



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 1

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] VTT, 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. Obratsov ^[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. Yebrá ^[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. Claub ^[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 Universitaet 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 Simulation539
	GAPILib - A Modelica Library for Model Parameter Identification
	Using Genetic Algorithms335
	Object Oriented Modelling of DISS Solar Thermal Power Plant.....449
Doshi, P.	Parameterisation of Modelica Models on PC and Real Time Platforms267
Ebner, A.	Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation261
Eborn, J.	Calibration of Static Models using Dymola615
	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 Modelica11
Elmqvist, H.	Automatic Fixed-point Code Generation for Modelica using Dymola.....621
	Calibration of Static Models using Dymola615
Enge, O.	Quasi-stationary AC Analysis Using Phasor Description With Modelica579
Ernst, T.	Advanced Modeling and Simulation Techniques in MOSILAB:
	A System Development Case Study63
Eschenbach, M.	Vehicle Model for Transient Simulation of a Waste-Heat-Utilisation-Unit Containing Extended
	PowerTrain and Fluid Library Components405
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 Performance319

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
Kittirungsi, 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
Obraztsov, 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 1a

Thermodynamic Systems for Power Plant Applications 1

Fast Start-up of a Combined-Cycle Power Plant: a Simulation Study with Modelica

Francesco Casella¹, Francesco Pretolani²

¹Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano ITALY

²CESI S.p.A., Via Rubattino, 54 – 20134 Milano ITALY

e-mail: casella@elet.polimi.it, pretolani@cesi.it

Abstract

The paper deals with the modelling and simulation of fast start-up transients of a combined-cycle power plant. The study is aimed at reducing the start-up time while keeping the life-time consumption of the more critically stressed components under control. The structure of the model, based on the ThermoPower library, and the main modelling assumptions are illustrated. Selected simulation results are included and discussed.¹

1 Introduction

The on-going process of deregulation on the electrical power grids throughout Europe demands for more aggressive operation policies for existing and future power plants. Faster start-up and load change transients can be beneficial to remain competitive on an increasingly open power market. In this context, the present work is aimed at understanding how to improve the current start-up procedures for the typical combined-cycle power plant installed on the Italian grid.

For the purposes of the present study, the plant model must have the following features:

- be able to represent the whole start-up procedure, including boiler start-up, turbine start-up, and load pick-up;
- include a model of thermal stresses in critical components, which pose a lower bound to the start-up time;

- include a simplified model of the plant control system;
- neglect phenomena and components which are not critical for the start-up phase, in order to keep the model complexity at a reasonable level.

The plant model, based on the ThermoPower library [1]-[3], has been parametrised with design and operating data from a typical unit, and validated by replicating a real start-up transient, as recorded by the plant DCS. The model has then been used to test faster start-up manoeuvres, with the objective of either reducing the plant life-time consumption at equal start-up times, or reducing the start-up time at the same level of plant life-time consumption. This study has been carried out by trial-and-error, but the long term goal is to couple the model (or a suitably simplified version of it) to state-of-the-art optimisation software, to compute the optimal transients.

2 The plant model

The plant under investigation is composed by a 250 MWe gas turbine unit (GT), coupled to a heat recovery steam generator (HRSG) with 3 levels of pressure, driving a 130 MWe steam turbine (ST) group. The limiting factors to a reduction of the start-up time are:

- the maximum load change rate of the gas turbine;
- the thermal stress in thick components (in particular, the steam turbine shafts);
- the ability of the control system to keep their controlled variables within the allowable limits.

¹ This work was supported by MAP (Italian Ministry for Productive Activities) in the framework of the Public Interest Energy Research Project “Ricerca di Sistema” (MAP decree February 28, 2003).

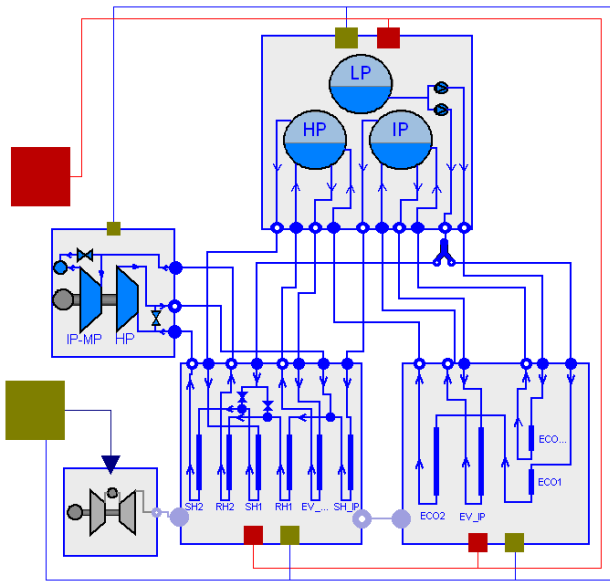


Fig. 1: The plant model at the system level.

At the system level, a detailed representation of all the parts of the plant working with low-temperature fluids is not required, since they are not critical, as far as their control and their thermal stresses are concerned. Therefore, the low-pressure part of the HRSG, the condenser and the feed-water system will be represented in an extremely simplified way. The plant model is then obtained (Fig. 1) by connecting the models of the GT unit, HRSG unit (divided into three parts for convenience), and ST unit via thermo-hydraulic connectors. Sensor and actuator signals are collected from/to each unit by means of expandable connectors.

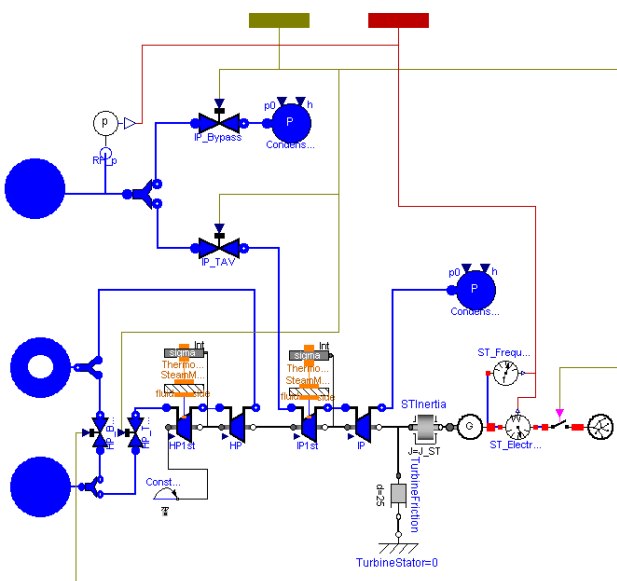


Fig. 2: The steam turbine unit model.

It is well-known that the HRSG start-up (several hours) is much slower than the GT start-up time (around 20 minutes). It is then possible to describe the GT unit in a highly idealised fashion, i.e. as an ideal source of hot flue gases, whose temperature and flow rate is prescribed as a function of the load level; the maximum load change rate is given by the unit specification, and is not a subject of the present study.

The steam turbine unit model (Fig. 2) is instead rather detailed, in order to correctly describe the various phases of the start-up transient. The high pressure turbine (HP) and intermediate-plus-low pressure turbine (IP) are modelled, as well as the turbine bypass circuits; the contribution of the low-pressure steam generator is instead neglected.

The most critical part of the plant, as far as the thermal stresses are concerned, is the turbine shaft in contact with the highest temperature steam, i.e. downstream of the first (impulse) stage of both turbines, which is then represented separately, and connected to a thermal stress model of the shaft section. The stress model contains a thermal model, based on Fourier's equation discretised by finite differences, to represent the radial distribution of the temperature; the thermal stress on the outer surface (which is the most heavily stressed part) is then computed as a function of the difference between the surface temperature and the mean temperature. The generator inertia and a simplified model of the connection to the grid complete the unit; a small torque is added in order to avoid the stopping of the steam turbine, which would lead to various model singularities.

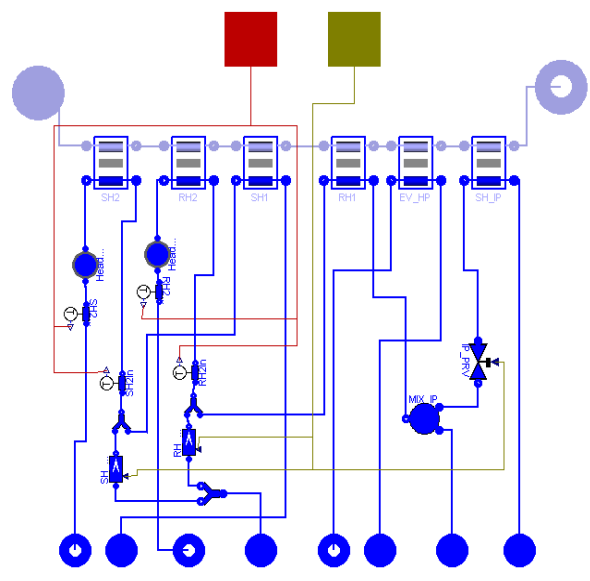


Fig. 3a: Heat exchanger units in the HRSG.

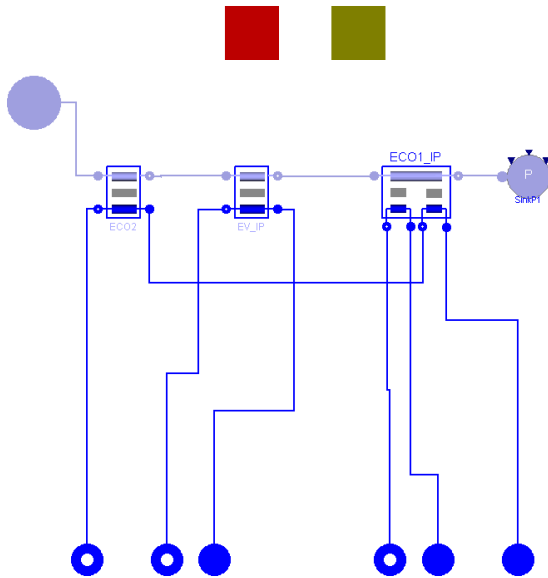


Fig. 3b: Heat exchanger units in the HRSG.

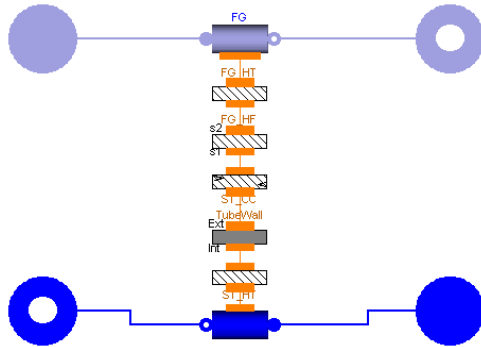


Fig. 4: A single, generic heat exchanger model.

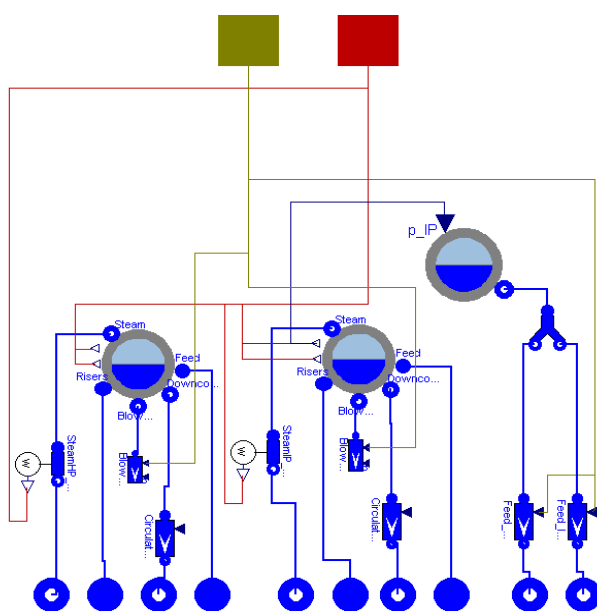


Fig. 5: The drum unit.

The heat exchangers along the flue gas path, i.e. the economisers, evaporators and superheaters, as well as the IP mixer and attempters, are contained in two units for convenience (Fig. 3a-b). The structure of the each heat exchanger (Fig. 4) includes finite-volume models of the flue gas and water/steam side, as well as of the fluid-wall heat transfer, and of the wall thermal inertia. Since there is no draft control in the flue gas path, the associated dynamics is negligible; therefore, to reduce the number of states of the model, the flue gas side model is quasi-static. Note that the fluid side model is replaceable: a `Flow1D` model is used for the economisers and superheaters, while a `Flow1D2ph` model is used to describe the 2-phase flow in the evaporators.

The plant model is completed by the models of the boiler drums (Fig. 5). Since we are not interested in the high-frequency pressure dynamics, the high-pressure (HP) and intermediate pressure (IP) drums are based on mass and energy balances assuming thermal equilibrium between the two phases. The low-pressure part of the HRSG is neglected, so that an idealised model of the low-pressure drum is only needed as a boundary condition for the IP and HP circuits, i.e. to connect the inlets of the corresponding feed-water pumps (Fig. 5, on the far right). The LP drum pressure (and thus temperature) is determined as a function of the IP drum pressure, tuned from operational data.

3 Control system model

Given the type of plant, and the modelling assumptions, the control system can be hierarchically split into two levels.

3.1 Low level controllers

The lower level is quite straightforward, and is not the subject of the optimization. It contains five independent PI/PID loops, controlling:

- the HP and IP drum levels, using the corresponding feed-water flows;
- the HP steam pressure, using the HP turbine bypass valve;
- the IP steam pressure, using an intermediate valve at the outlet of the IP superheaters, before the mixing with the HP turbine exhaust;
- the reheater steam pressure, using the IP turbine bypass valve.

Note that the HP pressure controller is only active during the initial phase of the plant start-up, when

the turbine admission valve (TAV) is closed, and the steam is dumped to the condenser. Once the steam turbine generator has been synchronised with the grid, the TAV is opened, the pressure diminishes, and the pressure controller reacts, eventually closing the bypass valve completely. The (continuous-time) PID controller model must then be able to operate correctly under saturation, providing suitable anti-windup action.

3.2 Supervisory control

The supervisory control level determines the actual start-up transient by acting on the following variables:

- GT load request;
- TAV opening (both HP and IP) *or* turbine speed set point;
- pressure controllers set points (HP and IP);
- generator-grid breaker;
- drum blow-down flow rates.

During the start-up transient, all of these variables are operated in an open-loop fashion, according to a pre-determined schedule which is the subject of the optimisation. The only exception is the TAV opening, which is determined in closed loop by a speed controller during the turbine start-up transient phase: in that case, a PI controller drives the TAV, and the scheduling variable is the speed set-point. It is therefore necessary that the PI controller provides a tracking mode as well, to manage the transitions between the off-duty, start-up and connected modes of operation of the turbo-generator in a correct fashion.

4 Model parametrisation and validation of the reference transient

The physical parameters of the model (dimensional data and operating points) have been set to match those of a real unit. Some data are known directly (e.g. number and dimensions of the tubes in the heat exchangers), other (e.g. the heat transfer coefficients or the valve and turbine flow coefficients) are computed from operating point design data.

The low-level controllers have been tuned in order to provide satisfactory performance (fast enough response with no significant oscillations and control overshoot).

The direct initialisation of the plant model in the shut-off state is numerically hard, due to the presence of low or zero flow rates and to the lack of knowledge of good guesses for the initial values.

Therefore, the model has been initialised near the full-load steady state by setting the `start` attributes of the state variables (pressures, temperatures, flow rates, turbine speed, controller states), then brought to the full load steady state by running a stabilisation transient. The use of variable step-size integration algorithms allow to perform this task in a reasonable time (less than 10 seconds, CPU time). The steady-state values of the variables of interest (pressures, temperatures, flow rates, powers) are correct by construction, as the model has been parametrised using those same values. Incidentally, an attempt was performed to get the steady state directly by using initial equations, but the non-linear solver failed to converge, probably due to bad selection of the start values for some iteration variables.

A plant shut-down transient was then performed, bringing the model to a state corresponding to the warm start-up of the plant:

- steam turbines with no steam flow and (almost) at standstill.
- pressures around 1 bar in both the HP and IP circuit.
- GT “almost” shut down (a small flow rate of warm flue gas is kept to avoid singularities in the flue gas side model).

The temperature distribution of the turbine shafts was then reinitialized to the desired initial value (corresponding to 180 °C). This constitutes the initial state for the start-up transient simulation.

A reference simulation was then performed to replicate the recording of an actual start-up transient, which was replicated with acceptable fidelity, as far as the measured variables are concerned. Note that the study is not targeted to a specific plant, but rather to a whole class of similar plants, so that a high accuracy is actually not needed. Some results are reported here to give an idea of the degree of complexity of the transient.

Fig. 6 reports the net electrical power outputs of the GT, steam turbine (ST) and total. During the first 5000 s, the GT is running idle, so that there is no net electrical power output, but a certain flow of hot exhaust gases is already available to start up the steam generator. The steam turbine is then started and synchronised, and at time $t = 13500$ s the steam turbine starts picking up steam, while the GT load is increased up to the maximum.

Fig. 7 shows the pressure in the HP and IP drums. The steam generator start-up is split into two phases, where both pressures are controlled; during the load pick-up phase, instead, the HP circuit operates in sliding pressure mode, to avoid reducing the turbine efficiency due to throttling.

Fig. 8 shows the steam production rates (HP in blue, IP in green), as well as the flow rate through the HP turbine (in red). Until $t = 9600$ s all the steam is dumped to the condenser; subsequently, a small amount is used to accelerate the turbine (see the turbine speed, Fig. 9), and only after the load pick-up has started the TAV are fully opened, sending all the steam into the turbine. Fig. 10 shows the interplay between the TAV and the bypass valves; the former are gradually opened during the transient, while the latter are closed by the pressure controllers.

All the transients shown so far (except the last) correspond to actual measurements taken on the plant.

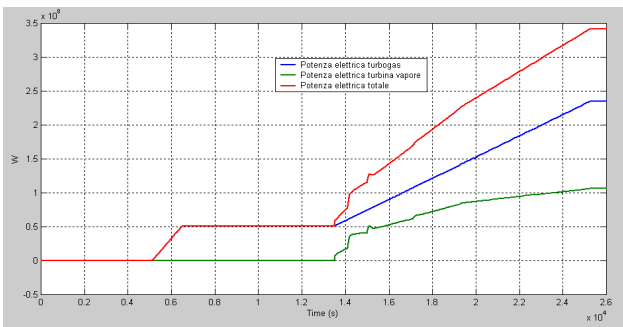


Fig. 6: Net electrical power outputs.

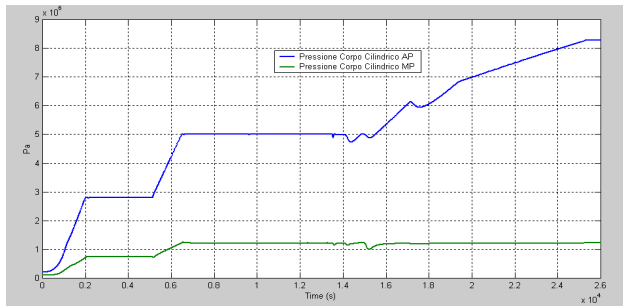


Fig. 7: HP and IP drum pressures

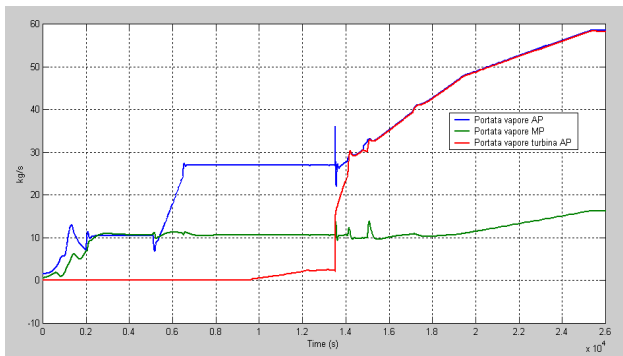


Fig. 8: Steam flow rates.

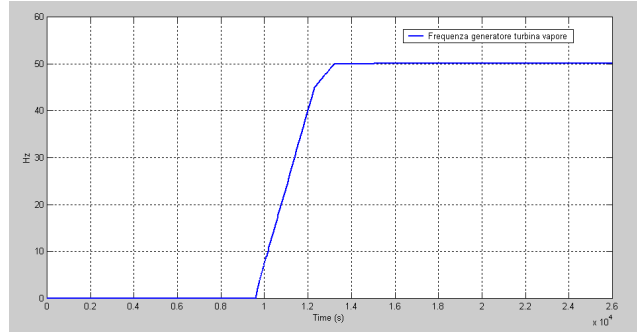


Fig. 9: Turbine speed.

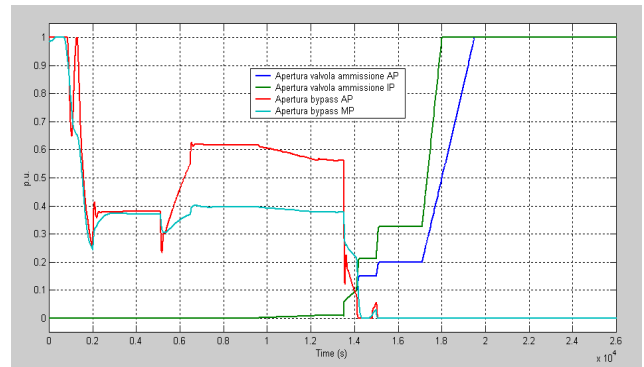


Fig. 10: Valve openings.

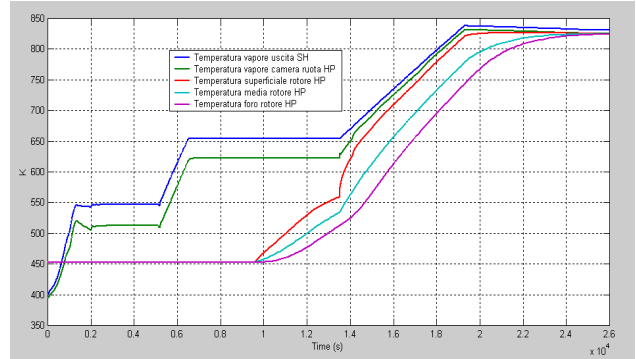


Fig. 11: HP turbine steam and rotor temperatures

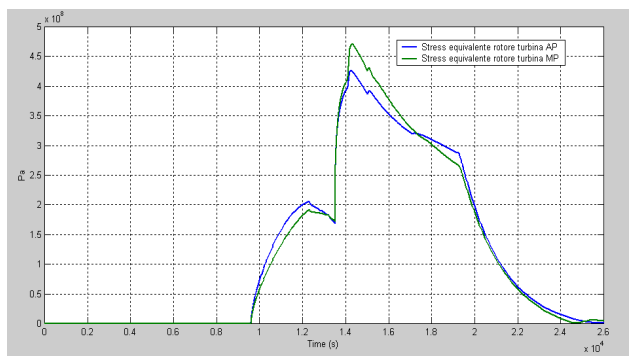


Fig. 12: HP and IP turbine rotor thermal stress.

The most interesting part of the simulation concerns the temperature distribution in the first section of the steam turbine shafts, and the corresponding thermal stress, which results in component fatigue and thus limits its useful lifetime. Fig. 11 shows the temperatures for the HP turbine: the three lower curves represent the internal, mean and external rotor temperatures, while the two upper curves represent the temperature of the superheated steam at the turbine inlet, and the (slightly lower) temperature of the steam downstream the nozzles of the first stage, which is where the steam comes into contact with the rotor. During the turbine start-up, the steam flow is very low, and so is the heat transfer coefficient; therefore, the rotor temperatures increase slowly; when the TAV are opened more aggressively, the external temperature gets much closer to the steam temperature. Note that, at time $t = 19200$ s, the steam temperature tops at 830 K; this is due to the GT exhaust temperature control, which keeps the hot flue gases at constant temperature for loads higher than 60%, by acting on the inlet guide vanes. At the end of the transient (steady state), all temperatures are equal, since there is no steady-state heat flow.

The corresponding thermal stresses at the rotor surface (for both HP and IP turbines) are shown in fig. 12. The peak values, which actually determine the lifetime consumption over the start-up cycle, are around 450 MPa, which is consistent with typical values estimated on the real plant during warm start-up transients. Note that this peak is reached at the beginning of the load pick-up phase.

5 Improving the start-up transient

The analysis of the reference transient shows that the current start-up procedure is quite conservative. There are a number of intermediate stops, which are not needed from a physical point of view, and have probably been provided to allow for ample margin against unexpected problems when starting up the plant. The current practice was in fact conceived when the plant was run in a vertically integrated context, which placed more emphasis on safety and availability rather than on efficiency and economy of operation. If those stops are completely eliminated, the corresponding transient can be run without any problems for the control system, resulting in a reduction of the start-up time from 25300 to 19200 s, and in fuel savings corresponding to the production of 208 MWh at full load (i.e. maximum efficiency). The same peak levels of stress are obtained. The details are omitted for the sake of brevity.

The “theoretical” minimum start-up time was then sought in two complementary ways:

- minimising the start-up time subject to the constraint of getting the same stress peak (and thus lifetime consumption) of the reference transient;
- reducing the stress peak (and thus increasing the lifetime consumption), without increasing the start-up time with respect to the reference transient.

