
}
//read nested classes recursively
int nNestedClassSize;
ifs >> nNestedClassSize;
for(int i=0; i<nNestedClassSize; i++)
{
  ClassBase *pClass = new ClassBase();
  pClass->ReadFromDOMFile(ifs);
  pNestedClasses->push_back(pClass);
}
//read components
int nComSize;
ifs >> nComSize;
for(int i=0; i<nComSize; i++)
{
  Component *pComp = new Component();
  pComp->Read(ifs);
  pComponents->push_back(pComp);
}
}
.....
}

```

From these pseudo-codes we can see, MWorks lookups the DOM file which contains the structured information of domain library from the installed directory first. Only if this process fails (DOM files does not exist), it will load domain library and execute preprocessing. If DOM file exists, MWorks will perform serializing process: loading but not parsing the DOM file, then rebuilding the DOM structured information of domain library in the memory.

5.2 Class translating process in MWorks

In MWorks, translating process of main class can be described as the following pseudo-code:

```

void RegularClass::Translating(EquationSystem & es)
{
  //generate equations of this class
  //container pointer of equations
  pEquaContainer->GenerateEquSys(es);
  //container pointer of initial equations
  pInitEquaContainer->GenerateEquSys(es);
  //container pointer of algorithms
  pAlgoContainer->GenerateEquSys(es);
  //container pointer of initial algorithms
  pInitAlgoContainer->GenerateEquSys(es);
  //pointer of modifications of this class
  pClassModif->GenerateEquSys(es);

  //generate equations of extends classes recursively
  for(int i=0; i<pBases->size(); i++)
  {

```

```

    string sClassName=(*pBases)[i]->GetFullName();
    ClassBase *pClass = GetClass(sClassName);
    if (pClass == null) { //does not build
      pClass = BuildDOM(sClassName);
    }
    pClass->Translating();
  }
  //neglect nested classes
  //generate equations of components
  for(int i=0; i<pComponents->size(); i++)
  {
    Component *pComp = (*pComponents)[i];
    string sCompClass = pComp->GetClassName();
    ClassBase *pClass = GetClass(sCompClass);
    if(pClass == null) {
      pClass = BuildDOM(sCompClass);
    }
    pClass->Translating();
    pComp->GenerateEquSys(es);
  }
  .....
}

```

The function *BuildDOM()* involves getting the path from full-name of the class, loading it with *LoadPath()* function and rebuilding DOM with *Rebuilding()* function.

As we know, first step of translating the main class is to build DOM of the class. And building the DOM of main model will lookup or construct DOM of its extends classes and component classes which usually referenced from the domain library. Because the domain library has been loaded when system starts up and DOM of the library has been constructed, this process only needs looking-up of DOM referenced classes. And this process only needs getting pointers to the DOM of domain library.

5.3 Generating Fulltext of Class from DOM

As a Modelica-based platform, MWorks has five views which express five aspects of the class respectively: *Fultext* view, *Icon* view, *Diagram* view, *HTML* view and *Simulation* view. The *Icon* view and the *HTML* view express icon and HTML information of the class and their information is obtained from the *pAnnotations* variable of the class and the base classes recursively. The *Diagram* view expresses diagram entities, components blocks and connect among components of the class, so its information comes from the *pAnnotation* variable of the main class and its base classes recursively, *pComponents* and connect part of *pEquaContainer* of the main class. The *Fultext* view is mo text expression of the main class. Because we have adopted domain library preprocessing, the mo text information of the classes

in the domain library was lost. So if we want to show the full-text of these classes, the mo text of each class must be reconstructed from the DOM structure. Generating process of the fulltext of a class is similar as the saving process from DOM information to DOM file. And the distinguish from saving is that it prints the DOM information to its original mo text string while saving process saves all the property data of the class to a binary file. As an example, the import class information *pImpClasses* converts to mo text like the following code:

```
void RegularClass::ToString(string & sText)
{
    //print pImpClass to string
    for(int i=0; i< pImpClasses->size(); i++)
    {
        sText += "import ";
        sText += pImpClasses[i].GetFullName();
        sText += ";\n";
    }
    .....
}
```

6 Conclusions

We have implemented a modern IDE for modeling and simulation of multi-domain physical systems based on Modelica - MWorks. It has a larruping feature of domain library preprocessing, which improves loading speed of library and translating speed of the simulation model rapidly. But from other point of view, domain library preprocess needs much larger memory space. In other words, we sacrifice space complexity to gain time saving. The following is time and space consuming contrast:

Time contrasting: for Modelica2.1, using a computer with its CPU being 2.4Gmps, memory being 512M, loading mo text file needs more then 300s, while if we adopt preprocessing, loading the DOM file only needs 6s.

Space contrasting: with the same computer, loading without building DOM structure only needs 11M, while through preprocessing the occupied memory will reach to or exceed 100M.

In addition, for the DOM file saves the structured information but not in mo text format, classes in the library can be encrypted easily through the pre-processing.

7 Acknowledgement

The paper was supported by project of Chinese National Science Foundation Committee (60574053), Chinese 863 high tech project (2003AA001031) and Chinese 973 project (2003CB716207).

References

- [1] Modelica, <http://www.modelica.org/>, 2006.
- [2] Dynasim. Dymola, <http://www.dynasim.se/>, 2006
- [3] MathCore. MathModelica, <http://www.mathcore.se/>, 2006
- [4] OpenModelica Fritzson, P., et al. *The Open Source Modelica Project*. in *Proceedings of The 2th International Modelica Conference*, 18-19 March, 2002. Munich, Germany.
- [5] C. Nytsch-Geusen et. al., Fraunhofer Institutes, Germany: MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics in *Proceedings of The 4th International Modelica Conference*, 7-8 March, 2005. Hamburg-Harburg, Germany.
- [6] M. Najafi, S. Furic, R. Nikoukhah, Imagine; INRIA-Rocquencourt, France: SCICOS: a general purpose modeling and simulation environment in *Proceedings of The 4th International Modelica Conference*, 7-8 March, 2005. Hamburg-Harburg, Germany.
- [7] Peter Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, 2004
- [8] Document Object Model (DOM). <http://www.w3.org/DOM/>, 2006