To reach the first goal, it is essential to note that the stress peak is basically due to the thermal shock at the beginning of the start-up phase. Once this peak has been hit, the lifetime consumption is the same regardless how fast the stress goes back to zero. Therefore, the GT load pick-up rate has been increased from 1 MW/min to 1.5 MW/min, in order to keep the stress transient flat (see fig. 15). Furthermore, once the GT load has reached 60%, the flue gas temperature does not increase further, so that also the superheated steam temperature will not increase substantially (it will actually decrease a little bit, as the steam flow rate increases), and the stress will decrease no matter how fast the load goes up.

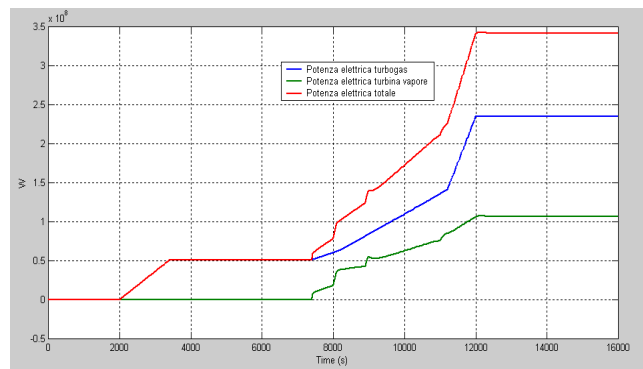


Fig. 13: Net electrical power outputs, fast start-up

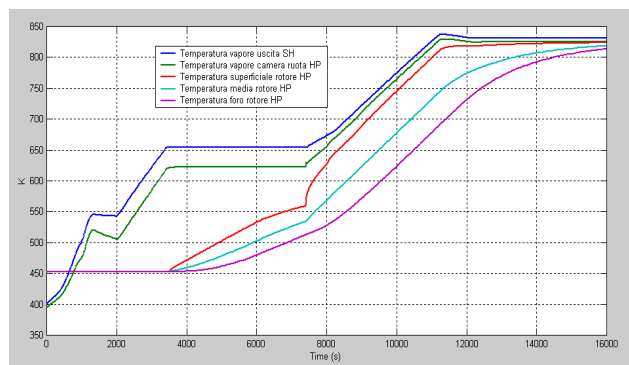


Fig. 14: HP turbine rotor and steam temperatures, fast start-up

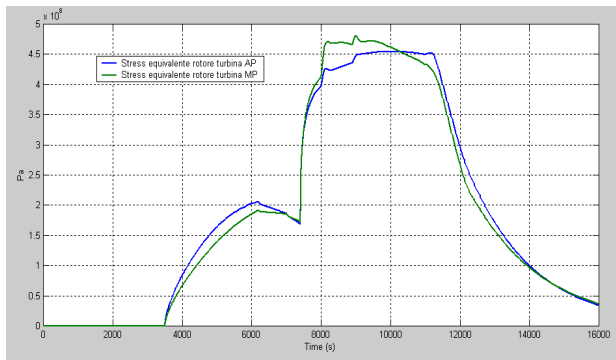


Fig. 15: HP and IP turbine rotor stress, fast start-up

Fig. 13 represents the net power outputs, while Fig. 14 and 15 represent the corresponding temperature profiles on the HP turbine rotor, and the corresponding thermal stresses on both turbines, respectively. It is possible to compare these figures with Fig. 6, 11, and 12. The peak stress is still around 450 MPa. The transients of all the other variables are similar to those already shown for the reference case, showing no particular problem as to the plant control.

Compared with the reference transient, the start-up time is reduced from 25300 s to 12500 s, and the fuel savings correspond now to 242 MWh at full plant load.

To reach the second goal (i.e. reduce the stress peak), it is necessary to reduce the thermal shock at the beginning of the steam turbine load pick-up. This can be obtained by allowing for a longer soak time for the turbine, i.e. once the turbine has reached full speed, it is kept running at no load so that the steam flow can heat up the rotor a little bit more. The turbine start-up is therefore initiated earlier than in the reference transient, and the turbine is kept idling for 3900 s. The load pick-up phase is then started, and the rate of change is adjusted to obtain a flat stress curve. Once the 60% level has been reached, the rate of change is increased to 7 MW/min, as in the previous case.

The resulting temperature and stress plots are shown in Fig. 16 and 17. In this case, two stress peaks are obtained (each one causing fatigue and thus lifetime consumption); however, the first peak value, around 200MPa, is only slightly higher than the limit of elastic behaviour (170 MPa), and thus correspond to a very low additional lifetime consumption, while the second, around 320 MPa, is well below the previous value of 450 MPa. The start-up time is reduced from 25300 s to 17500 s, while the fuel savings correspond to 114 MW/h at full plant load.

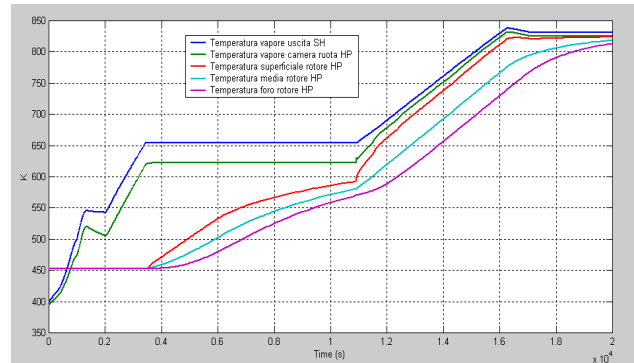


Fig. 16: HP steam and rotor temperatures, soaking.

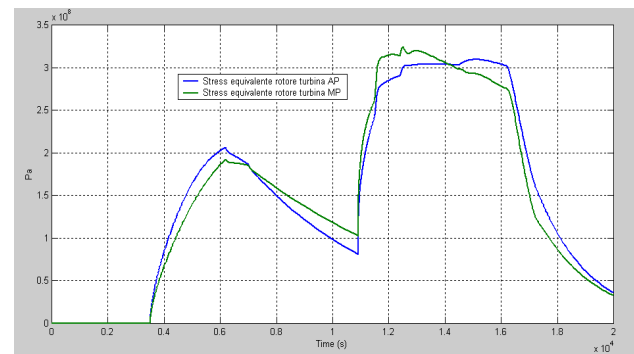


Fig. 17: HP and IP turbine rotor stress, soaking.

6 Conclusions and future work

The optimisation of the start-up procedure for a combined-cycle power plant has been studied, by means of a system simulator. The plant model was developed in Modelica using components from the ThermoPower library; the low-level control system model is based on continuous time PID controllers with anti-windup and tracking features. Compared to traditional simulation environments, it was relatively easy to customise the degree of detail of the model, both by writing extremely simplified component models where possible, and by developing ad-hoc models for the estimation of the thermal stresses in the steam turbines shafts. The final model has around 140 states and several thousands algebraic variables.

The simulation study was conducted using the Dymola [4] simulation tool, which allowed to compute the whole simulation transient in times around 400 s on a Pentium 3 GHz CPU.

The study is a first step towards the realisation of more simplified models, to be validated against the reference one, which will be employed together with optimisation software to automatically compute the

optimal transients. A further step could then be the design of a closed-loop model-based control system, capable of attaining similar performance in real time and in the presence of uncertainty.

7 References

- [1] F. Casella, A. Leva, “Modelling of Thermo-Hydraulic Power Generation Processes using Modelica”, *Mathematical and Computer Modelling of Dynamical Systems*, v. 12, n. 1, pp. 19-33, 2006.
- [2] F. Casella, A. Leva, “Modelica Open Library for Power Plant Simulation: Design and Experimental Validation”, *Proc. 3rd International Modelica Conference*, Linköping, Sweden, Nov 2003, pp. 41-50. URL: http://www.modelica.org/events/Conference2003/papers/h08_Leva.pdf
- [3] ThermoPower library home page, URL: <http://www.elet.polimi.it/upload/casella/thermopower/>
- [4] Dynasim AB, Dymola v. 5.3.

Modelling of a water/steam cycle of the combined cycle power plant “Rio Bravo 2” with Modelica

Baligh El Hefni

Daniel Bouskela

EDF R&D
6 quai Watier
F-78401 Chatou Cedex
France

baligh.el-hefni@edf.fr

daniel.bouskela@edf.fr

Abstract

In order to improve the performance of its simulation tools while reducing their cost, EDF is studying the interest and feasibility to replace LEDA, a tool developed and maintained by EDF for the modelling and simulation of the normal or incidental operation of nuclear and conventional thermal plants, by off-the-shelf available tools.

The combined cycle power plant “Rio Bravo 2” covers the important case of static studies. To test the capabilities of Modelica based tools to meet EDF needs for the modelling and simulation of such complex energy processes, the LEDA model of Rio Bravo 2 was translated from Fortran into Modelica and simulated using the commercial tool Dymola.

A library of fully static 0D thermalhydraulics component models was built. Each component is the complete translation of a LEDA model.

The full model was built by connecting the component models in a technological way, so that its topology reflects the functional schema of the plant.

A preliminary calibration of the model was made against measurement data obtained from on-site sensors using inverse calculations. The model was then able to compute precisely the distribution of the steam/water mass flow rates, pressure and temperature across the network, the exchangers thermal power, and the performance parameters of all the equipments. It converges very quickly, provided that the iteration variables are properly fed in by the user. The results are remarkably close to the LEDA reference.

Keywords : Combined Cycle Power Plant, Steady State Modelling, Inverse Problems

1. Introduction

Modelling and simulation play a key role in the design phase and performance optimization of complex energy processes.

Rio Bravo is a combined cycle plant designed, built, commissioned and operated by EDF. A model of the plant was built at the system level in order to verify and validate by simulation the energy and fuel consumption overall performance of the plant.

The modelling and simulation of the plant was originally carried out with LEDA. LEDA is a tool developed and maintained by EDF since 1982 for the modelling and simulation of the normal or incidental operation of nuclear and conventional thermal plants. LEDA models are used by researchers and engineers to improve their knowledge of existing or future types of power plants, verify the design accuracy and understand important transients.

In order to improve the performance of its simulation tools while reducing their cost, EDF (SEPTEN and EDF R&D) is studying the interest and feasibility to replace LEDA by off-the-shelf available tools.

Modelica based tools are considered as good candidates, because :

- Modelica is a multi-domain language which seems well designed for physical system modelling and simulation.
- Modelica is a declarative language, that captures the model equations in a way that is very close to their original mathematical form.
- Modelica allows to express inverse problems, which is a very important feature for computing operation points, which are defined by their observable outputs, and for system

sizing, to compute parameterised characteristics.

Besides these technical benefits, it is likely that using a common free non proprietary language will foster partnerships around joint R&D or engineering projects, thus giving the opportunities to share development costs between the participants.

Several important benchmark cases have been chosen, which cover the variety of modelling and simulation studies made at EDF [1].

The objective of this work is to evaluate the feasibility and efficiency of using Modelica based tools to perform steady state studies of power plants.

The Rio Bravo combined cycle plant has been chosen as a representative test case of the complexity of this type of study, aimed at verifying the design of the plant for a fixed set of operation points (nominal power output, 50 % of nominal power output, ...).

The modelling and simulation were carried out with the commercial tool Dymola, as it is the most advanced Modelica based tool to this date.

2. The LEDA Solver

LEDA is a tool that was originally designed for the modelling and simulation of power plants. To that end, it allows the user to develop numerical components of the different parts of the plant, and assemble them to build the full model of the plant. Thus LEDA is a model library based tool. The component models are written in Fortran. They represent the various equipments of the plant (pumps, heat exchangers, ...), and may be re-used across different projects.

Since LEDA has already been used by EDF over two decades, it offers the best available physical descriptions, for each improvement - correlations obtained from experimental results or 3D computations - is capitalized into the library.

As the initial state of the simulation is in general defined by the observable outputs of the system (e.g. the nominal power output, the pressure inside the boiler, etc.), it is necessary to solve an inverse problem to compute the initial state. Moreover, it is necessary to start the simulation from a stationary (or steady) state in order to avoid the numerical difficulties which arise when the system is started out of equilibrium (oscillations, stiffness, ...).

That is why LEDA has the ability to start the simulation from a stationary state, and compute this initial state by solving an inverse problem (it is in fact a requirement from the tool to start from a steady

state). It is also possible to feed in the analytical Jacobian of the system to improve the speed and the accuracy of the simulation. LEDA uses a trapezoidal implicit fixed step integrator.

The inverse problem is entered into the tool by setting the output variables to their known measured values, and freeing (i.e. setting as unknowns) the parameters, inputs and internal states to be computed.

Figure 1 below shows the structure of a LEDA component model.

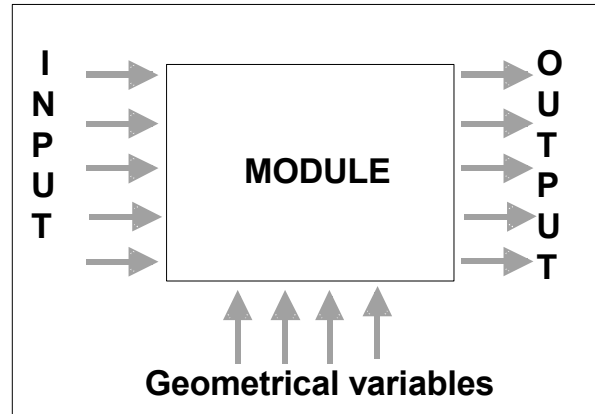


Figure 1 – Structure of a LEDA component model

The component model (also called module) is a series of Fortran files that contain the physical equations and the correlations of the modelled component. It is oriented, i.e. has computationally causal links. Geometrical variables are constant parameters (they would be referred with the Constant Modelica keyword in Modelica models). Parameters are considered as inputs, in order to be able to compute them by inverse calculation. More generally, all variables are either inputs or outputs, which means that the connections between the component models have to be carefully set in order to solve the simulation problem: outputs are always computed from inputs, so the user has to figure out the computational causalities of the problem at hand while building the model.

LEDA assembles the components in a way to produce a global matrix of the system. So the system is solved in an implicit way, not in a sequential way, in particular to solve algebraic loops and perform inverse calculations.

Application fields

- Nuclear power plants.
- Thermal fossil fuel fired power plants (pulverized coal, fluidized bed, ...).
- Combined heat and power plants.

- Waste to energy.

Utilization fields

- Operation and maintenance.
- Design and analysis.
- Innovate technology survey.

3. General presentation of a combined cycle power plant

The combined cycle includes a turbine cycle (gas turbine, which uses natural gas as fuel) and a steam cycle.

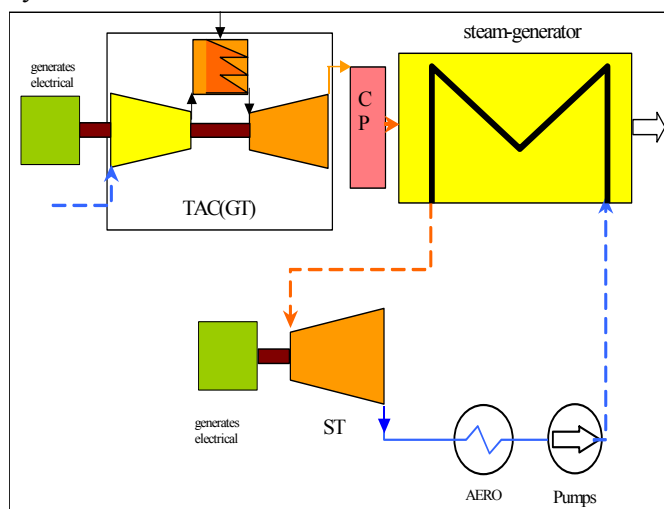


Figure 2 - Diagram of a Combined Cycle Power Plant

In the turbine cycle, air is compressed to the operating pressure of the gas turbine and heated in a combustion chamber (natural gas is burned in), followed by an expansion of the exit flue gas in a gas turbine, which in turn generates electrical energy in a turbo-generator.

The hot effluent of the gas turbine generates steam in the steam-generator of the steam cycle, which expands in the steam turbine and generates additional electrical energy. The steam cycle includes : the steam turbine, a boiler (with several tubular exchangers that transmits the heat of the exhaust gas to the water), 3 evaporating loops (low, medium and high pressure), an aero-condenser and several pumps.

4. The Rio Bravo component model library

As it has already been mentioned, Rio Bravo is a fully static model. Thus, a library of fully static 0D

thermohydraulics component models was built. Each component is the complete translation of a LEDA module. The library now contains models of a multifunctional heater (economizer, superheater, evaporator), a turbine, pumps, a separating balloon, valves, an aero-condenser and pipes.

The model equations take into account the non-linear and the state-of-the-art physical behaviour of each phenomenon of interest. In particular :

- The multifunctional heater component model contains precise and up-to-date correlations for the heat exchange coefficients and pressure losses. Convection and pressure loss correlations are valid for any flue gas composition, and for water in any phase (liquid, vapour or two-phase flow). The conduction equation is adapted with a fouling coefficient. To feed these equations, each model contains a very accurate set of geometrical data (technology, number of tubes, lengths, diameters, characteristic of the wings, etc).
- The steam turbine component model is based on an ellipse law and an isentropic efficiency.
- The aero-condenser component model is based on correlations of the manufacturer.
- The pump models are based on the characteristic curves of the pumps.
- The collector is based on the mass and energy balances for the fluid.
- The pressure drop in pipes is proportional to the dynamic pressure \pm the static pressure.

The Modelica translation required some reverse engineering of the component models, because :

- the LEDA components are Fortran codes, containing many procedural if ... then ... else constructs that cannot be written as such in Modelica,
- the LEDA documentation is sometimes incomplete.

Each new Modelica component had to be tested separately in order to remove as many modelling errors as possible before building the full Rio Bravo model. This task was not easy, as a model written in a declarative language has a totally different structure from a model written in a procedural language, so the LEDA experience was no great help.

In spite of these difficulties, it was possible to rewrite the complete LEDA modules in Modelica. The new Modelica component models behave exactly as their LEDA originals.

5. The Rio Bravo model

The full model is built by connecting the component models in a technological way, so that its topology reflects the functional schema of the plant (see Figure 3 in the appendix). It is composed of 91 component models, generating 5200 variables and 1100 non-trivial equations.

The model consists of 16 exchangers (3 evaporators, 6 economizers, 5 super-heaters and 2 re-heaters), 3 balloons, 3 steam turbine stages (HP, IP and LP), 6 pumps, 3 valves, 1 kettle boiler, 9 pressure drops, several mixers, several collectors, 1 power imposed and 1 aero-condenser.

The order of the exchangers downstream the exhaust gas flow is :

- the third HP super-heater,
- the second IP re-heater,
- the second HP super-heater,
- the first IP re-heater,
- the first HP super-heater,
- the HP evaporator,
- the fourth HP economizer,
- the IP super-heater,
- the third HP economizer,
- the LP super-heater,
- the second HP economizer,
- the IP evaporator,
- the first HP economizer and IP economizer,
- the LP evaporator,
- the LP economizer.

The low pressure loop

At the exit of the aero-condenser, water is pumped and heated in the LP economizer, then is sent to the LP balloon. The water leaving the LP balloon goes to the LP evaporator, the IP loop and to the HP loop. The produced steam is transmitted to the LP super-heater, then to a mixer with a pressure loss, then to the LP turbine. The steam at exit of the mixer is a mixture of the steam at the exit of the LP balloon, the steam at the exit of the IP turbine and the steam of racking at the HP turbine exit.

The intermediate pressure loop

At the exit of the LP economizer, water is pumped and heated in the IP economizer, then is sent to the IP balloon. The water leaving the IP balloon goes to the IP evaporator and the produced steam is transmitted to the IP super-heater, then to a mixer (to mix the steam at the exit of the IP super heater and the steam at exit of the HP turbine), then is sent to the first IP re-heater,

then to the IP de-superheating, then to the second IP re-heater, then to the IP turbine.

The high pressure loop

At the exit of the LP economizer, water is pumped and heated in the first, the second, the third and the fourth HP economizer, then is sent to the HP balloon. The water leaving the HP balloon goes to the HP evaporator. The produced steam is transmitted to the first HP super-heater, then to the second HP super-heater, then to the HP de-superheating, then to the third HP super-heater, then to the HP turbine.

The aero-condenser

At the exit of the turbine LP, the steam is condensed in the aero-condenser, then the water is pumped and sent into the LP economizer.

The steam bleeding

Steam bleeding at the entry and the exit of the HP turbine is necessary to ensure the tightness of the bearing of the IP and LP turbines.

The model was easy to build from the graphical library : as opposed to LEDA, no causality analysis is required from the user, this task being handled automatically by the code generator.

But it was initially difficult to converge, because of the lack of information about the iteration variables chosen by the code generator (iteration variables are variables that need to be properly initialised by the user in order to compute the system equations for the first time step). Once this information was correctly provided by the tool, it was fairly easy to make the model converge. This task, which is equivalent to computing the initial state, would even be easier if the set of iteration variables were stable across moderate model changes, and, of course, from one version of the tool to another.

So it is very important to provide an efficient way to handle these iteration variables, as the task of setting them properly is time consuming. It is also by no way automatic, since it requires a good expertise of the problem to be solved (the number of iteration accounts roughly 5% of the total number of variables, so can be rather large for a human).

More generally, as the numerical structure of the system equations is automatically generated, it is necessary that the tool provides an efficient way to trace the numerical system back to the original mathematical equations.

6. Model calibration

The calibration phase consists in setting (blocking) the maximum number of thermodynamic variables to known measurement values (enthalpy, pressure), taken from on-site sensors during performance tests. This method ensures that all needed performance parameters, size characteristics and output data can be computed.

The main computed performance parameters are :

- the fouling coefficients of the exchangers,
- the ellipse law coefficients of the turbines,
- the isentropic efficiencies of the turbines,
- the pressure drop corrective coefficients of the exchangers and of the pipeline between the equipments.

Example : the pressure of the vacuum in the aero-condenser or the flow rate of the circulating cooling are computed from the available measurements.

7. The thermodynamic properties

Properties of fumes

The thermo-physical properties of the fumes (for the exchangers, the aero-condenser and the collectors) were computed using Fortran tables called MONOMELD, which are normally used with LEDA. Using the same tables for both tools facilitated the comparison of the simulation results.

Properties of water and steam

The properties for water and steam were computed from polynomials defined by the international standard IAPWS-IF97. The efficient original Modelica implementation of H. Tummescheit was used. LEDA utilizes a variant of this standard implemented as a look-up table.

8. The simulation results

A preliminary calibration of the model was made against measurement data obtained from on-site sensors. The model was then able to compute precisely the distribution of water and steam mass flow rates, pressure and temperature across the network, the exchangers thermal power, and the performance parameters of all the equipments. It converges very quickly, provided that the iteration variables (approx. 5 % of the total number of variables) are properly fed in by the user.

The table below shows the differences between the LEDA and the Dymola numerical computation results, for the reference conditions (i.e. 100% nominal power – natural gas – yearly average conditions).

Table 1 - Differences between the results of LEDA and DYMOLA

	LEDA	DYMOLA	Δ
Electric power output (MWe)	177,33	177,42	-0,09
Pressure in HP balloon (bar)	135,0	134,0	1
Output steam flow at HP balloon (kg/s)	52,67	52,31	0,36
Output steam temperature at HP balloon (°C)	561,1	566,6	-5,5
Pressure in IP balloon (bar)	31,8	31,8	0,0
Output steam flow at IP balloon (kg/s)	8,84	9,08	-0,24
Output steam temperature at IP balloon (°C)	310,6	310,9	-0,24
Pressure in LP balloon (bar)	5,11	5,17	-0,06
Output steam flow at LP balloon (kg/s)	77,27	77,37	0.1
Output steam temperature at LP balloon (°C)	180,75	180,13	-0.3
Power exchanged in the aero-condenser (Mw)	357,80	357,91	0.03
Temperature of the exhaust fumes of the boiler (°C)	113.9 5	113.8	-0.13

The slight differences between the results of the two codes are due to the fact that the thermodynamic properties of water and steam are computed in different ways (as noted before, LEDA uses a variant of the IF97 standard).

The robustness of the model for different operating points was tested by varying the flow of the fumes in the range from 250 to 700 kg/s (250, 300, 350, 400, **464.48**, 500, 550, 600, 650, 700).

9. Conclusion

A static and rather large model of the Rio Bravo power plant has been translated from LEDA to Modelica with Dymola to evaluate the capacity of

Modelica based tools to perform steady state direct and inverse computations for the sizing of power plants.

The translation has been done without any loss of information from the original model, and at an acceptable, though still high cost.

To even further reduce the effort required to do Modelica modelling and simulation for such systems, it is necessary to provide more advanced tool functionalities to handle efficiently the iterations variables, and trace the automatically generated numerical system back to its original mathematical equations, as declared by the user with the Modelica language.

Nevertheless, this work shows that the Modelica technology is mature enough to replace proprietary solutions such as LEDA for the steady state modelling and simulation of power plants.

References

- [1] Avenas C. *et al.* Quasi-2D steam generator modelling with Modelica. ISC'2004, Malaga, Spain.

Appendix

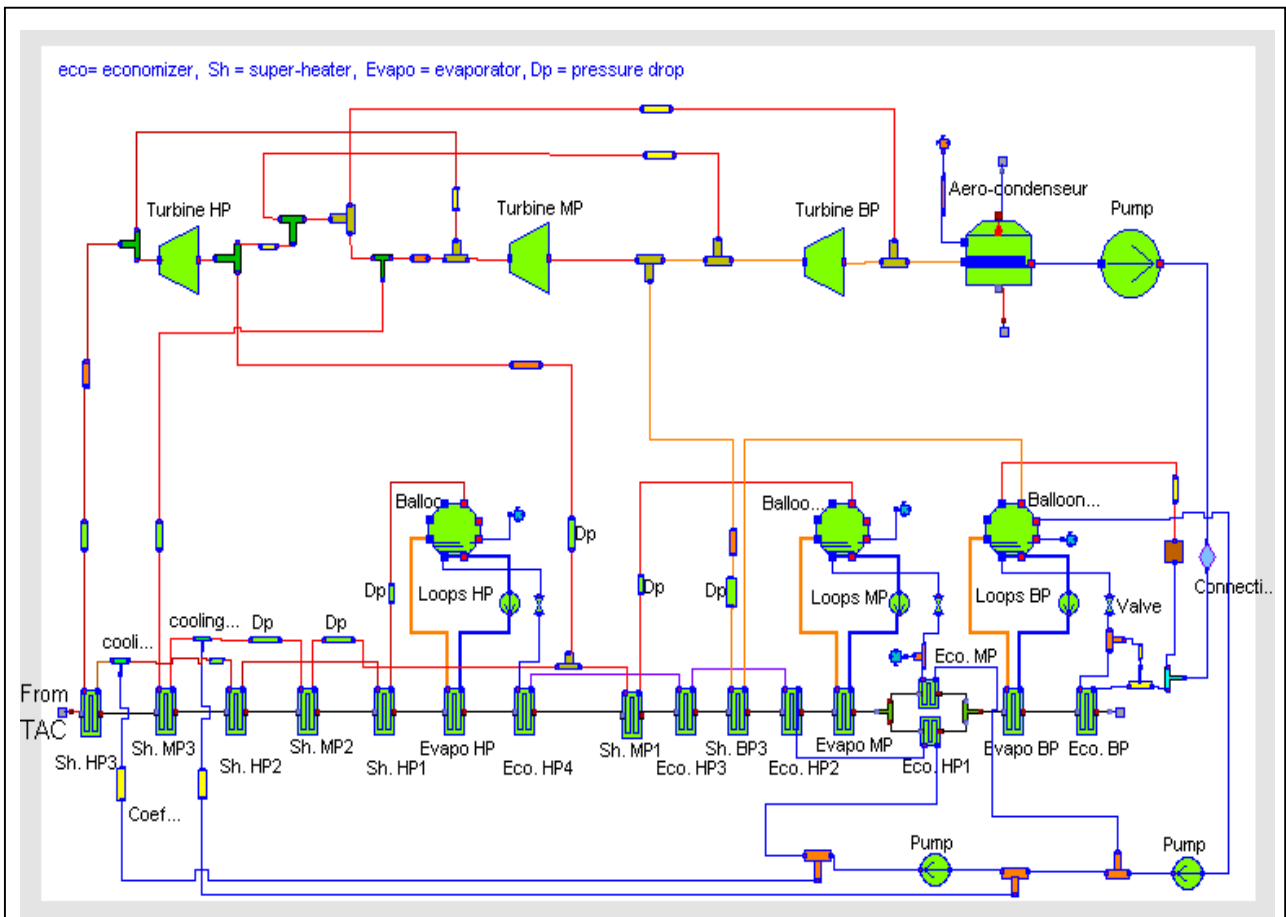


Figure 3 - The Rio Bravo Modelica model

Modeling and Dynamic Analysis of CO₂-Emission Free Power Processes in Modelica using the CombiPlant Library

Jonas Eborn Faruk Selimovic† Bengt Sundén†

Modelon AB
IDEON Science Park, SE-223 70 Lund

†Department of Energy Sciences
Division of Heat Transfer
Lund Institute of Technology, Box 118 SE-221 00 Lund

Abstract

The need to reduce CO₂ emissions from fossil-fuel based power production creates the need for new power plant solutions where the CO₂ is captured and stored or reused. Different concepts to capture CO₂ fall into the three main categories:

1. Precombustion decarbonization
2. Oxy-fuel combustion
3. Post-combustion removal of carbon.

In the first two types of processes Oxygen Transport Membrane (OTM) is the key component, as pure oxygen is usually required to process reactions (e.g. Integrated Gasification Combined Cycle IGCC, Advanced Zero Emission Plant AZEP). Post-combustion removal processes can for example utilize adsorption/desorption in certain salt solutions. This paper will describe two different applications of CO₂-emission-free processes, one using an OTM, the other a high pressure post combustion removal process, the Sargas process, which has been modeled in a project with Siemens Industrial Turbomachinery AB and Alstom Power Sweden AB. All modeling work was carried out in the modeling language Modelica, which is an open standard for equation-based, object-oriented modeling of physical systems. System models have been built using the CombiPlant library, a modeling library for combined cycle power plants from Modelon AB.

Keywords: power plant modeling; OTM; CO₂-removal; oxy-fuel combustion;

1 Introduction

Future profitability of power generation will involve, besides fuel and investment costs, even a trading of plant CO₂-emissions. Today, the use of coal and other low-grade fossil fuels are dominant for power generation, about 80%. Gas fired power plants produces about 20% of the total power output and an increased number of natural gas fired combined cycle power plants would result in lowering of CO₂ emissions.

1.1 Sargas process description

Sargas AS, a Norwegian company, has developed technology for separating CO₂ and NO_x from power plant flue gas. The Sargas process is a combined cycle system consisting of a gas turbine with an external pressurized combustion chamber in combination with a conventional steam cycle. This part of the process is a modified version of existing pressurized fluidized bed combined cycle (PFBC) power plants. The removal of the CO₂ takes place at high pressure after the combustion chamber. This minimises the volume of flue gas to be purified relative to the amount of power produced, providing near full CO₂ capture (more than 95%) and substantial reduction of NO_x (5ppm).

The Sargas process flow sheet can be seen in Figure 1, along with the corresponding model diagram in Figure 2. The air from the gas turbine compressor and natural gas are combusted in the pressurized boiler. The combustion process can take place at a low level of excess air (2% O₂ in exhaust gas). This will result in a higher concentration of CO₂ in exhaust gas than in conventional gas-fired Combined Cycle Power Plants (CCPP). The combination of elevated pressure

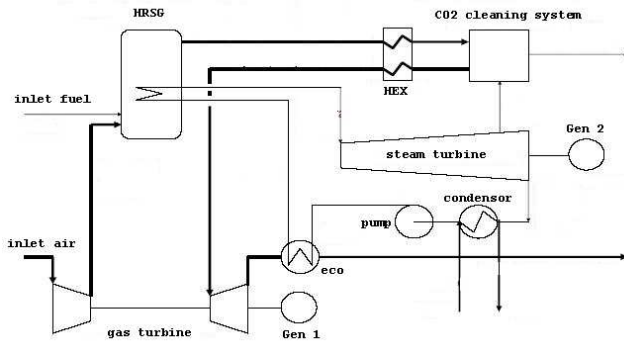


Figure 1: Sargas process flow sheet.

and high CO₂ concentration results in high CO₂ partial pressure, thus increasing the efficiency of the CO₂ separation process compared to conventional CCPP. Steam produced in the boiler is used to generate electricity from a conventional steam turbine. The temperature of boiler exhaust gas is approximately 850°C, somewhat lower than in CCPP. The exhaust gas is cooled in heat exchangers down to the optimal temperature (≈70°C) for the CO₂ separation process, which is an absorption/desorption process employing a salt solution as the working fluid.

After the CO₂ separation process takes place (with an efficiency of approximately 90% in this application), the exhaust gas with less CO₂ is reheated to about 840°C and expanded through the gas turbine to produce further electricity, before passing to the stack.

1.2 Oxy-fuel emission free power cycles

Compared to the capture processes which use complicated separation processes, the oxy-fuel power cycles uses pure oxygen in the combustion of fuel. Exhaust gases resulting from the combustion will therefore consist of mainly water and carbon dioxide. The exhausts can easily be cooled to condense the water leaving the carbon dioxide for further storage. OTM is an important part of novel oxy-fuel power cycles such as AZEP and Chemical Looping Combustion (CLC) where OTM is integrated in the system to enable stoichiometric combustion with oxygen.

The key of AZEP concepts is substitution of the conventional combustion chamber in a gas turbine by a mixed conducting membrane (MCM) reactor, which combines oxygen production, fuel combustion and heat transfer [1, 2]. The MCM reactor contains oxygen transfer membrane being surrounded by two High Temperature Heat Exchangers (HTHX), which supply energy needed for oxygen transfer process. The

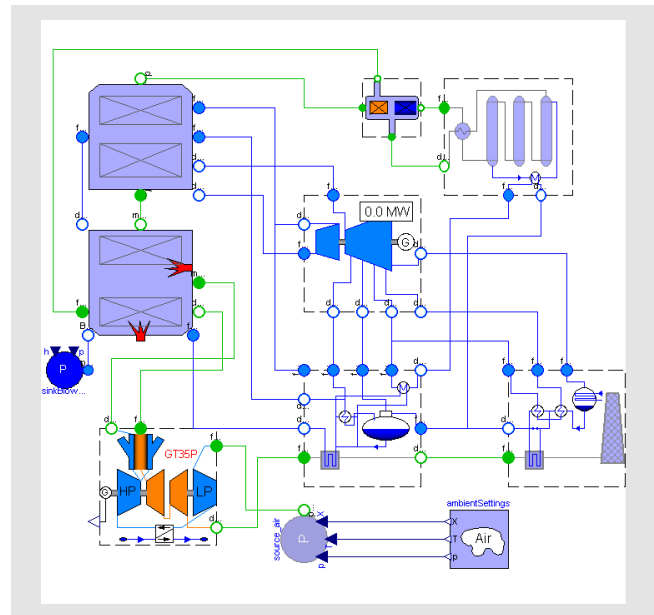


Figure 2: Sargas process model diagram as shown in Dymola window.

membrane is constructed of mixed ion electron conducting material and when heated it transfers the oxygen ions which are exchanged at surfaces with oxygen molecules [3, 4]. As can be seen from Figure 3, compressed low temperature air (500°C) from compressor enters the reactor at the first of two HTHX (Q arrows) which in turn increases the air temperature up to the needed level for oxygen permeation reaction. The oxygen migrates to the exhaust gas (called sweep gas). From the membrane section the air enters the second HTHX (upper Q arrows) where the temperature is raised to a value close to the hot exhaust gas temperature from combustor (1200°C). The hot oxygen depleted air is then led to the power generating turbine. As the oxygen is transferred to the sweep flow, excess of mass on the sweep side and deficit on the air side, respectively, will occur after the membrane section. The bleed gas heat exchanger compensates for this and increases the sweep temperature which at the final stage is used for generation of steam, in a HRSG, which is then expanded in the steam turbine.

The advantage in the power systems using oxy-fuel combustion is that it enables 100% CO₂ capture. However, the need of expensive oxygen separation methods (e.g. cryogenic separation of pressure swing absorption, PSA) would bring oxy-fuel method to its death because of a decreased thermal efficiency, down to only 10-20%. OTM is the key of oxy-fuel processes as it separates oxygen from air at low costs.

2 CombiPlant Library

The CombiPlant library is a commercial Modelica library for the unsteady (transient) simulation of Combined Cycle Power Plants and its components. The CombiPlant library uses well-known, published correlations for heat transfer and pressure drop for fluegas, steam and liquid water. Besides this more advanced user-defined correlations can be easily integrated, and also completely new models such as the oxygen membrane model built and used together with the library components.

Both gas and fluid side models in heat exchangers uses a discretized finite volume model with mass and energy balances for each volume. Two-phase behavior is captured using the integrated mean-density model [5] and by continuously tracking the phase boundary an accurate description of two-phase heat transfer is obtained. The pipe and heat exchanger models can be parametrized with different heat transfer and pressure drop models, which even can be added by the user, e.g. using proprietary correlations.

The library uses steam and fluegas medium models from the new Modelica.Media library [6, 7], which allows easy replacement of the medium models used in component and system models. The library structure contains the following packages:

ControllersAndSensors This package contains controllers and sensors for gas/water stream properties, needed for control of dynamic systems.

Examples is a package with test models and also some ready-to-use combined cycle plant section examples.

FlueGas package contains components, sources and sinks relevant for use with gas media, such as pipe, volume and combustor models.

HeatExchangers This package contains general models of heat exchangers for gas-gas or gas-water operations. Also models for steam/water plate heat exchangers are represented here.

Interfaces package contains connectors for gas and water streams. It is basically a mirror of the Modelica.Fluid and Thermal connector definitions.

Internal consists of several subpackages, providing functions for characteristic numbers, such as Reynolds, Nusselt, and Prandtl number, and other useful numerical functions. Internal also holds subpackages for user choices and component icons.

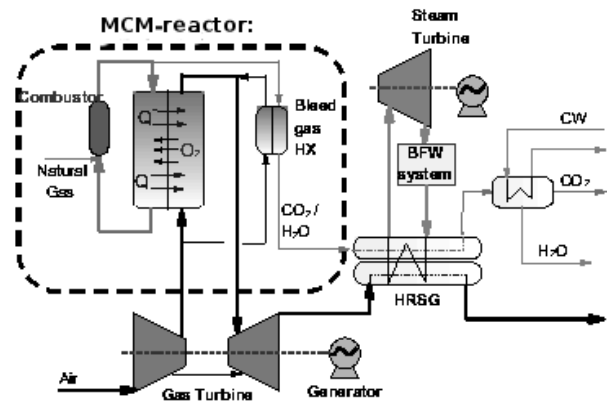


Figure 3: AZEP process flow sheet, inside the dashed square is the reactor system. (from [2])

Pumps package holds pump models described by the characteristic curve.

Water This package contains components, sources and sinks relevant for water/steam media, e.g. two-phase pipes and volumes, boiler drum, spray attemperator, and steam turbine models described by the Stodola equation.

SubComponents includes the subpackages, *Geometry* for heat exchanger geometry descriptions, *HeatTransfer* for different heat transfer correlations used in component models, and *Visualizers* with component models used for dynamic visualization of the plant and section model diagrams.

Valves package contains valves and pressure loss models.

3 Developed Models

The CombiPlant library includes components for conventional combined cycle power plant models. For the two applications described in this paper several specialized components and extensions to the CombiPlant library models were built. Some of the new components and modeling assumptions used are described in this section below.

3.1 Performance trade-offs for heat exchanger models

Cross flow gas-water heat exchangers used in the boiler, superheater and economizer sections of a power plant are modeled using a discretized model. To get

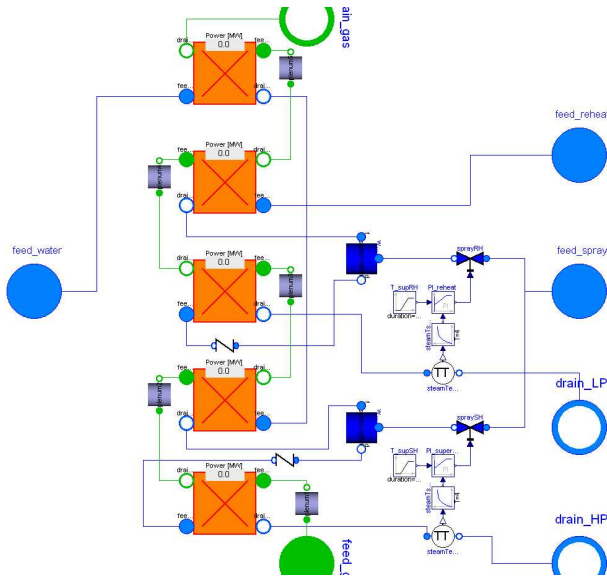


Figure 4: Superheater section model, showing example of how to combine heat exchangers with steady-state balance equations and dynamic plenum volumes.

good steady-state performance it is desirable to have high discretization, but a large number of dynamic states would give very long simulation times. As a trade-off that still retains all the relevant dynamics a quasi-static discretized model is used on the gas side of all HX's. The relevant thermal dynamics of the HX are included in the thermal inertia of the metal walls.

In the section models, several such HX's connected in series on the gas side. This would result in an undesirable static coupling between gas side balance equations, giving large non-linear equation systems that would also result in long simulation times. To avoid this situation, dynamic plenum volumes are introduced between HX's as can be seen in Figure 4. Dynamic states p , T , X are forced on the plenum volumes, using the Modelica `stateSelect` attribute. The dynamic states provide the boundary conditions for the static flow and heat transfer relations on the gas side of the HX and breaks up the large non-linear equation systems. This combination is a trade-off that provides both good steady-state and dynamic performance for discretized models of this type.

In the figure it can also be noted that the plenum volumes are coupled directly to the inlet of each HX without any separate pressure drop description. This is possible due to automatic index reduction, and has no ill effects since there are no gas volume dynamics inside the heat exchanger model.

3.2 Benfield process section model

The key feature of the Sargas process is to deliver power from natural gas without the environmental impact of fossil CO₂ emissions. This is achieved in the CO₂ removal unit, in the upper right corner of Figure 1. The process used is a so-called Benfield process, a standard commercial process using the adsorption/desorption properties of certain salt solutions to remove CO₂. It consists of several stages of condensing and humidifying the flue gas using water, to keep the gas at ideal conditions for the adsorption process. The Benfield section model is a simplified description of the CO₂ removal process without the salt solution circulation loop included. The main purpose of the investigation was to verify the dynamic behavior with respect to the composition and thermal dynamics, and thus no detailed description of the Benfield process was needed. The important moisture and gas thermal and volume dynamics are included via lumped models of the large volumes in the scrubber, condenser, absorber/desorber and humidifier. Moisture condensation and evaporation is assumed to be instantaneous at the current saturation temperature. The absorption itself is represented by a constant efficiency parameter. The model includes the absorption heat taken from the steam flow bled from the steam turbine, as this is important for the overall energy efficiency of the process. Below are the additional mass balance equations for the absorber volume, used to calculate the mass transfer of water and CO₂. The difference between the water saturation pressure and the actual mole fraction of steam in the gas volume is used as the driving force for mass transfer.

$$\begin{aligned}
 dy_{sat} &= (\text{WaterMedium.saturationPressure}(T) / p \\
 &\quad - \text{mole}_y[\text{H}_2\text{O}]) ; \\
 \text{feed.mXi_flow}[\text{H}_2\text{O}] + \text{drain.mXi_flow}[\text{H}_2\text{O}] \\
 &\quad + \text{absorb_flow}[\text{H}_2\text{O}] = \\
 &\quad dy_{sat} * \text{feed.mXi_flow}[\text{H}_2\text{O}] ; \\
 \text{feed.mXi_flow}[\text{CO}_2] * \text{eta_absorb} \\
 &\quad + \text{absorb_flow}[\text{CO}_2] = 0 ;
 \end{aligned}$$

The names `feed` and `drain` refer to the gas flow connectors on the volume, the parameter `eta_absorb` is the CO₂ removal efficiency parameter.

3.3 Oxygen transport membrane reactor model

Oxygen Transport Membrane (OTM) consists of dense ceramic membrane. It is generally accepted that such dense membranes have significant future potential in the gas and energy industries with a wide variety of

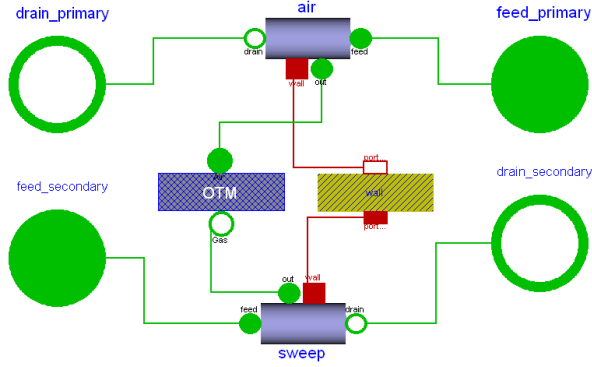


Figure 5: Model diagram view of the membrane reactor model: *OTM* -Oxygen transfer membrane, *Air* -air flow model, *Sweep* -exhaust gas flow model, *wall* -wall model for heat transfer.

applications, such as, the separation of oxygen from air and the conversion of natural gas to syngas. The OTM is usually constructed of mixed ion electron conducting material and when heated it transfers the oxygen ions which are exchanged at surfaces with oxygen molecules [3]. Energy for heating the membrane can be exchanged from process exhaust gases. The most attractive membrane materials today which have been employed successfully in membrane reactors are: $\text{Ba}_{0.5}\text{Sr}_{0.5}\text{Co}_{0.8}\text{Fe}_{0.2}\text{O}_{3-\delta}$ (BSCFO) and $\text{BaCo}_{0.4}\text{Fe}_{0.4}\text{Zr}_{0.2}\text{O}_{3-\delta}$ (BCFZO), [8].

The membrane reactor model was conducted from existing flow models from CombiPlant library with the exception of the model for oxygen transfer which in its turn is developed and introduced into library. Since the oxygen transfer model involves mass transfer operation of only oxygen as a single component, the Modelica `semiLinear` function for calculation of the fluid flow and fluid enthalpy in connectors is not suitable. The `semiLinear` function can only be applied to well mixed flows. Instead, the code below has been used. Media model of air for this case has 5 different components, (ex. Moist air with Ar, CO₂, H₂O, N₂, O₂), and the mass flow rates of the four non-permeable ones were set to zero.

```

m_flowO2 =
  J_O2*memPars.A_mem*Medium.data[5].MM;
Air.H_flow      = h_O2*m_flowO2;
Gas.H_flow      = -h_O2*m_flowO2;
Gas.mXi_flow[1:4] = 0,0,0,0;
Gas.mXi_flow[5]  = -m_flowO2;
Gas.m_flow      = -m_flowO2;
Air.mXi_flow[1:4] = 0,0,0,0;
Air.mXi_flow[5]  = m_flowO2;
Air.m_flow      = m_flowO2;
    
```

Gas and Air are the flow connectors on each side of the membrane, connected to the corresponding pipe model. The code above takes care of mass flow rate of oxygen and assigns this value to connector. Oxygen permeation rate J_{O_2} has been traditionally calculated by the Wagner equation:

$$j_{O_2} = \frac{1}{16F^2d} \int_{\mu_{1s}}^{\mu_{2s}} \frac{\sigma_i \sigma_e}{\sigma_i + \sigma_e} d\mu \quad (1)$$

where j is the permeation flux density of molecular oxygen, d is the membrane thickness, σ_i and σ_e are the partial ionic and electronic conductivities, μ is the oxygen chemical potential. Oxygen chemical potential is expressed here as:

$$\mu = RT \log(p_{O_2}) \quad (2)$$

Combination of eq. 2 and eq. 1 and integration of 1 with the fact that oxygen ions are much slower than electron gives following expression:

$$j_{O_2} = \frac{c_i D_a}{4d} \ln \frac{p_{O_2}^1}{p_{O_2}^2} \quad (3)$$

where c_i is the density of oxygen ions, D_a represents ambipolar diffusion coefficient of oxygen ion-electron hole pairs, d is the thickness of membrane, $p_{O_2}^1$ and $p_{O_2}^2$ is the oxygen partial pressure for low and high oxygen partial pressure sides across membrane respectively. The ambipolar conductivity was assumed to have Arrhenius dependence on temperature:

$$D_a = D_a^0 e^{\frac{E_a}{R} \left(\frac{1}{T} - \frac{1}{1273.15} \right)} \quad (4)$$

where D_a^0 is the preexponential factor and E_a is the activation energy for the ambipolar conductivity. The BSCFO membrane possesses high oxygen ion conductivity. For the predominantly BSCFO mixed-conductor the D_a^0 is expected to be close to the ionic self-diffusion coefficient D_i ($D_a \approx D_i$), and then the Nernst-Einstein equation is applicable to calculate the oxygen ionic conductivity of BSCFO:

$$D_i = \frac{RT \sigma_i}{4c_i F^2} \quad (5)$$

The experimental measured oxygen flux in [8], has been used to express correlation of ionic conductivity of BSCFO in this work, Figure 6.

4 Simulation Results

Examples of transient simulations carried out on the two types of CO₂ free power processes presented are shown here.

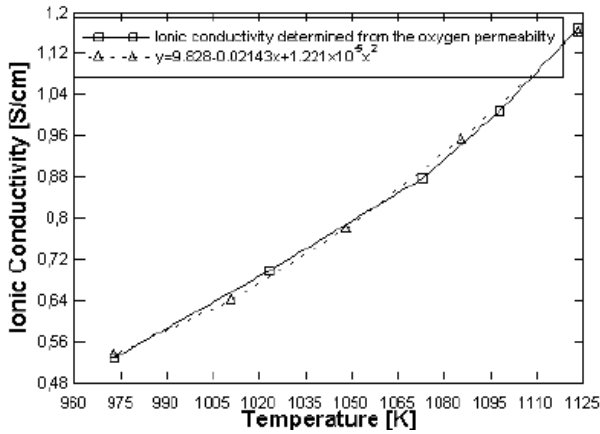


Figure 6: Oxygen ionic conductivity at different temperatures, from [8].

4.1 Load reduction transient on the Sargas plant model

To see the effects of the volume lag from the CO₂ removal plant on the power plant behavior, a load reduction transient has been conducted on the Sargas plant model. The simulation was done on a plant model without the gas turbine, with the load reduction performed by ramping the compressed air flow rate into the boiler from 100% to 80% of the design flow rate. Natural gas flow to the burners was reduced proportionally. A plot of the resulting mechanical power generated by the steam turbines is shown in Figure 7. Generators connected to the gas turbine would contribute another 15 MW at 100% load conditions.

In Figure 8 the mole fraction of CO₂ into and out of the Benfield process section is shown. The process removes 92% of the carbon dioxide in the gas flow. Dur-

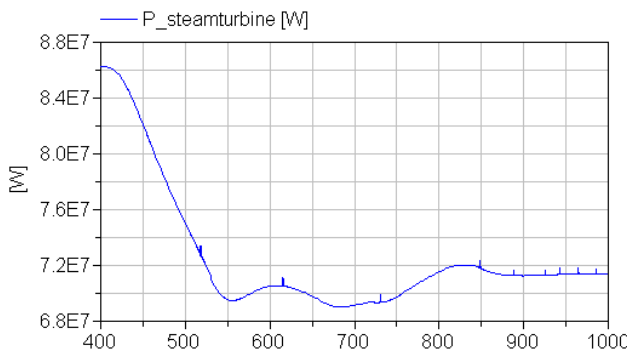


Figure 7: Total mechanical power generated by steam turbines during load turn-down on Sargas plant model.

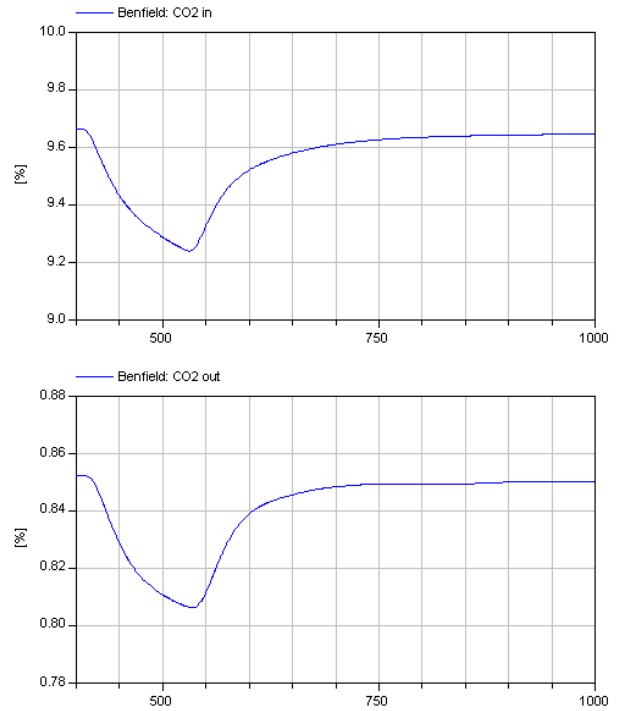


Figure 8: Mole fraction of CO₂ into and out of the Benfield plant section.

ing the load reduction transient between 400 and 520 seconds the inlet composition changes, but the outlet composition follows with little lag. The high volumetric flow rate of about 10 m³/s gives a hold-up of only a few seconds in the gas heat exchangers of the Benfield process.

Figure 9 shows the mass flow of H₂O in the flue gas stream into and out of the Benfield process section. It is important to maintain the water balance and avoid adding or removing process water. In the simulation a control valve hits the maximum limit. This is the reason why the outlet steam flow is larger than the inlet steam flow and thus the water balance can no longer be kept.

4.2 OTM reactor startup transient

A transient simulation test was carried out for the membrane reactor model shown in Figure 5 with the OTM integrated into 3.5 mm OD tubes where one side of the membrane was exposed to air (total air flow 10kg/s), while the other side was exposed to exhaust/sweep gas (total gas flow 1kg/s). Design parameters for membrane tubes and size of reactor are easily set in the standard geometry parameter dialog from the CombiPlant library, Figure 10.

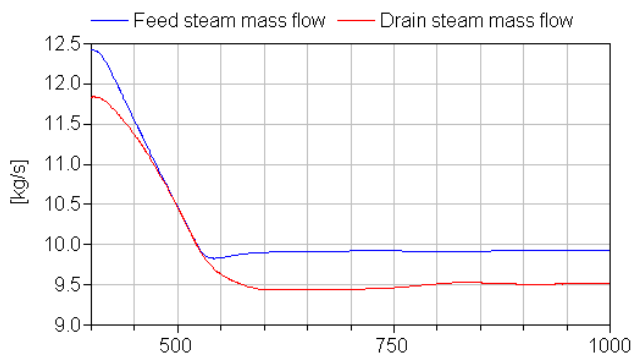


Figure 9: Mass flow of steam in the gas flow into and out of the Benfield plant section during load turn-down. Humidity control tries to match the flows to keep the plant water mass balance and avoid using make-up water.

Simulations show that a steady state condition is reached after approximately 3 hours, see Figure 11. BSCFO membrane material shows relatively short start up time compared to the SCFO materials which can take 500 hours until reaching steady state operating condition.

5 Final Remarks

The paper shows how the CombiPlant library, with components for standard combined cycle power plants, can be extended and used to build component and plant models for power plant concepts providing CO₂ emission free power. The library was used in a project with Siemens Industrial Turbomachinery to build a dynamic model of the Sargas power plant concept, which uses the commercial Benfield process to separate up to 95% of the CO₂ from the exhaust gases. In another application example, mass transfer through an oxygen transfer membrane has been described. This is a critical component in oxy-fuel combustion cycles such as AZEP.

References

- [1] Sundkvist, S.G., T. Griffin, and N.P. Thorshaug, AZEP - Development of an integrated air separation membrane - gas turbine, In *Proceedings of Second Nordic Minisymposium on Carbon Dioxide Capture and Storage*, Gothenburg, Sweden, 2001.
- [2] Griffin, T., S.G. Sundkvist, K. Åsen, and T. Bruun, Advanced Zero Emissions Gas Turbine Power Plant, In *Proceedings of ASME Turbo Expo 2003*, Atlanta, USA, 2003.
- [3] Selimovic F., B. Sundén, M. Assadi, and A. Selimovic, Computational Analysis of an O₂ Separating Membrane for a CO₂-Emission-Free Power Process, In *Proceedings of Asme, IMECE2004-59382*, 2004.
- [4] Selimovic F., *Modeling of Transport Phenomena in Monolithic Structures Related to CO₂-Free Power Processes*. Licentiate Thesis, ISSN 0282-199, Department of Energy Sciences, Lund Institute of Technology, Sweden, 2005.
- [5] Casella, F., Object-Oriented Modelling of Two-phase Fluid Flows by the Finite Volume Method. In *Proceedings of 5th MATHMOD*, Vienna, Austria. Argesim, Vienna, February 2006.
- [6] Casella, F., M. Otter, K. Prölb, C. Richter and H. Tummescheit, The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In *Proceedings of the 5th International Modelica Conference*, Vienna, Austria. Modelica Association, September 2006.
- [7] Tummescheit H., *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis ISRN LUTFD2/TFRT-1063-SE,

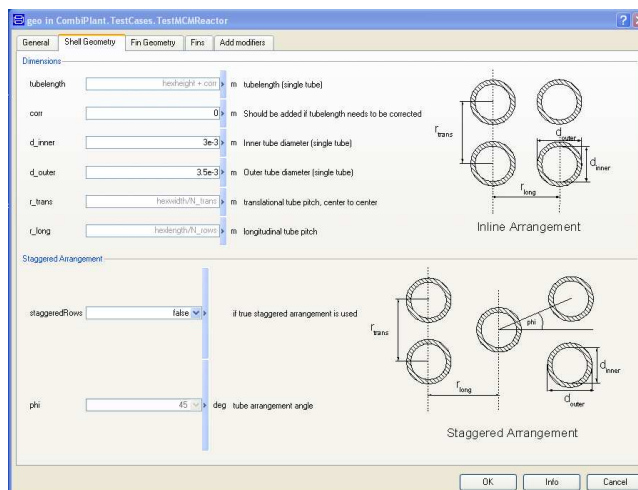


Figure 10: CombiPlant dialog for specifying heat exchanger geometry parameters. Friendly user interface provides help to input all required design parameters used in simulations.

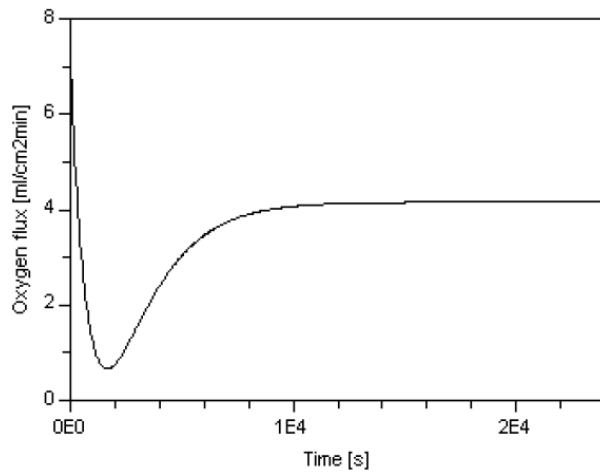


Figure 11: Oxygen permeation flux of the of the membrane reactor as a function of time.

Department of Automatic Control, Lund Institute of Technology, Sweden, August 2002.

- [8] Lu H., Y. Cong, and W.S. Yang, Oxygen permeability and stability of Ba_{0.5}Sr_{0.5}Co_{0.8}Fe_{0.2}O_{2-d} as an oxygen permeable membrane at high pressures, Solid State Ionics, Vol.177, pp.595-600, 2006

Session 1b

Automotive Applications 1

Simulation of Hybrid Electric Vehicles

Dragan Simic Harald Giuliani Christian Kral Johannes Vinzenz Gragger
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 the fuel consumption of two vehicles is compared. For investigating the fuel consumption a whole vehicle simulation is developed with the *SmartPowerTrains* library and the *SmartElectricDrives* library. Both libraries are written in *Modelica* language. As a reference vehicle a conventional vehicle with manual transmission is modeled. A parallel hybrid electric vehicle with a starter/generator and two transition machines at the front and rear drive train wheels is compared with this reference vehicle. The operating strategy of the hybrid electric vehicle is explained. Furthermore, an analysis for finding the optimal sizes of the electric machines for the parallel hybrid electric vehicle using the developed simulation tools and the developed operating strategy is presented. The practical operation modes of the parallel hybrid electric vehicle are considered with regard to the implemented drive train configuration. A dynamic operating method is developed to determine the optimal power split between the internal combustion engine and the electric energy sources. The computer simulation results show the improved fuel consumption of the hybrid electric vehicle.

Keywords: hybrid electric vehicle; operational strategy; drive train configuration; optimization

1 Introduction

The focus of this paper is to compare the efficiency and the fuel consumption of a conventional vehicle and hybrid electric vehicles (HEVs) with scaled electric components. The *SmartPowerTrains* (SPT) library focussing on hybrid electric vehicle concepts is used for the presented investigations. The SPT library was developed at Arsenal Research. It is written in *Modelica* language [1]. All mechanical components of a ve-

hicle are modeled with the SPT library. The electrical components in the HEV concepts are implemented using the *SmartElectricDrives* (SED) library. The used components of the SED library are electric machines, power sources, measurement devices, modern electric drive control algorithms and power electronics.

Compatibility with other *Modelica* libraries such as, the new SED library, the new *VehicleInterfaces* library [2], the *Modelica* standard library and the *Modelica.Thermal.FluidHeatFlow*, can be guaranteed to the user due to proper interfaces harmonization throughout the development process. This compatibility gives rise to the development of simulation models with the potential to be expanded to hybrid- and fuel cell configurations of automotive vehicles and electric drives of vehicle auxiliaries [3]. The SPT library is developed to determine the energy flow in the entire vehicle including the energy consumption of electrical- and mechanical components. Furthermore, the fuel consumption and the exhaust emissions of the vehicle and the internal combustion engine (ICE) can be calculated and determined by using SPT library components.

An intelligent operation strategy for optimizing the fuel consumption is essential in HEV concepts. A control signal exchange between the energy sources, energy consumers and all other components of the HEV is facilitated by the implemented bus concept of the SPT library. The energy flow in the HEV can be regulated and optimized independently of the drive mode of each component.

2 The *SmartPowerTrains* library

The SPT library, shown in Figure 1, is developed under the framework of Arsenal Research. This library provides components for modeling and simulating conventional vehicles and HEVs. The electrification of conventional vehicles and vehicle components such as the water pump, the cooling fan and air condition

system give rise to higher efficiency compared to conventional vehicle designs. Mild and full electric vehicles are expected to have the highest efficiency. The simulation of the entire vehicle is needed to find the optimal drive strategy of an HEV. Particularly, the coexistent simulation of the mechanical components and electrical components of the vehicle is crucial. Such simulations are possible with the developed SPT library, the SED library and the *Modelica* standard library respectively.

In the following the function and the structure of the package components of the SPT library are explained:

1. *AuxiliaryComponents*: This package contains the basic components and partial models. Rotational and translational mechanical friction models are included in this package. With the blocks and the models of this package it is possible to implement the vehicle parts such as the friction of the bearing, the clutch and the rolling resistance of the vehicle.
2. *Chassis*: The *Chassis* package includes different chassis models with drive resistances such as rolling-, aerodynamic-, climbing-, and accelerating resistance.
3. *DriveTrains*: Models of belt drives, mechanical and automatical gear box concepts, differentials, power split devices, cardan shafts, chain drives, clutches, etc. are designed in this package.
4. *Electricals*: The electric components such as the starter/generator (SG), the battery and the traction machines can be found in this package. All components of this package are implemented using the the SED library.
5. *Engines*: The *Engines* package contains components of the ICE and the interpolation tables of the fuel consumption and the exhaust emissions. Only mechanical components of the ICE are modeled in this package.
6. *Environments*: Two sub-packages are included in this package: the *Cycle* package providing different cycle models for the determination of the operation cycle and the *Ambient* package with different ambient models. The cycle models are implemented with interpolation tables.
7. *Examples*: This package contains the different concepts of vehicles.

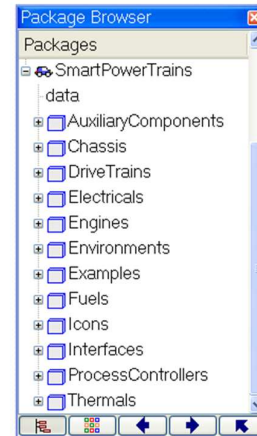


Figure 1: The package browser of the *SmartPowerTrains* library in *Modelica*.

8. *Fuels*: In this package the fuels are defined.
9. *Interfaces*: The *Interfaces* package groups the *Modelica* standard bus connectors, *VehicleInterfaces* bus connectors and advanced bus connectors.
10. *ProcessControllers*: This package contains the different blocks and models for implementing the necessary control algorithms such as the virtual driver, the operating strategy for vehicles, etc.
11. *Thermals*: The *Thermals* package includes the thermal models of the thermal management such as for instance the cooler, the pipeline, the water pump, etc.

The communication between the SPT and SED libraries as well as the independent control of each component allow the highest flexibility in the design of HEV concepts. In the SPT library, the components of the power train and the remaining vehicle components are implemented by algebraic and differential equations whereas the ICE is implemented by characteristic curves. All parameters of the algebraic differential equations are defined by geometrical data of the specific components.

3 The *SmartElectricDrives* library

The SED library uses basic models of electric machines which already exist in the *Modelica* standard library [4]. However, without suitable drive control blocks, these machine models cannot be utilized in a resourceful and easy way. Based on the machine library, the SED library facilitates the modelling of dif-

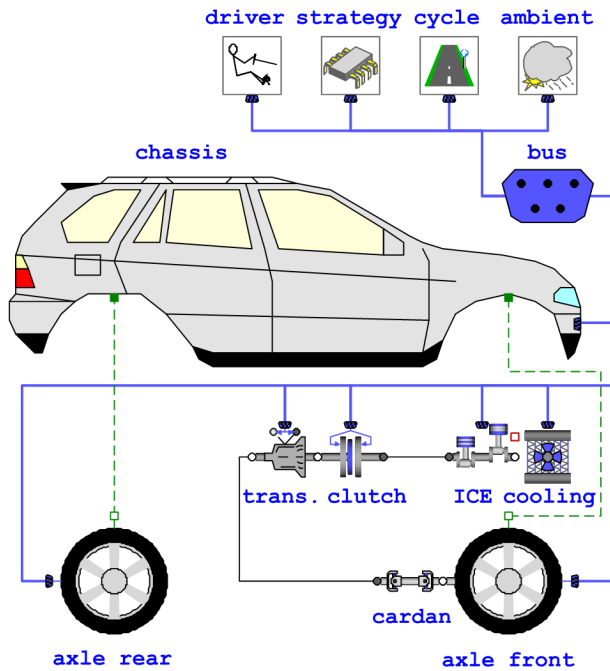


Figure 2: Model of a conventional vehicle.

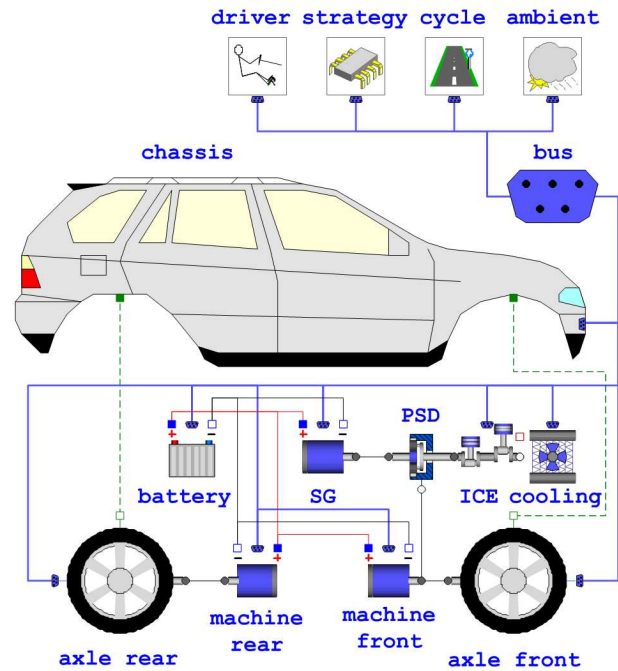


Figure 3: Model of a HEV.

ferent control structures and control strategies. Power electronics and energy storage models are included as well [5].

4 The example of a conventional vehicle

Figure 2 shows a conventional vehicle concept with front axle drive. All vehicle components are connected by a expandable bus connectors. These bus connectors are extended models based on the *VehicleInterfaces* connectors [2]. The mechanical connection of rotational components is indicated with solid lines and circled flanges. The dashed lines show the mechanical connection between the translational components.

In the block *driver* the behaviour of a virtual driver is implemented. The virtual driver controls the state of the following variables: the gas pedal position, the brake pedal position, the clutch pedal position and the gear ratio of the transmission. The block *cycle* and the block *ambient* define the desired vehicle velocity and the state of the environment temperature, the pressure, the density, etc. The entire control unit of the vehicle is implemented in the *strategy* block.

The drive train is realised using a conventional ICE with a simple *cooling* circuit, a *clutch*, a conventional manual transmission, marked as *trans.*, a *cardan* and the front axle. Both axles include two

tire models considering slip, two brake models and a differential model considering losses. The vehicle handling is implemented in the *chassis* model.

5 The example of an HEV

The model of the HEV is based on the *Toyota Lexus* concept. In this HEV concept three electrical machines are used. One electrical machine is used as the starter/generator, indicated as *SG* in Figure 3, the two other machines are traction machines (*machine front* and *machine rear*). The *SG* is coupled with the sun shaft of the power split device (*PSD*) and the *ICE* is coupled with the carrier shaft respectively. The *machine front* is coupled with the axle front via the front differential input shaft and the *machine rear* is coupled with the axle rear via the rear differential input shaft, respectively.

A battery model taken from the *SED* library is used as the energy source for the electrical components of the HEV model. The intelligent operational strategy is implemented in the *strategy* block. The operational strategy is implemented using the *Modelica_LinearSystems* library. All other components and models of the HEV are the same as in the conventional vehicle simulation.

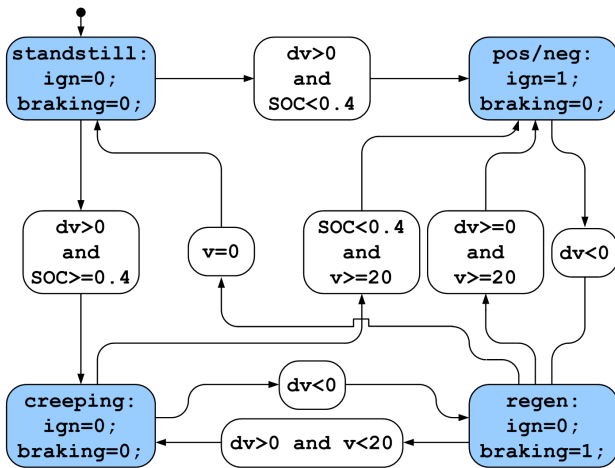


Figure 4: Bubble diagram of the operation strategy

6 The operational strategy of the HEV

For the optimization of the energy consumption of an HEV concept an intelligent operational strategy is essential. The operation of energy sources (ICE, SG and traction machines) in an HEV drive train is monitored and controlled with the intelligent operation strategy. Each energy source is regulated independently based on the state of the entire HEV system.

6.1 Operation modes

In this HEV concept the operation modes of the ICE and the brake is defined as a state algorithm in the intelligent operation strategy. In this HEV concept the operation strategy is implemented using a state algorithm that defines the main operation modes determined by the state of the ICE and the state of the brake. The following variables effect the ICE state and the brake state: the vehicle velocity, the vehicle acceleration and the state of charge (SOC) of the battery. Figure 4 represents basically the switch logic between the possible modes [6]. This operation strategy has four main modes: the *standstill* mode, the *creeping* mode, the *regen* mode and the *pos/neg* mode. In each mode the ICE state, indicated as *ign* (ignition), and the brake state, indicated as *braking* is defined. The switching between the main modes is shown in Figure 4. The variable *v*, with the non-SI unit km/h, is the vehicle velocity and the variable *dv* is vehicle acceleration.

Standstill mode

If the vehicle is in *standstill*, the vehicle velocity is zero and the ignition of the ICE is off. If the SOC of the battery is low, or the battery and the ICE need to be warmed up, the ignition of the ICE is on and working near the idle speed to facilitate the charging of the battery.

Creeping mode

If the vehicle speed is lower than 20km/h, the creeping mode is active. In creeping mode the operation strategy is divided into three sub-modes:

1. If the desired torque is not large and the SOC of the battery is not below the lower limit, the desired torque is distributed to both traction machines.
2. If the desired torque is very large and the SOC of the battery is above the lower limit, the desired torque is distributed to both traction machines and the SG.
3. If the SOC of the battery is below the lower limit, the ignition of the ICE is on and the battery gets charged.

Pos/neg mode

In *pos/neg* mode the ICE, the SG and the traction machines are operated such way that maximum fuel economy is achieved. Depending on the SOC of the battery and the desired torque, the calculation of the distributed torque can be divided into three different sub-modes: positive power split, parallel drive and negative power split.

1. *Positive power split*: In this sub-mode, the SG is operated as a generator. If the SOC of the battery is below the lower limit, the battery gets charged. As long as the SOC of the battery is not above the upper limit, the battery gets charged. If the SOC of the battery is between the lower and the upper limit, the distribution of the torque is used to achieve higher efficiency of the ICE.
2. *Parallel drive*: If the *parallel drive* mode is active, the vehicle is driven by both traction machines and the ICE. The distribution of the torque is used to achieve higher efficiency of all power sources (the ICE, the front and rear electric machine and the battery).

3. *Negative power split*: If the SOC of the battery is very large, the SG is operated as a motor. In this case the speed of the SG accelerates in the negative direction and the speed of the ICE becomes much lower. If the vehicle velocity is very large, this sub-mode is active.

Regenerative mode

This *regenerative* mode, indicated as *regen* in Figure 4, is active if the brake pedal is applied or the deceleration of the vehicle is determined. In this case the ignition of the ICE is off, and consequently, the ICE gets stopped. Both traction machines are operated as generators. If the vehicle is in the *regenerative* mode and the SOC of the battery is below the upper limit the battery gets charged.

6.2 Torque distribution

In each operation mode the desired torque (reference system power, P) is distributed. This torque is defined by the virtual driver and by the power split equation (1). In this equation, ω_{ICE} is the speed of the ICE and τ_{ICE} is the torque of the ICE, ω_{SG} is the speed of the SG and τ_{SG} is the torque of the SG, ω_{FM} is the speed of the front traction machine and τ_{FM} is the torque of the front traction machine and ω_{RM} is the speed of the rear traction machine and τ_{RM} is the torque of the rear traction machine. The behaviour of the power split in a planetary gear, PSD, in Figure 3, is extracted from [7].

$$P = \omega_{ICE} \cdot \tau_{ICE} - \omega_{SG} \cdot \tau_{SG} + \omega_{FM} \cdot \tau_{FM} + \omega_{RM} \cdot \tau_{RM} \quad (1)$$

7 Sources data

The ICE of both investigated vehicle concepts is a gasoline engine. The following parameters are taken for modeling and parameterizing the ICE [8]: 1.5 cc displacement, 43 kW maximum power at 4000 rpm, 102 Nm maximum torque at 4000 rpm, four cylinders. The fuel consumption map of this ICE is defined according to [9].

The following parameters are taken for modeling and parameterizing the Ni/MH battery of the HEV concept: 1.2 V cell voltage, 6 cells per module, 38 modules, 273.6 V battery package voltage, 6.5 Ah rated capacity.

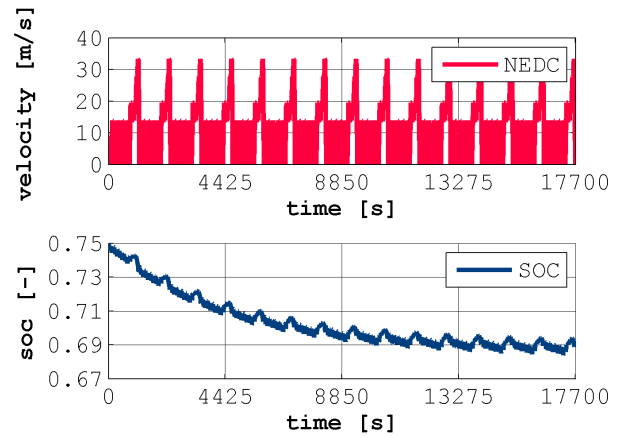


Figure 5: The simulated values of vehicle velocity and battery SOC

Both traction machines are 3 phase alternating current (AC) permanent magnet synchronous machines with an output power of 33 kW. The SG has an output power of 13 kW and is also implemented as an permanent magnet synchronous machine. All electric machines are modeled using the SED library.

The parameters of the front axle and the rear axle (brakes, differentials, inertia of rotational parts, etc.) are defined using [10]. Basically, the *Toyota Prius* component data are used to parameterize the HEV components. The only difference between the proposed HEV concept and the *Toyota Prius* is that the proposed HEV concept is featured with one electric traction machine for each axle. Both traction machines have the same rated power.

8 Energy balance

Investigating the fuel consumption of a vehicle it is necessary to assure that the system energy at the start and the end of a test cycle is the same. In this work both vehicle concepts are simulated in the New European Driving Cycle (NEDC). The new NEDC defines vehicle velocity profiles for one urban drive cycle (UDC) as well as one extra-urban drive cycle (EUDC). The simulation time of the HEV simulation is chosen so long that the SOC reaches a steady state. The upper diagram in Figure 5 shows the vehicle velocity of a vehicle concept driving 15 NEDC. The SOC of the battery is represented in the lower diagram in Figure 5. It can be seen that the SOC of the battery is in balance on the last cycle. This value of the SOC is used as the initial value of the SOC for the fuel consumption calculation in one NEDC.

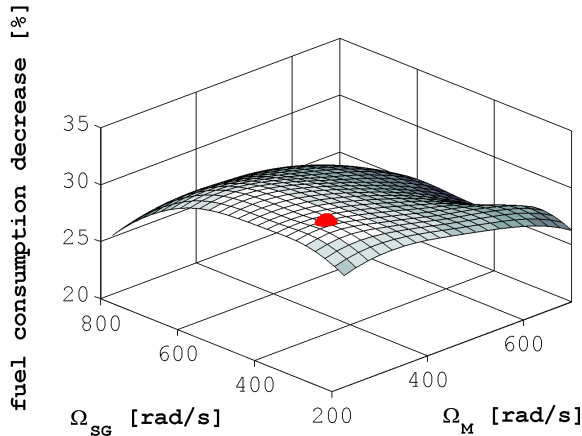


Figure 6: The decrease in the fuel consumption at different nominal speeds of the electric machines in HEV

9 Simulation results

For the optimization of the HEV the nominal operation points of the SG and the front- and rear machine are varied. All other design parameters of this components are kept constant. The nominal speed of the SG, Ω_{SG} , shown in Figure 6, is varied from 200 rad/s to 800 rad/s and the nominal speed, Ω_M , of the front and rear electric machine is varied from 270 rad/sec to 700 rad/s. The size of the used battery is kept constant and the components of the drive train in the HEV are not varied.

Figure 6 shows the simulated results of the HEV with different nominal operation points of the traction machines and the SG. The decrease in the fuel consumption (fuel consumption decrease) is given in percent and shows the economic gain compared to the conventional vehicle concept. The possible decrease in fuel consumption is in the range from 20% to 31.4% for this varied range of the nominal speeds of the electric machines. The best efficiency of the HEV is at $\Omega_{SG} = 310$ rad/s and $\Omega_M = 298$ rad/s. In this point the decrease in fuel consumption is 31.4%.

Here the best design point of an HEV concept is calculated and defined. In the simulations a number of NEDCs is used. Therefore, the simulation results can not be applied to other drive cycles. The simulation results are valid for the performed NEDC only.

10 Conclusions

The implemented vehicle simulation allows the determination of the actual fuel consumption and shows ev-

idence of economic savings potential by using alternative drive train concepts – in this case a HEV. Different vehicle drive train concepts and operating cycles can rapidly be analyzed and tested on fuel consumption and efficiency. Based on the presented results, the design point of the energy sources and the energy consumption of each vehicle component can be calculated and determined. It is shown that the presented libraries can be used to accelerate the design process of innovative vehicle concepts significantly.

This investigation can only be used for the design of vehicles, which are driven in an exact operating cycle such as the exact city drive, the exact reiteration drive, etc.

Based on the calculated decrease in fuel consumption the reduction of energy and emissions of a vehicle on European level can be projected. Focus of the developed process is the realisation of integrated and optimized electric components in automotive industry. For demonstration purpose selected electrical components such as an SG will be built and integrated in a test vehicle in the near future.

11 Abbreviations

HEV	hybrid electric vehicle
SPT	SmartPowerTrains
SED	SmartElectricDrives
ICE	internal combustion engine
SG	starter/generator
PSD	power split device
SOC	state of charge
AC	alternating current
NEDC	New European Driving Cycle
UDC	urban drive cycle
EUDC	extra-urban drive cycle

References

- [1] Peter Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, Piscataway, NJ, 2004.

-
- [2] M. Dempsey, H. Elmqvist, M. Gaefvert, P. Harman, C. Kral, M Otter, and P. Treffinger, “Coordinated automotive library for vehicle system modelling”, *5th International Modelica Conference 2006*, 2006.
- [3] D. Simic, H. Giuliani, C. Kral, and F. Pirker, “Simulation of conventional and hybrid vehicle including auxiliaries with respect to fuel consumption and exhaust emissions”, *SAE World Congress 2006*, April 2006.
- [4] C. Kral and A. Haumer, “Modelica libraries for dc machines, three phase and polyphase machines”, *Modelica Conference*, pp. 549–558, 2005.
- [5] H. Giuliani, C. Kral, J.V. Gragger, and F. Pirker, “Modelica simulation of electric drives for vehicular applications . the smart drives library”, *ASIM*, 2005.
- [6] Y. Zhu, Y. Chen, and Q. Chen, “Analysis and design of an optimal energy management and control system for hybrid electric vehicles”, *Electric Vehicle Symposium, EVS19, Busan, Corea, October 2002*, October 2002.
- [7] W. Beitz and K.H. Küttner, *Dubbel Taschenbuch für den Maschinenbau*, Springer Verlag, Berlin, 14 edition, 1981.
- [8] G.C. Kim and Y.J. Lee, “The influence of driving conditions on fuel economy and exhaust emissions of the toyota PRIUS HEV”, *The 20th International Electric Vehicle Symposium and Exposition*, 2003.
- [9] National Renewable Energy Laboratory (NREL), “Advisor documentation, advisor data file fc prius jpn”, www.ctts.nrel.gov, 2002.
- [10] Toyota Motor Corporation, *Prius II Service Manual*, Toyota Motor Corporation, 2003.

Coordinated automotive libraries for vehicle system modelling

Mike Dempsey¹, Magnus Gäfvert², Peter Harman³,
Christian Kral⁴, Martin Otter⁵, Peter Treffinger⁶

¹ Claytex Services Limited, Hatton, UK

² Modelon AB, Lund, Sweden

³ Ricardo UK Limited, Leamington Spa, UK

⁴ arsenal research, Vienna, Austria

⁵ DLR, Oberpfaffenhofen, Germany

⁶ DLR, Stuttgart, Germany

vi@claytex.com

Abstract

A new free Modelica library, called VehicleInterfaces, has been developed to promote compatibility between Modelica automotive libraries. The library provides standard system interface definitions that enable the whole vehicle system to be conveniently modelled. Example vehicle models are also provided to illustrate the use of the interface definitions. Practical applications and examples of how these interface definitions can be used are presented.

1 Motivation

A number of Modelica library developers are working independently on automotive libraries focused on different vehicle systems such as PowerTrain [1], Transmission [2], VehicleDynamics [3] and SmartElectricDrives [4]. For many simulation activities it is desirable to be able to create whole system models [5] that combine elements from the different libraries

and provide easy ways to integrate user-developed models.

This work is based on previous work carried out by the authors and also builds on the work done on the Modelica Vehicle Model Architecture published by Tiller et. al.[6].

The approach adopted in developing this library has been to focus on standardising the subsystem interfaces rather than developing a standard vehicle model architecture. Several different example architectures based on these interfaces are provided as examples in the VehicleInterfaces library. A typical example is shown in Figure 1 where all the main subsystems (driver, driverEnvironment, engine, transmission, driveline, chassis, brakes, accessories, road, atmosphere, etc.) are included at the top level of the model. In this example it is assumed that the controllers for the respective subsystems are part of the subsystem model. In other architectures, the controllers might be on the same level as the subsystems.

All subsystem models in the architecture are de-

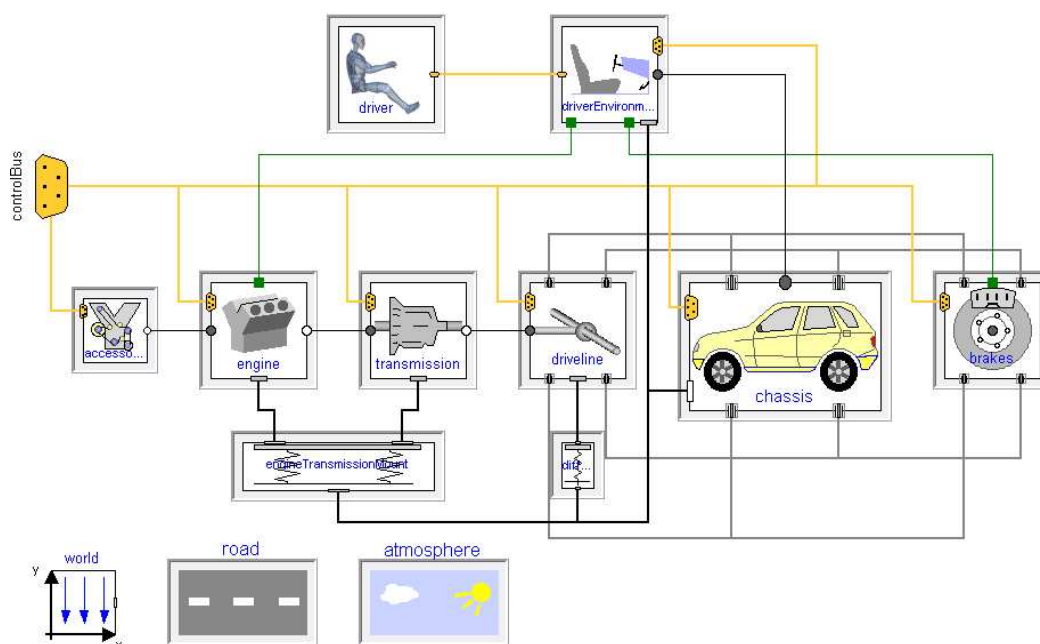


Figure 1: Example vehicle model using VehicleInterfaces shown in Dymola 6.0b

clared as “replaceable”. When instantiating an architecture model, the desired subsystem model can be selected via a redeclaration. In Dymola there are two ways to redeclare components using the graphical user interface as shown in Figure 2.

The dialog box shown at the top of Figure 2 is accessed through the model browser by right-clicking on the base class being extended. The scroll down menus show the available subsystem models that can be selected. As usual, these menus are automatically constructed via the annotation “choicesAllMatching = true” (all model components are shown that are loaded into the Modelica simulation environment and are derived by inheritance from the respective superclass).

In Dymola 6 redeclarations can be made by right clicking on a replaceable subsystem, the context menu then allows the user to select an option from the list of matching types. The list is determined

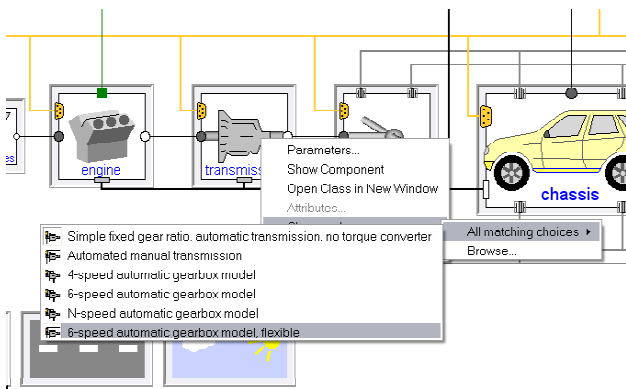
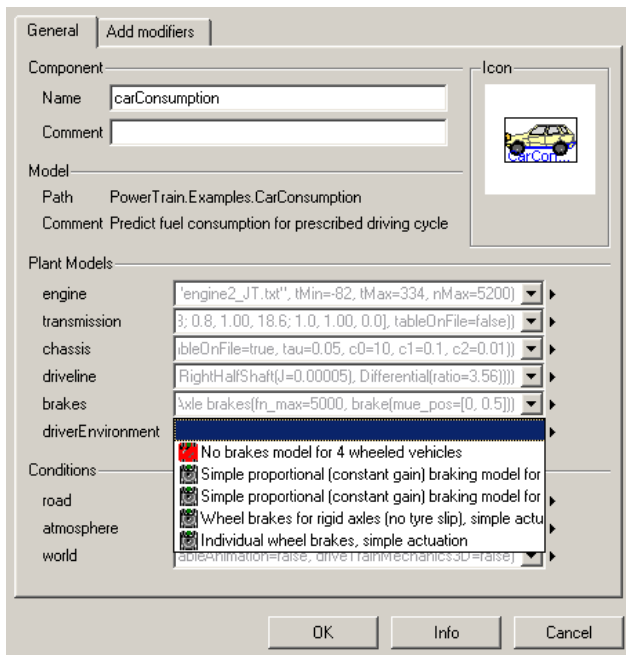


Figure 2: Two methods for the redeclaration of subsystem models in Dymola 6

using the same annotation as the dialog box method.

2 Interface Definitions

The subsystem decomposition used within the VehicleInterfaces package follows the same basic structure as used in the Modelica VMA developed by Tiller et al [6]. In the following sections we will discuss the subsystems and modelling methods introduced with the VehicleInterfaces package.

2.1 Conditional Connectors

The VehicleInterfaces package makes extensive use of conditional connectors so that a single subsystem interface definition can be used for a wide range of applications. A conditional component, such as a connector, is one that is only instantiated in the model if the corresponding Boolean condition is true. The Modelica code for an example conditional component is shown in Figure 3. In this example the component **shaft** is only instantiated in the model if the user sets the parameter **includeInertia=true**, if the parameter is false then the component and all related connections are completely removed from the model.

If we consider the engine subsystem, this includes conditional connectors for the engineMount and acceleratorPedal connections. The engineMount connector models the physical connection between the engine block and the engine mounting system as a MultiBody connection. This is not always required, for example when modelling the engine as a simple 1D system. Similarly the acceleratorPedal, which is a 1D translational connector provides a physical connection between the engine subsystem and the driver environment, which isn’t required in a drive-by-wire vehicle.

By using conditional connectors these components are completely eliminated from a model when they are not required. If they were not eliminated it would be necessary to add additional components to ensure that all the flow variables within the connectors were being properly defined. This would add additional overhead to the simulation task and should

```

model Example
  parameter Boolean includeInertia=false;
  Rotational.Inertia shaft if includeInertia;
  ...
end Example;
    
```

Figure 3: Example of a conditionally instantiated Inertia component.

be avoided whenever possible.

Conditional connectors are used wherever a connector might not be needed in every application. These include all MultiBody connectors and all the physical connections between the driver environment and the subsystem models.

2.2 Modelling Rotating Components

Within VehicleInterfaces rather than limiting ourselves to modelling rotating components as a simple 1D rotation we have built in the flexibility to model rotating components as MultiBody systems. This has been possible through the development of a new connector called FlangeWithBearing. This is a hierarchical connector that contains a 1D rotational connector and a conditional MultiBody connector. The use of the MultiBody connector is controlled by a parameter in the connector. This enables the FlangeWithBearing connector to model rotational effects as a purely 1D system or it can correctly include MultiBody effects. The Modelica definition of this new connector is shown in Figure 4 and the connector is now available within the MultiBody.Interfaces package of the Modelica Standard Library 2.2.1.

Where this connector is used in an interface definition the parameter **includeBearingConnector** is linked to a parameter at the model level so that the model developer can easily activate this connector if required. An example of how this is used is shown in Figure 5 where the driveline interface definitions for a 2-axle vehicle are shown. In the Base class we can see that three Boolean parameters are declared as protected parameters. This means they are only available to the model developer as they create the subsystem model and cannot be changed when the model is used.

The use of the bearing connectors needs to be carefully considered to avoid inadvertently creating mechanical loops in the model. The following example guidelines for the engine and transmission subsystems should help highlight the issues so that the

```
connector FlangeWithBearing
  parameter Boolean
    includeBearingConnector=false;
  Rotational.Interfaces.Flange_a flange;
  MultiBody.Interfaces.Frame bearingFrame
  if IncludeBearingConnector;
end FlangeWithBearing;
```

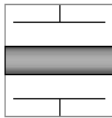


Figure 4: Modelica code and icon for the FlangeWithBearing connector

```
model Base
  MultiBody.Interfaces.FlangeWithBearing
  TransmissionFlange(
    final includeBearingConnector=
      includeTransmissionBearing);
  VehicleInterfaces.Interfaces.ControlBus
  ControlBus
  MultiBody.Interfaces.Frame_a drivelineMount
  if includeMount;
protected
  parameter Boolean
    includeWheelBearings=false;
  parameter Boolean includeMount=false;
  parameter Boolean
    includeTransmissionBearing=false;
end Base;

model TwoAxleBase
  extends Base;
  MultiBody.Interfaces.FlangeWithBearing
  WheelHub_1(
    final includeBearingConnector=
      includeWheelBearings);
  MultiBody.Interfaces.FlangeWithBearing
  WheelHub_2(
    final includeBearingConnector=
      includeWheelBearings);
  MultiBody.Interfaces.FlangeWithBearing
  WheelHub_3(
    final includeBearingConnector=
      includeWheelBearings);
  MultiBody.Interfaces.FlangeWithBearing
  WheelHub_4(
    final includeBearingConnector=
      includeWheelBearings);
end TwoAxleBase;
```

Figure 5: Modelica code for the driveline interface definition

model developer can determine when it is appropriate to use the bearing connectors. For the engine and transmission subsystem models:

1. When they are modelled as a pure 1D rotational system then no bearing connectors are required.
2. When they are modelled as a 1D rotational system with reactions on to a MultiBody system then no bearing connectors are required
3. When they are being modelled as a MultiBody system but they are rigidly connected together then the bearing frame between the engine and transmission should not be included. In this case the transmissionMount connector should support the MultiBody elements of the transmission. The rest of the model then needs to be considered before deciding whether to include the bearing between the transmission and driveline or between the engine and accessories subsystems.
4. When they are being modelled as a MultiBody system but they are **not** rigidly connected to-

gether then the bearing frame between the engine and transmission will be required to support the intermediate drive shaft.

2.3 Driver and Driver Environment Subsystems

The driver and the physical interaction between driver and vehicle can be modelled as separate subsystems or combined as a single subsystem within the `VehicleInterfaces` package. The decision is down to the model developers and will influence the extent to which models are reusable in other applications. The physical interactions, such as steering and throttle controls and feedback signals, are referred to as the driver environment within `VehicleInterfaces`.

When the driver and driver environment are modelled as separate subsystems the driver environment model is responsible for converting the normalised instructions passed from the driver model in to the correct values for this particular vehicle model. For example, a driver model should demand normalised steering wheel angles between -1 and 1 and this should be translated by the driver environment subsystem in to the appropriate steering wheel angle for the current vehicle. This enables the driver model to be defined in a generic way for use on many different vehicles.

The interaction between the driver and driver environment subsystem is modelled using an expandable connector known as the driver interaction bus. The connections passed across this bus are a combination of signal values and normalised physical connections. A normalised physical connection contains both the normalised position and the actual force or torque being applied across the connection. The naming and types for the signals that are exchanged between these two subsystems is defined within the `VehicleInterfaces` package to ensure compatibility between driver models and driver environment models from different libraries.

When only the driver environment subsystem is present this should also include the driver model and it is the responsibility of the individual model developers to provide a logical separation between the environment and driver models.

2.4 Powerplant Mounts Subsystem

Unlike in the Modelica VMA the Powertrain mounting systems are modelled as separate subsystems when they are required in a model. In Figure 1 we see that there are two separate mounting systems, one for supporting the engine and transmission and

another that supports the differential. The modelling of the mounting systems in this way reflects the physical reality of a rear-wheel drive vehicle in which the engine and transmission are rigidly connected together and mounted as one system at the front of the vehicle and the differential in the rear axle is independently mounted. Vehicles with different driveline configurations would require a different arrangement for the mounting systems.

The mounting subsystems are all defined by extending a base class that includes a `MultiBody` connector that should be connected to the vehicle body. There are 3 mounting subsystem templates provided within the `VehicleInterfaces` package that can be used to support differing numbers of powertrain subsystems. The connections to the powertrain subsystems are modelled using `MultiBody` connectors.

When the mounting systems are not being modelled these subsystems can be removed from the model architecture to simplify the vehicle model.

2.5 Road Subsystem

The road subsystem is used to define the road surface and supports varying friction coefficients, curvature, gradients and banking. The road is defined as a series of replaceable functions that are used to determine the position along the road, the normal to the road surface, the current heading of the road centre line and the friction coefficient. By redeclaration of these functions a wide range of road models from a straight flat road through to a curved undulating road can be created.

When a road is used at the top level of a model it should be declared with the prefix `inner` so that it can be referenced from any subsystem or component within the model that needs to determine information about the road surface. When a subsystem needs to refer to the road subsystem it should contain an outer version of the road subsystem and this will then enable it to access the road definition from the top-level of the model.

2.6 Atmosphere Subsystem

The atmosphere subsystem defines the ambient conditions including temperature, pressure, humidity, wind speed and direction. The atmosphere is defined as a series of replaceable functions that determine these conditions at a specified point in space. This enables the ambient conditions to vary with the vehicle position so effects such as wind can be varied as the vehicle drives along a track.

2.7 Example Vehicle Architectures

Using these interface definitions we can create a variety of vehicle model architectures to suit different applications. Figure 1 shows an example architecture for a rear-wheel drive automatic transmission passenger car that includes a separate driver model and powertrain mounting systems. Figure 6 shows some other possible model architectures including (from top to bottom) a manual transmission vehicle; an alternative layout for an automatic transmission vehicle; a power-split hybrid vehicle model. In all these cases it is possible to re-use the same subsystem models because the interface definitions are consistent even though the top-level model appears very different.

3 Control Bus Structure

3.1 Overview

Within the VehicleInterfaces library every subsystem that forms part of the vehicle model has a connection to the control bus. The control bus is used to pass information between the subsystems that would normally be passed along the CAN bus or similar vehicle communication network. The VehicleInterfaces control bus does not model how the vehicle network communication actually works but instead provides a structure by which the same information can be exchanged between the various subsystems.

The control bus is modelled using a series of hierarchical expandable connectors, which means that the user can place any signal they need on to the control bus. As part of the VehicleInterfaces library a minimum set of signals and a structure for the control bus is recommended so that systems that follow these recommendations can easily be coupled together.

A hierarchical structure to the control bus is proposed where the subsystem name is used to help structure the signals on the bus. For example signals placed on to the control bus from the chassis subsystem should be placed within the chassisBus structure of the controlBus, see Figure 7 for an illustration of the current minimum set of signals for the control bus. A full naming convention is included with the VehicleInterfaces package.

3.2 Working with the Control Bus

Every subsystem within the VehicleInterfaces package contains a controlBus connector that will allow the subsystem access to the complete control bus

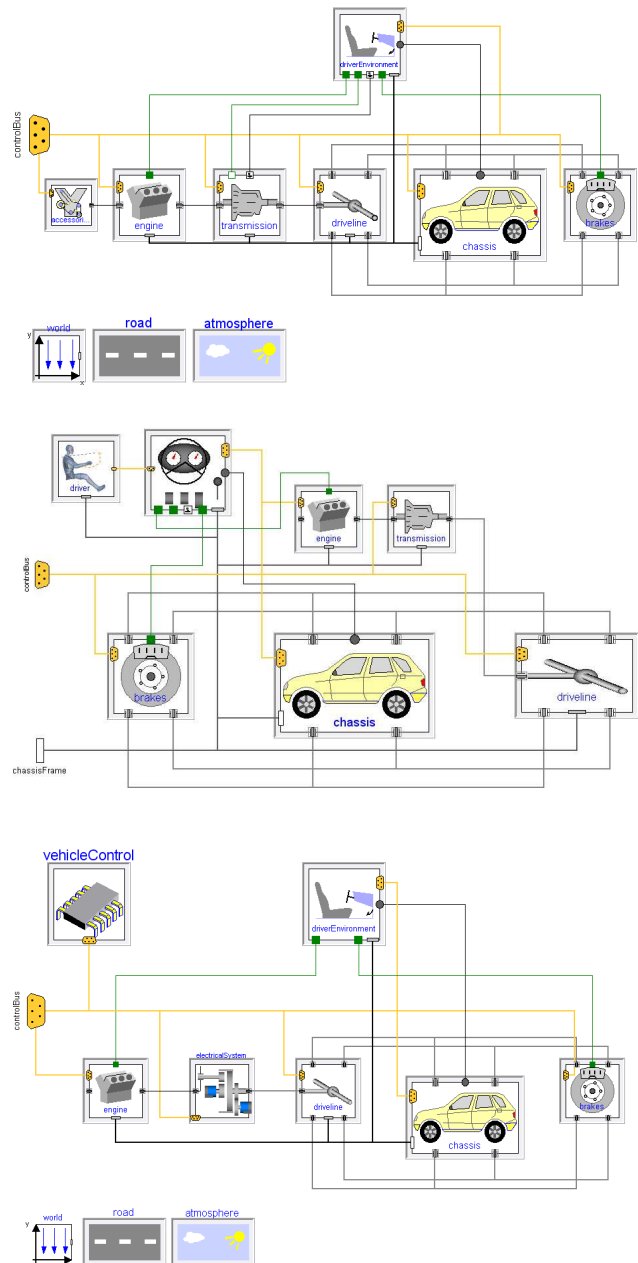


Figure 6: Example model architectures

structure. To access a signal within the control bus hierarchy it is first necessary to add the appropriate sub-bus connector to the model as a node, i.e. a protected connector. Signals within this part of the control hierarchy can then be accessed by connecting to the sub-bus connector. Figure 8 shows how the longitudinal velocity signal within the chassisBus sub-bus on the vehicle controlBus can be accessed. The Modelica code for this example is also shown. This methodology is necessary due to the way the Modelica language specification defines expandable connectors [7].

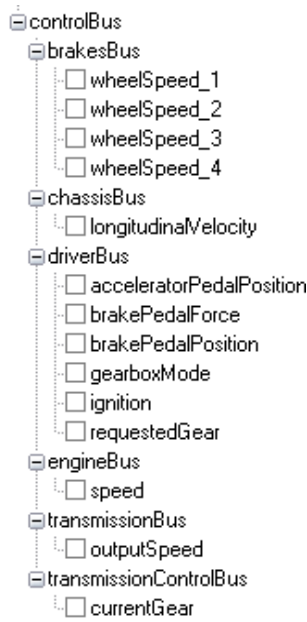
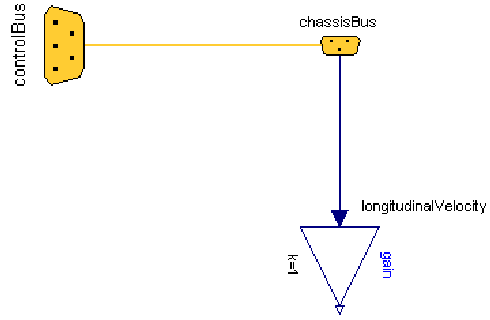


Figure 7: Current minimum set of signals in the control bus



```

model controlBusDemo
  extends TwoAxleBase;
  protected
    VehicleInterfaces.Interfaces.ChassisBus
      chassisBus;
  public
    Modelica.Blocks.Math.Gain gain;
  equation
    connect(chassisBus,
            chassisBus);
    connect(gain.u,
            chassisBus.longitudinalVelocity);
end controlBusDemo;

```

Figure 8: Accessing signals on the control bus (model diagram in Dymola)

3.3 The Sub-bus Connectors

The sub-bus connectors are defined within the VehicleInterfaces.Interfaces package and are all defined as expandable connectors. As such the connectors don't contain any signal names and yet we would like it to be possible to generate a list of pre-defined names to make it easier for the user to connect to the control bus and access the appropriate signal.

To enable this to happen, within the VehicleInterfaces package each expandable connector has been extended and the standard signals have been defined within these extended connectors. These connectors are not intended for use directly within a model but are necessary to enable a Modelica tool to generate a list of possible signal names. Within Dymola, when a connection to an expandable connector is made a dialog box is generated with a list of signal names. The list of signal names is now determined by searching through the open libraries to find connectors that extend from the type of expandable connector used in the current connection. All the signals that are defined within the connectors that extend from the base connector are then added to the list and the user can select the appropriate one.

This functionality means that the user can easily connect to one of the standard signal names but also means that it is not absolutely necessary for them to assign values to every signal that is defined as part of the standard VehicleInterfaces control bus. It also allows different library developers to extend the con-

trol signal bus in appropriate ways for their library and to have the names automatically appear in models.

4 Usage Examples

Three different usage examples are presented to illustrate different ways that the interface templates can be used to model vehicles with different levels of detail. The first two examples illustrate two different approaches to modelling a rear-wheel drive driveline and the third example illustrates how the different commercial model libraries that are adopting these interface definitions can be coupled together.

4.1 Rear-Wheel-Drive Vehicle as a 1D System

The simplest way to model a vehicle powertrain is as a 1D rotational system. This approach does have many uses, such as fuel economy studies, and this example illustrates how the interface templates can be used in this way. Figure 9 shows the driveline model diagram for a rear-wheel drive vehicle modelled as a purely 1D rotational system. Components in this model are taken from the Modelica Standard Library and the PowerTrain library.

In this example the parameters that control the conditional connectors for the driveline interface class are all left with their default values of false. This means

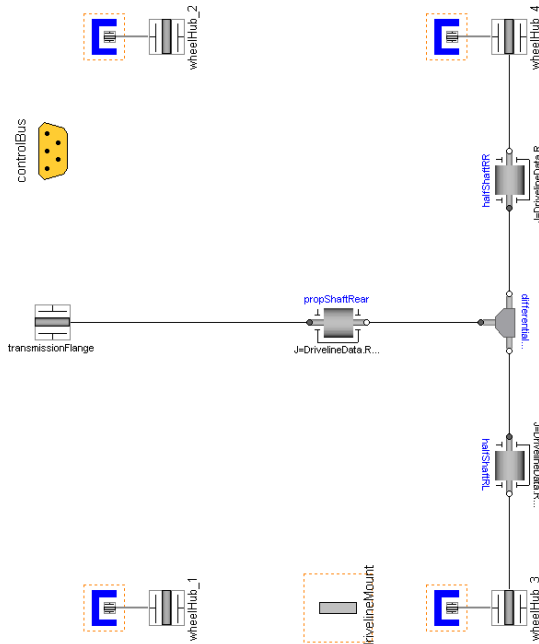


Figure 9: Simple 1D Rotational model of a rear-wheel drive driveline

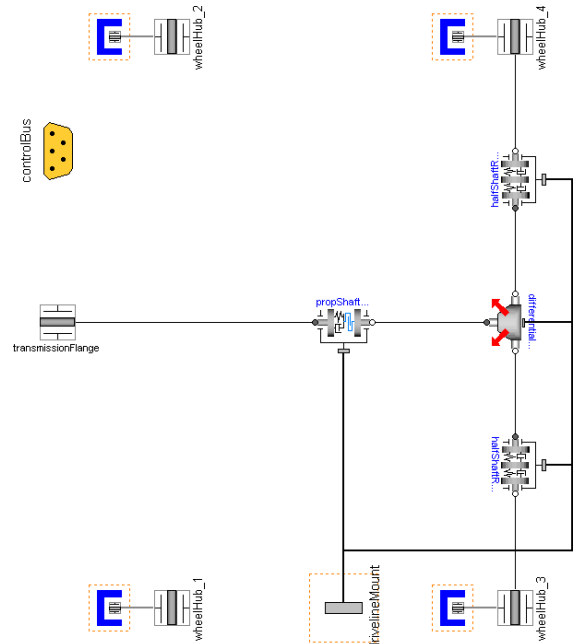


Figure 10: 1D Rotational model of a rear-wheel drive driveline with reactions on to a MultiBody system

that the bearing frame connectors within the FlangeWithBearing connectors (transmissionFlange, wheelHub_1, etc.) are not instantiated in the model and neither is the drivelineMount connector. This leaves us with a simple model using just the 1D rotational connectors for the transmission and wheel hubs.

When modelling a 1D rotational system it is sometimes necessary to include the reactions of the 1D rotational system on a MultiBody system [8]. Adapting the rear-wheel-drive model to include MultiBody effects would lead to the diagram in Figure 10. To enable this model to be built the drivelineMount connector needs to be enabled so that the MultiBody reactions can be transmitted in to the vehicle body. The bearing frame connectors within the transmissionFlange and wheelHub connectors are not required in this model as the driveline is not being modelled as a MultiBody system.

4.2 Rear-Wheel-Drive Vehicle as a MultiBody System

The same driveline interface template can be used to model the complete driveline as a MultiBody system. In this case the use of the bearing connectors within the FlangeWithBearing connectors needs to be thought about carefully in order to make sure mechanical loops aren't inadvertently created. Consideration needs to be given to the way in which the MultiBody components are being supported both in the physical system and in the model itself.

In the case of a rear-wheel drive vehicle the propshaft is supported by the transmission and the differential. So in this case the bearing frame in the transmissionFlange needs to be included so that this end of the propshaft is correctly supported. The differential is also being modelled as a MultiBody system so this will support the other end of the propshaft. The differential itself is typically supported by an elastic mounting system that would be connected

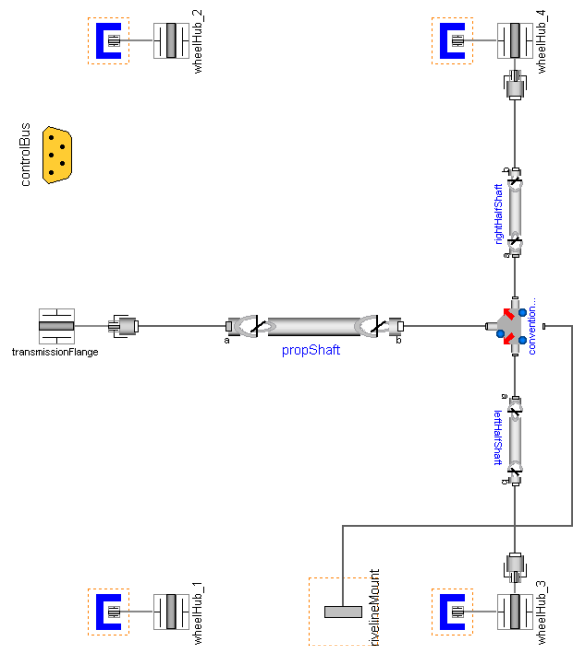


Figure 11: MultiBody model of a rear-wheel drive driveline

to the driveline model via the drivelineMount connector.

Finally it needs to be considered how the halfshafts are supported, one end is attached to the differential and supported by the output bearings of the differential, the other end is attached to the wheel hub and supported by the wheel bearing. This means the bearing frames in the wheel hub connectors need to be included. An example of how this subsystem might look is shown in Figure 11.

4.3 Active 4WD Vehicle Model

By combining models from the PowerTrain and VehicleDynamics libraries it is possible to study the handling benefits of active four-wheel-drive systems and compare this to the handling of the same vehicle with a conventional, passive four-wheel drive system.

The vehicle model is created using various subsystem models from the PowerTrain library and the VehicleDynamics library. The PowerTrain library contains an active four-wheel drive system model shown in Figure 12. The driveline is modelled as a 1D rotational system and includes the reactions on to the vehicle body. To make this 1D driveline model compatible with a MultiBody chassis model from the VehicleDynamics library we need to activate the flag **usingMultiBodyChassis** in the “Advanced” menu of the driveline component parameter dialog. When this

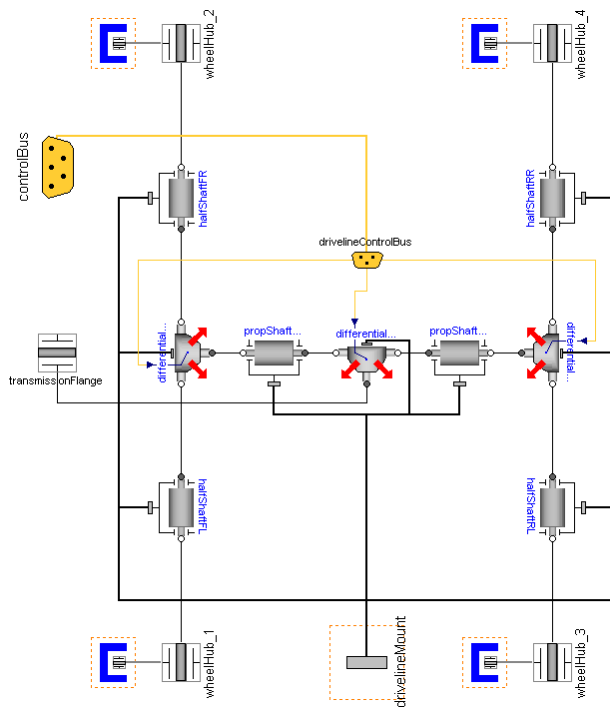


Figure 12: Active four-wheel drive system from the PowerTrain library

parameter is set to true the bearing connectors within the wheelHub connectors are included and zero forces and torques are applied to these bearing connectors.

The driveline control system model within the PowerTrain library provides parameters to enable or disable the control of the active differentials. When disabled the driveline behaves as a conventional, passive four-wheel-drive system so this model can easily be used to assess the benefits of active versus passive four-wheel-drive.

The vehicles are tested by accelerating from rest to 100kmh and then negotiating a tight chicane at 100kmh whilst trying to maintain this speed. Figure 13 shows how the yaw rate, steering angle and longitudinal speed of the two cars varies during the test. As a chassis model from the VehicleDynamics library is being used the behaviour of the cars during the test can be animated, Figure 14 shows a comparison of how the two cars behave.

5 Outlook

The first version (1.0) of the VehicleInterfaces library has been presented. Future developments and refinements will be based on feedback from automotive library developers and users of the VehicleInter-

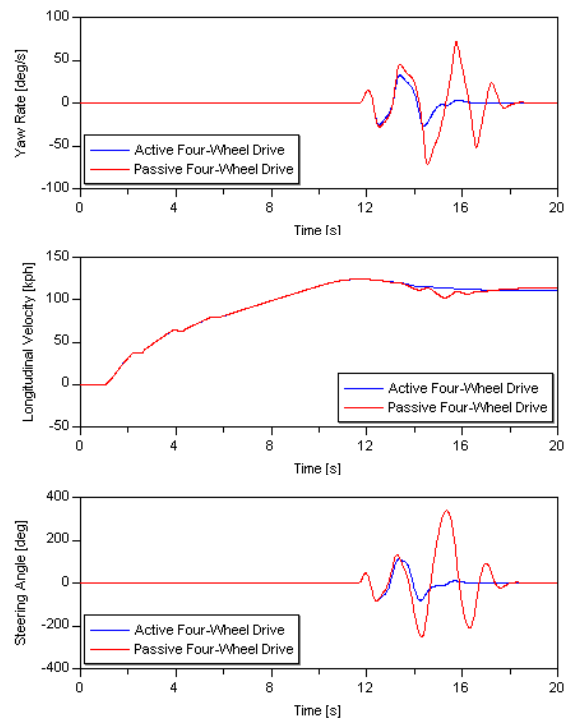


Figure 13: Comparing Active and Passive Four-Wheel Drive. Yaw-rate (top), Longitudinal velocity (middle) and Steering angle (bottom).

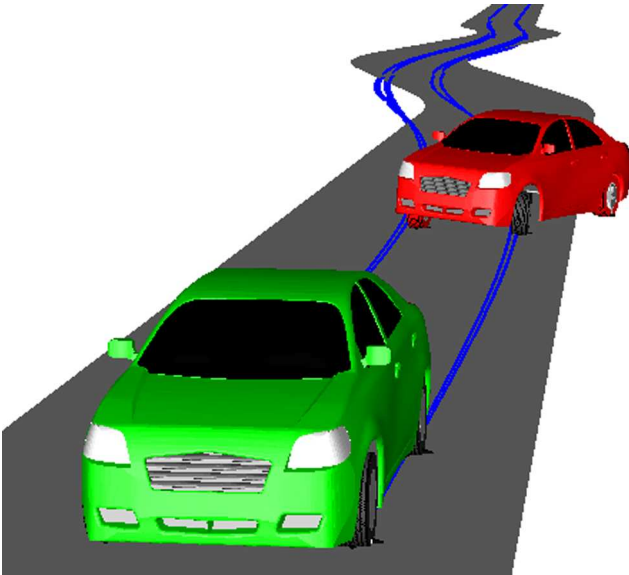


Figure 14: Visualising the behaviour of the two cars in Dymola. The green car has active four-wheel drive and the red car has passive four-wheel drive.

faces library. Currently only a small set of standardised signals have been defined on the control bus and it is likely that this will need to be extended significantly to meet the needs of users.

6 Acknowledgements

A number of automotive library developers and consultants have co-operated to develop this release of the VehicleInterfaces Library. The developers can all be contacted by emailing vi@claytex.com. In addition to the authors of this paper the following people have also contributed:

Arsenal Research: Franz Pirker, Anton Haumer,

DLR Oberpfaffenhofen: Christian Schweiger, Jakub Tobolar.

DLR Stuttgart: Marcus Baur, Jörg Ungethüm

Modelon AB: Johan Andreasson

Ricardo UK Ltd: Mark Ingram

This library has been developed from work on the original Modelica VMA [6] developed by Michael Tiller et al. Additional ideas from intermediate work by members of DLR Oberpfaffenhofen and Modelon has also been incorporated.

Hilding Elmqvist from Dynasim AB is responsible for bringing this group of developers together with the objective of developing a standard automotive model architecture. Dynasim have funded much of the development of this library.

References

- [1] Schweiger C., Dempsey M., Otter M.: *The Power-Train Library: New Concepts and New Fields of Applications*. Proceedings of Modelica 2005 Conference. http://www.modelica.org/events/Conference2005/online_proceedings/Session6/Session6a1.pdf
- [2] Brandao F., Harman P.: *An Integrated Simulation Approach: Ricardo Transmission and Driveline Dynamic Simulation Library*. IMechE Integrated Powertrain and Driveline Systems 2006
- [3] Andreasson J., Gäfvert M.: *Vehicle Dynamics Library*. Proceedings of Modelica 2006 Conference.
- [4] Giuliani H., Kral C., Gragger J.V, Pirker F.: *Modelica Simulation of Electric Drives for Vehicular Applications – The Smart Drives Library*. ASIM conference, 2005
- [5] Alexander T., Liu C.S., Monkaba V.: *Multi-Body Dynamic Modeling Methods and Applications for Driveline Systems*. SAE 2002-01-1195
- [6] Tiller M., Bowles P., Dempsey M.: *Development of a Vehicle Modeling Architecture in Modelica*. Proceedings of the Modelica 2003 Conference. http://www.modelica.org/events/Conference2003/papers/h32_vehicle_Tiller.pdf
- [7] Modelica: Language Specification 2.2. Feb. 2005. Section 3.3.8, pp. 54 – 59 (expandable connectors). <http://www.modelica.org/documents/ModelicaSpec2.2.pdf>
- [8] Schweiger C., Otter M.: *Modelling 3D Mechanical Effects of 1D Powertrains*. Proceedings of Modelica 2003 Conference, Nov. 2003. http://www.modelica.org/events/Conference2003/papers/h06_Schweiger_powertrains_v5.pdf

The VehicleDynamics Library — Overview and Applications

J. Andreasson and M. Gäfvert

MODELON AB

{johan.andreasson,magnus.gafvert}@modelon.se

Abstract

The VehicleDynamics Library provides a foundation for model-based vehicle dynamics analysis, in particular related to road-vehicle handling. This commercial new library is a major improvement and expansion of the previously available free vehicle dynamics library. This paper gives an introduction and overview of the new library and gives some example use-cases.

1 Introduction

The potential of Modelica as a mean to efficiently describe vehicle models has been recognised for quite a while [1, 2, 3, 4, 5] and lead to the initiative to develop a library for vehicle dynamics studies [6]. This resulted in the free VehicleDynamics library that was developed and maintained by Johan Andreasson during 2000–2004, on the initiative and with support from Hilding Elmqvist (Dynasim AB) and Martin Otter (DLR Research Center Oberpfaffenhofen).

The great interest in the free VehicleDynamics library and increased requirements and demands on new features, continuity in maintenance and support, and not the least complete documentation triggered the decision to develop the library into a commercial product. This initiative was taken by MODELON AB in 2004, and has resulted in the commercial VehicleDynamics Library. The current version 1.0 of the library is a substantial extension and improvement of the now deprecated free VehicleDynamics library.

The VehicleDynamics Library, or VDL, is intended for vehicle dynamics analysis related to mechanical design and control design of automotive chassis. Handling behaviour is the primary target, but the library is also well suited for studies of other vehicle properties. The sequel will enlighten the contents and use of the library, pinpoint some key issues in the development and outline some expected enhancements.

2 Model and Library Structure

A road vehicle contains a multitude of parts and sub-systems. Conceptually, these can be organized in a hierarchy. For example, a car contains a chassis, that contains front and rear suspensions, that contains left and right suspension linkages and steering, etc. In analogy with this, it is natural to structure a model of a vehicle similarly and that is also reflected in how models in the VehicleDynamics Library and the library itself are organized as illustrated in figure 1 which contents are discussed more in section 3.

There are also vital models that are not reflected in neither of these illustrations that are gathered in special sub-packages throughout the package hierarchy:

Interfaces The Interfaces sub-packages contain common connector and parameter definitions and are used as base classes to ensure compatibility. Advanced users that define their own component models should use these classes, when applicable.



Templates The Templates sub-packages contains template classes that can be regarded as "wizards" to create more complex component models, vehicles, and experiments. By extending a template model and filling the placeholders with actual component models, new models can be implemented in an efficient way that is easy to maintain.



Components The Components sub-packages contain models that are used to build component models in the corresponding package. These models can be used by advanced users that build their own component models.



Internal The Internal packages contains models of no normal interest for the user, and these models should not be directly used by the user.



Experiments Models in VDL that are complete and meaningful to simulate are called *Experiments* and are indicated by a specific icon. Templates for experiments are located throughout the



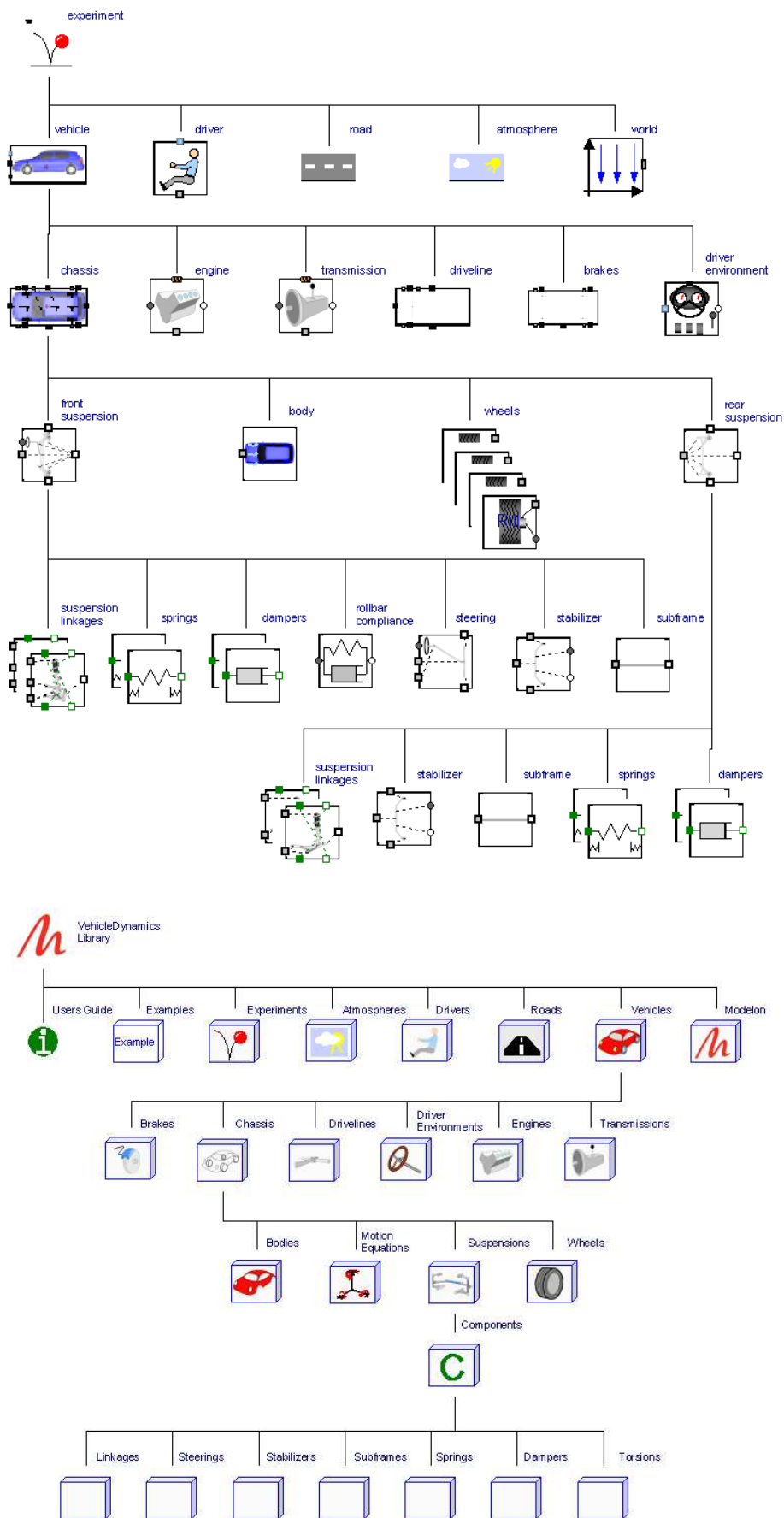



Figure 1: Hierarchy of an experiment model (top) and part of the hierarchical library structure (bottom).

library structure and include setups like full vehicle or chassis on a road, as well as test-rigs for analysing individual components or subsystems as discussed in section 5.

As for the sub-packages, there are also component types that are found throughout the library:

Base Base class for group of components in a sub-package. Located in Interfaces subpackage. Partial (incomplete) classes that must be extended before being instantiated.

Standard Base class for group of components in a subpackage. The Standard base classes differs from the Base base-classes in that they are less general. Standard base classes are designed to provide a standard connector interface that covers the most common variants for the component type. Standard base-classes extend from Base base-classes.

 **None** No (empty) implementation of a component that still simulates. Null components are mainly used in templates as a way to "leave out" components.

Typical Component that represents a typical variant.

Basic Basic model of a component.

Ideal Ideal model of a component.

3 Library Contents

3.1 Atmospheres

The Atmospheres package contains models of atmospheric conditions such as wind speed, humidity and density. This information is useful for analysis involving aerodynamic resistance or downforce, and responses to wind disturbances. It can also be used for other models that requires atmospheric information such as engine cooling.

3.2 Drivers

The Drivers package contains configurable open-loop, closed-loop, and event-driven driver models. The open-loop models has an extended set of sources to also be able to handle standard maneuvers such as instability triggering sine with dwell time or sine increasing amplitude. The closed-loop models use road information (see below) for positioning on and speed control. More complex sets of instructions are handled by the event driven driver model that allows changing between and combining open and closed loop tasks.

3.3 Roads and RoadBuilder

The Roads package contains road models and related components. The flat ground with optional inclination is useful for open-loop studies, while the versatile 3-dimensional table-based road model can be used with closed-loop driver models to analyze maneuver responses and driving on road-segments with user-defined geometry. The road models also contain reference trajectories for drivers without planning abilities. Obstacles such as cones and wind indicators can be used to make animations more illustrative.

The RoadBuilder sub-package contains utilities to create tables for the 3D-road model, either specialized for e.g. a double lane change or more generic, figure 2 illustrates typical input for the latter: Curvature is the inverse of the radius of the center line (1), road slope at the center line is the altitude increment along the road heading direction (2) and banking is the altitude increment perpendicular to the road center line (3). Road is the downwards-outwards increment from the center line which is present on roads to allow efficient water drainage (4). Left (5a) and right (5b) road edges are defined so that positive values defines the offset to the left of the center line. Note that the right edge always should have a greater value than the left edge. The start position (6) and heading angle (7) is the position and angle of the center line at zero distance, respectively. The track position (11) is defined as an offset from the road's center line that a driver model may follow by varying the steering input. If the offset is 0, the driver will follow the road center line and accordingly there is a velocity reference (11). The user can also specify the maximum error that is allowed due to linear approximation of a curve segment (8) and a maximum allowed distance between two consecutive grid points, this only occur for long straights (9). The resolution (10) defines the minimum distance between two grid points that RoadBuilder consider as separate when merging the input tables. Additionally there are cones that can be positioned along the road.

3.4 Vehicles

The Vehicles package contains models of whole vehicles and related components. Vehicle models are partitioned into the subsystems: chassis, driver, environment, engine, transmission, driveline, and brakes. The focus of VehicleDynamics Library is on chassis models, but there are also models of other subsystems such as driver, environments, engines, transmissions, drivelines, and brakes. The Vehicles package contains one sub-package for each subsystem.

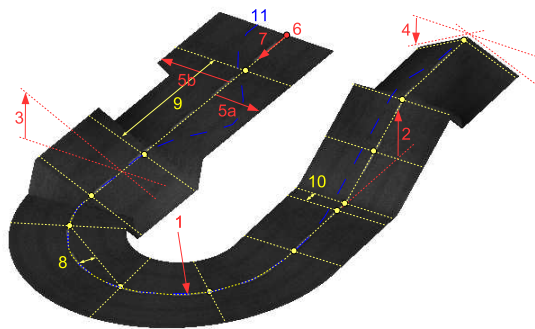


Figure 2: An example of possible RoadBuilder user inputs, note that the inputs in the example are exaggerated to make them clearer.

3.5 Chassis

The Chassis package is further partitioned into sub-packages for bodies, suspensions, wheels, etc. It contains a set of typical chassis configurations that represent standard cars like compacts, sedans, and SUV:s. In particular, there are models of different fidelities with the same interface and it is thus possible to use them with e.g. the same driveline and brake models. This minimizes modeling effort.

3.6 Suspensions

VDL includes a wide range of suspension models of both independent, semi-independent, and rigid axle types. The fidelity level spans from planar models with fixed roll and pitch centers, to table-based and geometric kinematic and elasto-kinematic models. As described in section 5, there are full-fledged kinematics and compliance (K&C) experiments of full suspensions as well as sub-components for analysis and model reduction.

3.7 Bodies

The Bodies package contains body models including inertial and aerodynamic properties as well as animation shape. Bodies can also conveniently be configured for different payloads.

3.8 Wheels and Tyres

The main content of this package is the tyre models. VDL includes a selection of well-known and commonly used tyre models for handling such as variants of Magic Formula, Pacejka models, Rill models, and semi-empirical models based on brush-model mechanics. An advantage is that the partitioning of the models

allow the same models throughout the chassis fidelity range which allow convenient configuration and comparison of results.

3.9 Sensors

Sensors can be freely located on the vehicle to be used as feedback signals for controllers or observers. VDL includes a set of sensors that are easy to configure and extend.

3.10 Drivelines and Brake Systems

The driveline and brake system models in VDL include 3D-position and orientation as described in section 4.1. Models of joints, shafts, differentials and more resolve driveline effects in a detail suitable for handling analysis. VDL also includes ideal and basic hydraulic brake systems and also allow inclusion of user-defined models based on any Mechanics interface from the Modelica Standard Library (MSL).

3.11 Engines and Transmissions

The engine package contains map-based and MVEM engine models and basic models of manual, automatic and CVT transmissions. Again, the adaption of a rotational interface with 3D capabilities, user defined models based on both the Rotational and the Multi-Body interfaces in MSL can be adapted.

3.12 The Modelon Package

The Modelon Library contains classes that complements the MSL. The structure of the library follows MSL whenever possible to facilitate navigation. Components that have a corresponding component in MSL are in general modified/improved versions but with the same basic functionality as the MSL version. The next section highlights some of the contents.

4 Development key issues

4.1 Rotational3D

There has been some development to describe the three-dimensional effects of components in the rotational library such as coriolis forces [7] but there was no satisfactory approach available for slightly more complex components such as shafts with different joint geometries. To overcome this, a package Rotational3D was introduced with a composite interface definition consisting of a 1D rotational flange resolved in a MultiBody frame. This allow the minor rotations of the axle mounts to be separated for the

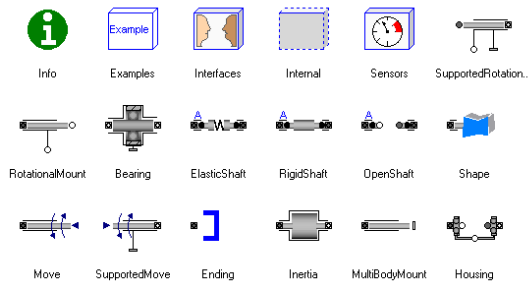


Figure 3: Rotational3D package contents.

axle spin which allows a consistent interface to both `MSL.Rotational` and `MSL.MultiBody` and a snapshot of the library contents is seen in figure 3.

4.2 State Selection in Suspensions

Unlike many other tools, Dymola use symbolic manipulation to resolve kinematic constraints which essentially mean that for a linkage model without included elasticities, the number of states correspond to the number of degrees of freedom that needs to be chosen with care. This is best illustrated with an example; consider the double wishbone linkage in figure 4. The leftmost version is completely rigid and the five links used constrain five of the original six degrees of freedom and there are thus two states that needs to be selected. If Dymola is able to select the states by its own, there are at least four combinations that could come up but only two of these turn out to be reasonable. Having the state in the unsuspended body is not reasonable for at least three reasons: First the numerical accuracy, if the vehicle is moved a considerable distance from its reference position, the resolution of the wheel travel would be low since it means subtracting two large numbers from each other. Secondly, since the motion of the upright would be defined relative to the world frame which would mean that if the vehicle would roll enough, the representation would be numerically tough and eventually singular. The third reason is common with having the state in the strut and allow multiple solutions according to figure 5. For some suspension linkages, multiple solutions could also occur when having the state selection in the joints but experience has shown that this is far easier to deal with simply by just supplying reasonable start values.

As seen from the discussion above, the ability to explicitly chose states is of significant important and it is not enough to be able to select all or no states in a joint as illustrated by the elasto-kinematic linkage in figure 4. The kinematic linkage needs two states to

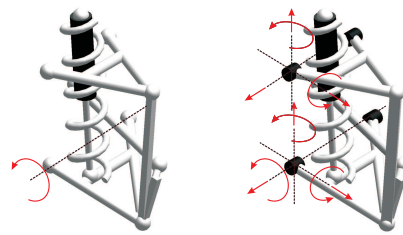
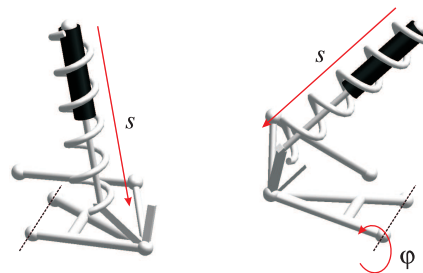


Figure 4: State selection for a kinematic (left) and an elasto-kinematic (right) double wishbone suspension linkage.

Figure 5: If s would be chosen as state, it is difficult to separate between wanted (left) and unwanted (right) configurations. Choosing φ instead makes the configurations unique.

specify the only degree of freedom which can be set in either of the joint pairs. Using bushings instead of inner joints gives ten additional degrees of freedom, i.e. eleven in total. However, since each of the pair of bushings have twelve potential states, 22 out of 24 have to be selected.

The standard `MultiBody` library has been designed with ease of use in mind [8] so the available models have the state selection lumped for all degrees of freedom which in the above case only would allow the user to specify 12 of the 22 states and leaving the rest for Dymola to figure out which works in some cases but not always. To overcome this issues, a set of joint models were developed with advanced users in mind, where each joint state candidate can be set independently of the others. Additionally, a body model without potential states was implemented, which enforces the possible states to be relative motion in joints. Also the quaternion representation was disabled since it require additional code and events for the dynamic state selection.

4.3 Signal Bus

Any complex modeling involving some kind of control sooner or later run into the need to structure informa-

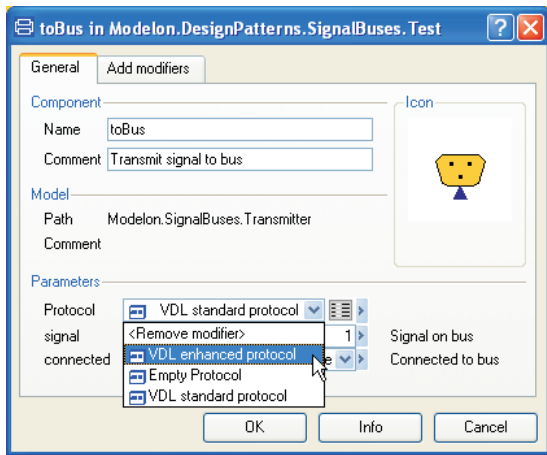


Figure 6: Choice of protocol in a transmitter.

tion exchange. More specifically, the following desires were identified: 1) Being able to predefine a set of possible signals on the bus. 2) Being able to replace this set. 3) Being able to have several sets at the same time. 4) Allowing efficient diagnosis for fault detection of connections. 5) Allowing GUI support from the tool. 6) Being able to model the dynamics of the actual bus (e.g. delay). 7) No need to draw extra connections to have components exchange signals with the bus.

The suggested implementation use the dynamic name lookup using the `inner` and `outer` constructs that allow coupling of models throughout the hierarchy. The actual bus definition is built around a base package definition containing a bus and a protocol and this package is declared as a `replaceable` type in the receiver and transmitter models.

By this construction, a user can define any own protocol by extending the base package definition and complement with the signal names that should be available. When connecting a component to the bus using a transmitter or a receiver component, the user can select what of the predefine protocols should be used and there is also enough information for a tool such as Dymola to actually display a list of the available signals in the selected protocol, see figure 6.

Despite any bus definition, the actual signal flow remain complex and being able to support the user in fault detection and debugging is crucial for efficient usage. Since a general Modelica model is free from causalities, it occurs as systems of equations and thus if a signal on the bus is not defined or defined more than once, the whole model will become singular and it is hard for a tool to give useable feedback on these types of errors.

The suggested construction allow multiple busses but require that there is only one set of signals per bus

which allow efficient diagnosis. By introducing a debug mode with a cardinality variable, the bus model can ensure that one and only one transmitter is connected to each signal and also replaced undefined signals with dummies. In this way warnings can be given if two or more transmitters try to send the same signal to the bus, if a receiver tries to access a variable that is not sent by a transmitter etc.

5 Library Usage

This section highlights some issues relating to the library usage.

5.1 VDWorkbench

VDWorkbench is a user-writable area that is installed together with the VehicleDynamics Library. It is opened from the **File|Libraries** menu. In this area users can work with examples and store their own models and experiments. The structure of VDWorkbench is laid out to mimic the one in VehicleDynamics and users that produce a lot of custom models and components are encouraged to continue this.

5.2 Summary Records

Many standard components contain a `Summary` record, indicated with an icon in the diagram layer. There are different summary records for different classes of components. The summary record contains variables that are of general interest for post-simulation analysis and plotting. For example, the summary record for a standard engine contains the variables for engine speed, torque, and power.

Summary records are convenient to use in the Variable Browser in the Simulation tab. By clicking the **Advanced** button and entering "summary" in the search filter the variable list is restricted to summary records as illustrated in 7.

5.3 Custom Vehicles

The library provides templates for building models of standard cars. Non-standard vehicles are built either directly from the component base classes. It is also possible to define custom templates for non-standard vehicles. The library contains a large number of components that can be used in the templates and it possible for users to define their own components to use in templates. For example, parts like A-arms, bushings, struts, links, gears etc. are available in the Suspensions package for users that defines their own suspensions. Figure 8 shows the the implementation of a



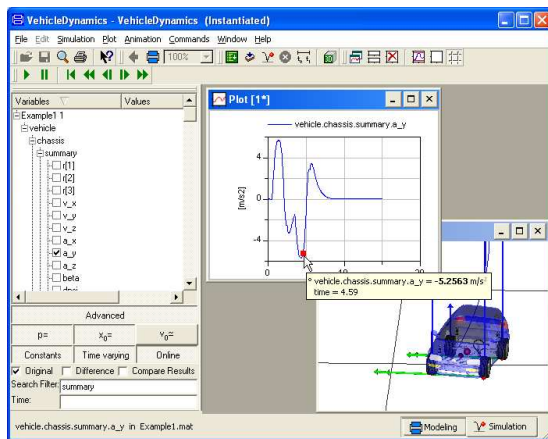


Figure 7: An example of using a summary record.

vehicle with *autonomous corner modules* (ACM), [9], where each wheel is steered, cambered, suspended and driven/braked individually.

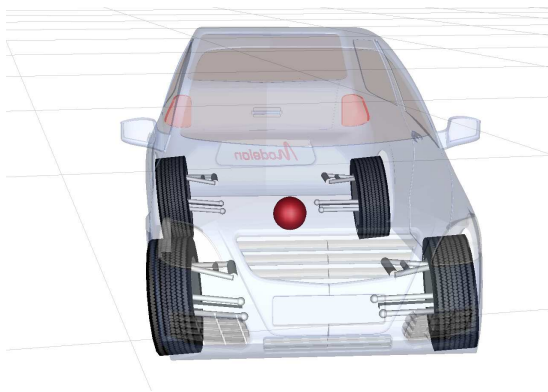


Figure 8: Custom vehicle design exemplified with the ACM concept.

5.4 Migration

The ParsfileConverter utility is used for migrating models from the Parsfile format used by CarSim into VehicleDynamics Library. The utility can handle hierarchical Parsfiles that are split into multiple files as well as flat single-file Parsfiles. The ParsfileConverter is a stand-alone executable that is supplied with VDL.

5.5 Vehicle Control

A major area of application for VDL is vehicle control. This example illustrates how active stabilizers that can be used to enhance stability are incorporated in a VehicleDynamics model. Figure 9 shows the response of an evasive lane change performed by a closed loop drivers for two full loaded vehicles, with and without roll control.

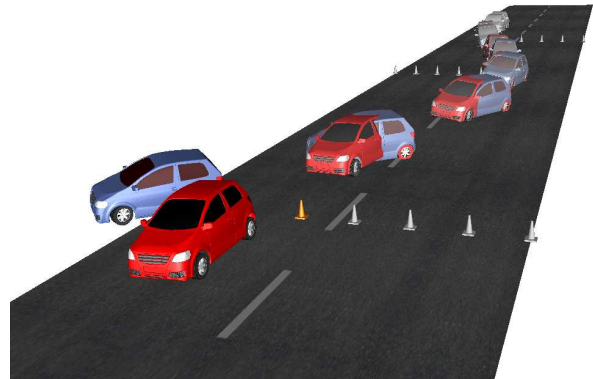


Figure 9: Active stabilizers can be used to prevent roll-induced instability.

5.6 Component Analysis

A vital part in any system design is a thorough understanding of the involved subsystems and an analysis of a full vehicle is often useless if the behavior of involved subsystems is not sufficiently understood. To isolate component behavior and to avoid unnecessary computational cost, test rig experiments are great aids. In VDL they are designed, parameterized and animated as real test rigs where applicable. To a test rig, various controller can be attached to govern the experiment and facilitate the test procedure. Figure 10 illustrates a typical K&C experiment with the corresponding user settings.

5.7 Wheels and Tyres

The tyres are critical components to determine the response of a road vehicle to driver inputs. Tyres are also famously difficult to model and calibrate with experimental data. VDL include several of the most well-known and established tyre models for handling studies. It is important to understand the differences between the different models, and to have full insight into the properties of different data-sets for the model parameterizations in order to correctly interpret experiment results on chassis. To support the users in this, VDL includes versatile test rigs where wheels and tyres can be studied in isolation. The test rig in figure 11 can be used to study quasi-static and dynamic properties under combinations of sweeps and constant levels of slip-angle, slip ratio, camber, load, and velocity.

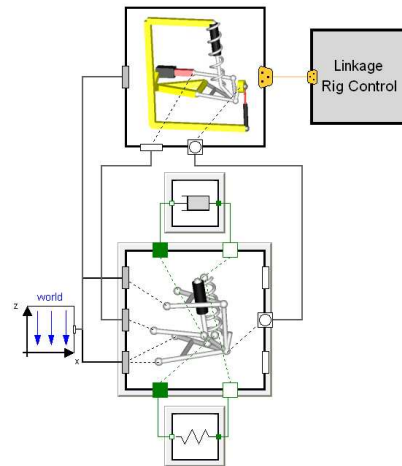
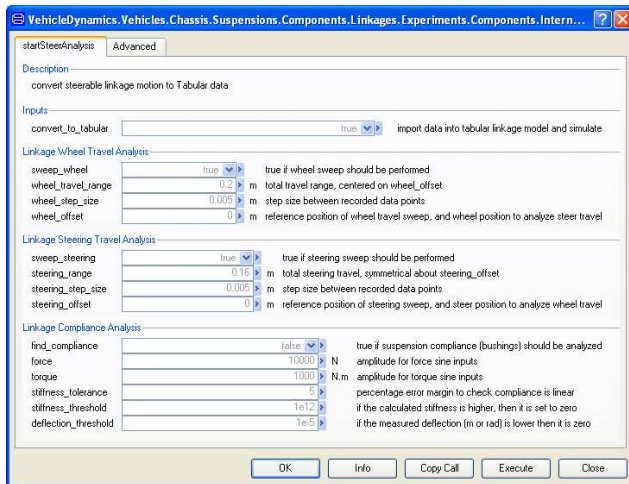


Figure 10: K&C experiment (right) with user settings (left).

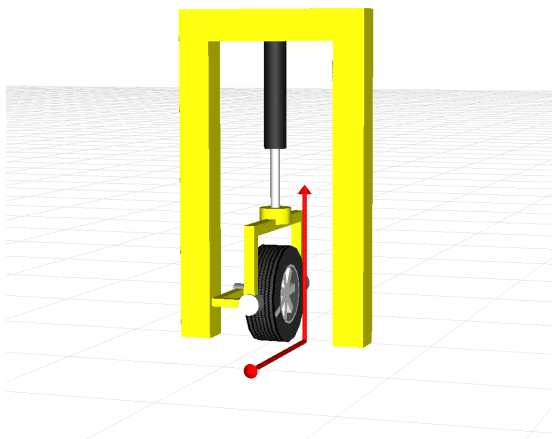


Figure 11: Tyre test rig.

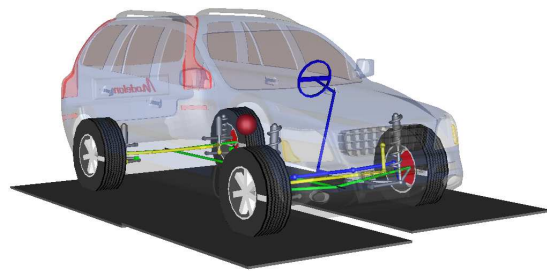


Figure 12: Vehicle mounted in a test rig.

5.8 Vehicle Analysis

Vehicle analysis can be performed both on road using a driver or robot to control the vehicle which allow simulations to mimic real test procedures to generate e.g. cornering characteristics diagram. Additionally, there are also ideal models of e.g. drivelines that allow precise wheel spin velocities or torques to be applied. Additionally, just as for a component, the vehicle can be constrained in different test rigs to generate various kinds of measures that requires extensive laboratory equipment to be performed in reality.

In figure 12 such an experiment is exemplified using a shaker rig. The model allow the level under each wheel to be adjusted individually and depending on the input used, this can be used to analyze chassis out-of-plane characteristics such as roll and pitch dynam-

ics, spring and damper settings and roll moment distributions. Additionally, it can for example be used to analyze driveline motion and study joint angles, shaft lengths, and other variables during various suspension-travel scenarios to verify or tune the geometric design.

6 Ongoing and Future Extensions

This section enlightens some of the extensions of the VDL that are ongoing and/or under consideration. Two other Modelon libraries, PneuLib and HyLib, opens up for incorporation of more detailed hydraulics and pneumatic models. Hydraulics models are relevant for especially brake systems but also other hydraulic systems found in e.g. power steering. Pneumatics is highly relevant for the ongoing enhancement to also consider heavy vehicles, figure 13, both for brake systems and air spring suspensions. For this extension, flexible elements are of increased impor-



Figure 13: Tractor with semitrailer negotiating a turn.

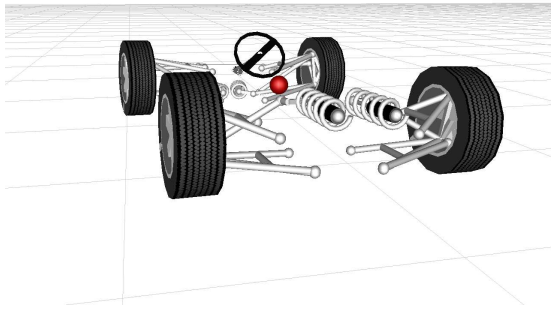


Figure 14: Cornering characteristics test of a Formula SAE chassis.

tance and this is discussed in more detail in [10].

A racing extension is also considered, requiring mainly different types of suspensions and analyzes, figure 14 shows a test run of a Formula SAE chassis.

Acknowledgements

The development of the VehicleDynamics library has included the generous support and encouragement of many people and organizations. Especially, the authors would like to thank:

- DLR, Oberpfaffenhofen, main author of the Modelica MultiBody library, in particular Prof. Martin Otter, for support and cooperation in realizing a Modelica vehicle dynamics library
- Royal Institute of Technology, Division of Vehicle Dynamics, Stockholm, Sweden, for generous support in the realization of the library

- Dynasim AB, for extensive support and enhancements of Dymola to form an optimal platform for vehicle dynamics modeling and simulation. Particular thanks to President Dr. Hilding Elmquist.

References

- [1] J. Andreasson, A. Möller, and M. Otter. Modeling of a racing car with Modelicas MultiBody library. In *Proc. of the 1st Int. Modelica Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [2] S. Drogies and M. Bauer. Modeling Road Vehicle Dynamics with Modelica. In *Proc. of the 1st Int. Modelica Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [3] B. Jacobson et al. Modelica usage in automotive problems at Chalmers. In *Proc. of the 1st Int. Modelica Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [4] M. Tiller et al. Detailed vehicle powertrain modeling in Modelica. In *Proc. of the 1st Int. International Modelica Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [5] M. Dempsey et al. Coordinated automotive libraries for vehicle system modelling. In *Proc. of the 5th Int. Modelica Conf.*, Wien, September 2006. The Modelica Association and arsenal research.
- [6] J. Andreasson. VehicleDynamics library. In *Proc. of the 3rd Int. Modelica Conf.*, Linköping, November 2003. The Modelica Association and Linköping University.
- [7] C. Schweiger and M. Otter. Modelling 3D Mechanical Effects of 1D Powertrains. In *Proc. of the 3rd Int. Modelica Conf.*, Linköping, November 2003. The Modelica Association and Linköping University.
- [8] M. Otter, H. Elmquist, and S.E. Mattson. The new Modelica MultiBody library. In *Proc. of the 3rd Int. Modelica Conf.*, Linköping, November 2003. The Modelica Association and Linköping University.
- [9] S. Zetterström. Electromechanical steering, suspension, drive and brake modules. In *Proc. of 56th Vehicular Technology Conference*, volume 3, pages 1856 – 1863. IEEE, 09 2002.
- [10] N. Philipson. Leaf spring modeling. In *Proc. of the 5th Int. Modelica Conf.*, Wien, September 2006. The Modelica Association and arsenal research.

Session 1c

Language, Tools and Algorithms 1

Modelica CDV

A Tool for Visualizing the Structure of Modelica Libraries

Martin Loeffler¹, Michaela Huhn¹, Christoph Richter², Roland Kossel³

¹Technical University of Braunschweig, Institute for Programming and Reactive Systems, Germany

²Technical University of Braunschweig, Institute for Thermodynamics, Germany

³TLK-Thermo GmbH, Germany

m.loeffler@tu-bs.de, m.huhn@tu-bs.de, ch.richter@tu-bs.de, r.kossel@tlk-thermo.de

Abstract

The simulation language Modelica is an object oriented language with all the advantages and potential drawbacks that are characteristic for object oriented programming languages. The reusability of source code and the possibility to develop nicely structured libraries using inheritance, aggregation and polymorphism are two of the main advantages object oriented languages have to offer. Although there are good mechanisms given to structure libraries, one of the drawbacks is that it can become very hard to understand large libraries especially for users who just want to use them to carry out simulations without getting into all the details. The presented work has the goal to provide an easy to use tool that is capable of graphically visualizing the structure of Modelica libraries and that therefore enables the developer as well as the end user of Modelica libraries to better control and understand the structure of libraries.

Keywords: Object oriented modeling; visualization; class diagram

1 Introduction

The object oriented character of Modelica is one of its very important features. It enables the developer to reuse code in a very efficient way, improves team work in the design process of a library and helps the user to easily exchange components in a simulation. There are other advantages that could be added to this list. But there also are drawbacks that almost every Modelica developer and user knows from his own experiences. A growing aggregation depth and multiple inheritance can make a library almost illegible. Tracking bugs, implementing new models or changing existing models might become very difficult for developers that are not completely familiar with the library and its structure. The

sustainability of the library might be in danger while it should be improved by using object oriented techniques. Having been in this situation gave us the idea to develop a tool that helps to analyze the structure of Modelica libraries. The developed tool, Modelica CDV, enables the Modelica developer to improve the structure of the developed library and offers the user of the library a simple way to understand its structure. The tool will be available as freeware.

2 Modelica Libraries

To build and maintain models on the basis of Modelica libraries the developer needs a clear and detailed understanding not only of the library elements but also of the object oriented structure of the library and in particular of the relations among the elements. The Modelica Association coordinates the development of free libraries. There are for example libraries available to simulate multi-body systems, to model fuel cells and to model magnetic actuators and drives. For more information about available libraries see [1]. The structuring concepts of Modelica like multiple inheritance, polymorphism, aggregation, and composition are a prerequisite for the compatibility and reuse of submodels (for details see [2]). Consequently, they considerably contribute to the efficiency and conciseness of modeling in Modelica. However, if the structuring concepts are used in combination in a large library, the overall structure may become far from trivial.

Modelica is based on a Cardelli type system [3] and supports multiple inheritance. In difference to nominal type systems like in Java [4], subclasses cannot only be declared explicitly by a keyword like *extends*, e.g. “model A extends model B”, but also implicitly by the fact that a class extends the set of public attributes of another class. For building a

subtype this way, all public attributes of the desired base class have to be implemented (using the same names and types) manually. Hence, to reconstruct the inheritance hierarchy within a library the information about the explicitly stated extensions has to be joined with the analysis on extensions implicitly given by inclusion of public attributes.

Modelica provides powerful mechanisms for polymorphism which are an essential device for compositionality and exchangeability of submodels. The first mechanism for polymorphism concerns exchangeable objects, i.e. using the keyword *replaceable* an exchangeable object is built as an attribute of a class. When the class gets instantiated the type of the object can be changed in the class modifier. This way an element of a circuit (for example a resistor) can easily be replaced by another element (for example a capacitor).

The second mechanism for polymorphism is *local classes*. A local class is used if one wants to replace a number of objects within a complex model but all replacements have to coincide on the type, e.g. in an electric circuit several resistors may be replaced under the condition that all of them belong to the same resistor type. Technically, exchangeability is achieved by declaring a local class as a parameter of the model. The parameter is set to a concrete type when instantiating the model and all submodels (objects) that are derived from the local class within the model are set to that concrete type.

The third and most complex mechanism for polymorphism is variable inheritance. It is used for modeling generic objects that can assume the shape of all objects of a category.

```

model GenericResistor
  replaceable model ResistorModel =
    CeramicResistor extends Resistor;
protected
  extends ResistorModel
end GenericResistor;

model Circuit
  GenericResistor resistor(redeclare model
    ResistorModel = SyntheticResistor);
end Circuit;

```

Figure 1: Modelica code example for variable inheritance

Figure shows the implementation of a generic resistor using variable inheritance. The generic resistor is derived from the exchangeable local class “ResistorModel”. Changing the type of the local

class when instantiating an object of the generic resistor results in changing the base class of the generic resistor. The advantage compared to the other mechanisms is that the exchange of an object is not only possible inside its container class.

The keyword *redeclare* cannot only be used for exchanging objects. The expression “redeclare record extends GeometryData” for example allows the extension of the already existing record “GeometryData”. Additional attributes can be created easily that way.

Member variables (also called attributes) are typed. They are either simple or complex. Complex attributes representing associations between objects further enhance the structuring concepts of Modelica libraries. An association represents a dependency relation between objects. Of particular interest are aggregation and composition as two specific associations that express whole/part relations. For instance, a car can be modeled as composition of a motor, wheels, etc. In Modelica, composition is mainly used to assemble complex objects. Composition, the stronger form, is most appropriate in the context of physical objects where the components are assigned to a unique compositum, e.g. a motor object is part of a particular car. Whether a complex attribute represents an aggregation or composition is hardly to analyse statically because it results from the context in which the attributes will be used.

The structuring concepts of Modelica significantly ease the work of a library developer but they complicate the analysis of a library for the user.

Modelica libraries are stored in hierarchically structured directories. The position of a file reflects the affiliation of the classes stored in this file with respect to a package, because directories represent packages. Thus the structure of the repository partially reflects the structure of the library. Alternatively, all classes of a library can be stored within one file. Then the affiliation to packages only depends on the arrangement of the classes inside the file.

For analyzing the object oriented structure of Modelica libraries it is important to understand the structure of the classes, objects and dependencies among each other. Because the “algorithms” and “equations” inside a class are less relevant for the structure of the library, they are neglected in this paper.

To summarize, Modelica provides a great variety of structuring concepts that may be used in combination and lead to complex library structures that are hardly understandable on the basis of pure code review.

3 Parsing

Parsing is the transformation of text files into an internal data representation. Most of the Modelica parsers are used in commercial applications and are not available for developers. However, there are some free parsers available. An example is ModelicaXML that parses Modelica code to an XML representation [5]. We still did not use one of the free parsers to be as independent as possible. The parser that was developed within the scope of this work only parses information that is required for an object-oriented analysis of the library but could easily be extended to parse more information.

In our application the goal of parsing is the creation of an internal data structure that contains all information of a Modelica library that is required for drawing a class diagram. Therefore every class of a library will be represented by an object containing the following information

- name and type (i.e. package, model,...)
- path of the text file that contains the class
- code
- local classes
- all attributes (member variables)
- all associations (inheritance, aggregation)
- version of the class (if specified)
- package(s) the class belongs to

The process of parsing a Modelica library can be decomposed into the following steps

- read in the code
- format the code
- detect classes
- detect attributes of classes
- detect associations
- analyze redeclarations
- perform name lookup
- analyze dependencies caused by Cardelli type system

The steps have to be performed in the given order due to their interdependence. Detecting associations for example cannot be performed before all classes have been detected.

The following passage explains the parser in more detail.

Reading in a Modelica library starts with reading in the required text files. There are in general two ways to store Modelica libraries: all classes and packages of a library can be stored in one file or they can be split up into different files that are nested in folders

to represent the affiliation to packages. If the classes are stored in several files, there needs to be a file “package.mo” in each folder declaring the package. All other classes declared in files within this folder or in sub-folders belong to this package. The parser takes the path of the folder that contains the library as an input and checks, whether a file “package.mo” exists in the specified path. The parser will then read in the “package.mo” and all dependent files in the same folder and will register the package affiliation for each file read in. This process is repeated for each subfolder containing a “package.mo”. If there is only one file containing the whole library, the content of this file can be read in without analyzing the package affiliation.

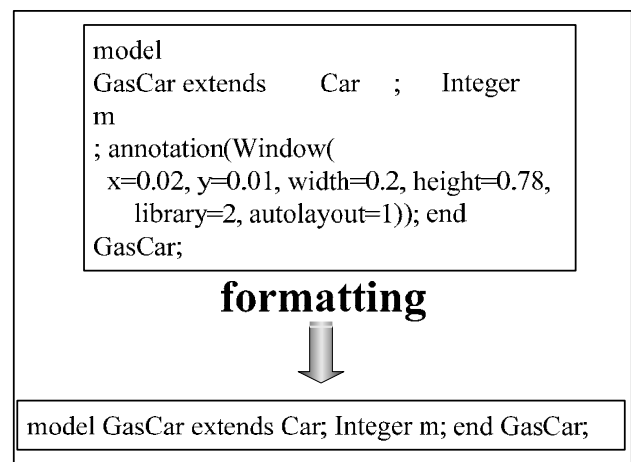


Figure 2: Example for formatting process during parsing

The entire library is internally available as a character string after this first step. The parser formats the code in a second step carrying out the following operations:

- word-wraps get removed
- superfluous space characters get removed
- annotations other than “version information” get removed (i.e. annotations for documentation, icon information)

This second step simplifies the following process by transforming the unformatted string into a well formatted character string containing only the information necessary to perform an object-oriented analysis. Figure 2 shows an example for the formatting process.

The next step is to detect all classes and to create objects representing them. This step requires considering the nested structure of classes in Modelica. Classes are detected recursively starting at the top level. The beginning and the end of a class is detected on the base of keywords. If a class has been

found, an object with all necessary attributes for its representation is generated. Not all attributes of the represented object can be generated at this time because some information (associations, member variables) is not yet available. The name of the class, its source code, comments and information about its package affiliation are stored. The code of a nested Modelica class is not stored within the containing class if it is not a local class. Local classes are for example used for polymorphism. The model “ResistorModel” from Figure 1 is a local class.

Detecting attributes of each class includes detecting the type and the name of each attribute as well as its default value and the comment if specified. Start values and further attributes such as min and max values are omitted but could be included in a future version.

The keyword *extends* defines inheritance in Modelica. A derived class inherits all attributes from its super class, implying that all attributes of the super class have to be copied into the derived class. However, this is not possible at that point because all classes are represented by an object but they are not linked to each other. This means that copying the attributes from super to derived classes has to be performed after the name lookup. If an attribute is not primitive it is handled as aggregation.

```

package BaseGeometry
  replaceable record GeometryData
    Real length;
  end GeometryData
end BaseGeometry;

package Cylinder extends BaseGeometry;
  redeclare record extends GeometryData
    Real diameter;
  end GeometryData;
end Cylinder;

```

Figure 3: Modelica code example for redeclaration

The keyword *redeclare* indicates in Modelica that the type of the object can be changed. Figure 3 shows the extension of a record “GeometryData” within the package “Cylinder” “GeometryData” is extended by the attribute “diameter”. If the class “GeometryData” from “Cylinder” is used somewhere, it contains the attribute “diameter”. The parser handles this just like inheritance. A new class is internally generated that contains all attributes from “GeometryData” plus the attribute “diameter”. This class belongs to the package “Cylinder” and

will later be displayed as derived from “GeometryData”.

All information collected until now is represented as character strings. As mentioned earlier, it is very often necessary to have a link to an object representing a certain class. The already described situation of inheritance is a very good example in which all attributes of the super class have to be copied to the derived class. If there is a definition like “model GasCar extends Car” the parser has to resolve the class “Car”. The problem arising here is that there might be several classes with the name “Car” within the analyzed library because Modelica allows different classes having the same name. Name lookup ensures that the correct class is used within the given context. In general all classes within the package containing the class whose name has to be resolved are possible choices. It is also possible to give a bit more specific information when extending a class. One could for example write “model GasCar extends Basics.Car” “Basics” is in this case the package that the class “Car” belongs to. Another important case to be considered is the “import” statement. With this keyword, namespaces can be defined or used. If there is a definition like “import myCar = Basics.Car” the name lookup has to be performed with “Basics.Car” instead of “myCar”.

It is also possible to use short notations such as “package C = B;” in Modelica. In this case all classes of package “B” get duplicated and copied into package “C”. The declaration can be interpreted as equivalent to its long form “package C extends B”. The associations of a class are stored inside the representing object. Thereby associations from and to another class are available.

Modelica uses the Cardelli type system which is a structural type system. Many other object oriented languages such as Java use a nominal type system (see [3] for more information). There has been a discussion within the Modelica language group to change the Modelica type system in a future version but no decision has been made so far. To analyze type equivalence of classes or to find subtypes of a class it is necessary to compare all public attributes of each class. If two classes have the same public attributes they are type equivalent within the scope of a Cardelli type system. If they have at minimum the same public attributes the class with more public attributes is a subtype of the other class. When there is a large number of classes with many attributes, comparing all of them with each other can take quite some time.

After all parsing steps have been performed the internal data representation is available and can be used for displaying the object oriented structure of the library in the style of a UML class diagram (see [6]).

4 Layout

For the automated graphical representation of the structure of Modelica libraries, the selection and tuning of the layout algorithm is crucial. Thus, we consider three types of layout algorithms for drawing the object oriented structure of Modelica libraries. Layout algorithms rely on graph theory. The object oriented structure of libraries is interpreted as a directed graph where a node represents a class and an edge represents an inheritance relationship or aggregation. The selected layout algorithms optimize the graphical representation according to the following goals:

- Minimization of the area used for the resulting chart because libraries can be large.
- Minimization of the number of crossing edges supports the understanding and conceives the diagram. If there is no crossing edge the graph is called planar. The direction of edges is also an important fact: A class that inherits from another should be placed below the class it inherits from.
- Computational efficiency is important so that even for large libraries the class diagrams are generated in an acceptable time.

The first group of layout algorithms relevant for the representation of structural information is the group of the so-called “tree algorithms” [7] which are available in many variations. They are especially suited to illustrate the inheritance relationships of classes while their runtime is linear. A class is represented by a node and all derived subclasses become children of this node. However, tree algorithms are less appropriate for the layout of aggregation relationships since the generated trees can become wide and might need a large area.

Another way to layout graphs is the “Spring Embedder” [8]. The graph represents a physical model where nodes repulse each other. The closer two independent nodes get, the larger the mutual force of repulsion becomes. Nodes also gravitate towards each other in case of a common edge. This algorithm works iteratively. All forces are calculated and the nodes are relocated according to the affecting forces. After a certain number of iterations all forces will be balanced. The size of the resulting chart is

small but it might contain a lot of crossing edges. Moreover, runtime of the Spring Embedder may become a critical issue.

The third group of algorithms minimizes the set of crossing edges.

The Spring Embedder and the algorithm for minimizing crossing edges do not care about the direction of edges and the right adjustment of classes that inherit from another. Each algorithm has its advantages and drawbacks. In order to get a good layout, it is necessary to use two algorithms in combination.

Modelica CDV generates a class diagram closely related to the UML notation (see [6]). The most important elements, the classes, are represented by rectangular boxes containing the name, the attributes and the operations. Here we will omit the operations, i.e. equations and algorithms of a model.

The layout module in Modelica CDV is still subject of discussions and experiments as the advantages and disadvantages of different variants have to be balanced carefully.

A class diagram is considered as a directed graph where a node represents a class and an edge represents an association, i.e. inheritance or aggregation.

For a good layout of a Modelica library several partially contradicting criteria have to be taken into account.

Usually, derived classes are placed below their super class. Consequently, the algorithm tries to do this the same way. Another optimization criterion is a short distance between classes having a relation since one often needs to look at the associated classes as a whole to understand the aggregation or inheritance structure. Hence, minimizing the distance improves readability. But when minimizing the distance between nodes one also decreases the area on which classes are placed. Unfortunately, in larger diagrams with a lot of complex associations you often will not see the end of an association without changing the point of view. Additionally, crossing edges are complicating readability. To simplify the detection of crossing edges, associations are restricted to horizontal and vertical straight lines.

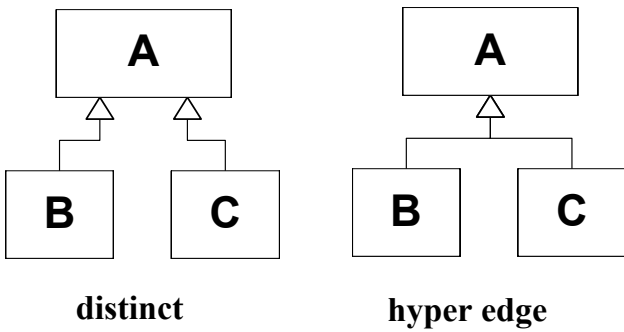


Figure 4: Edge notation for inheritance

After several experiments we decided that the user should direct the layout to his point of interest: Thus, the user may select the elements of a class he wants to see. This way, the user may choose the level of detail he is interested in. If detailed information is faded out, the user will see more classes and associations on the screen.

The size of a class is calculated dynamically before the layout algorithm starts. The size of classes depends on the settings given by the user. Specifying that additional attributes should be shown in the diagram will change the dimension of classes. Every time settings are modified all calculations have to be repeated because the algorithm calculates the absolute position of every class on the panel. If a class becomes larger it probably would overlap with

another one otherwise. The width of the class representation results from the longest word that has to be displayed inside a class. For inheritance associations a “hyper edge” notation is used (see Figure 4). Aggregation associations are characterized by their multiplicity. Therefore an inscription would be needed which might be misleading at the hyper edge. For this reason aggregation is displayed as distinct edges.

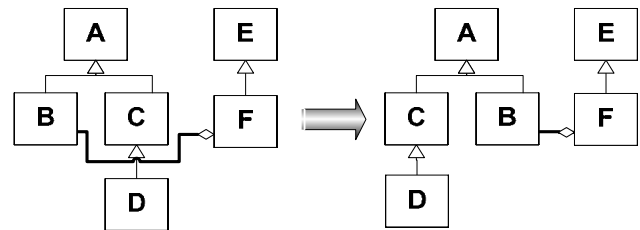


Figure 6: Swapping leaves for minimizing crossing edges

The algorithm for lay outing the classes first calculates the size of every class. Thereafter inheritance associations are analysed and layouted by a tree algorithm from bottom to top and the dimension of the resulting tree is calculated.

The algorithm for adding aggregation associations is still under investigation. After all aggregation associations have been added a swap procedure will

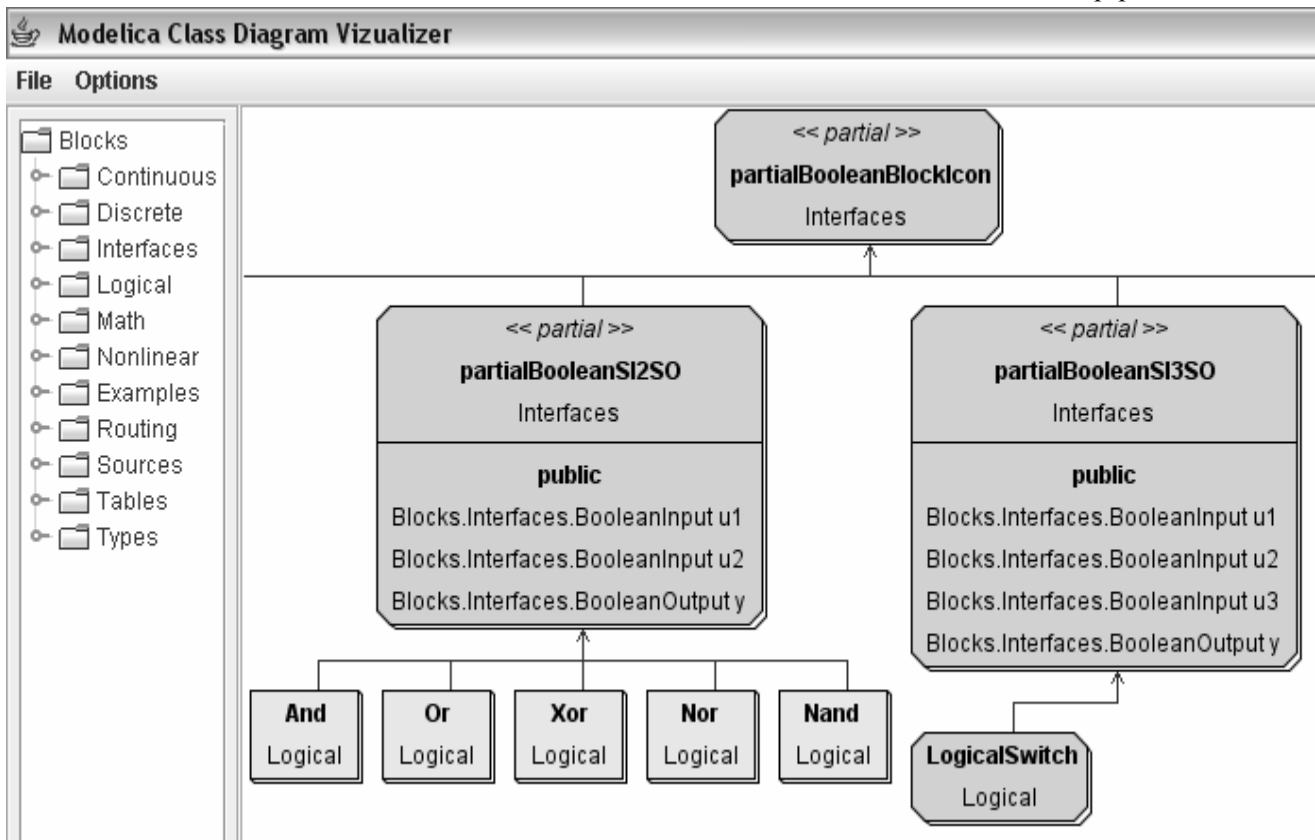


Figure 5: Screenshot of the Modelica Class Diagram Visualizer

remove crossing edges by swapping some leaves (inherited classes) of a tree or the trees itself (see Figure 6). Now nails for the associations will be calculated and all elements will be displayed. The algorithm for swapping and rearranging the leaves (classes) is still in a prototype state at the moment.

5 Using Modelica CDV

The purpose of Modelica CDV is to help the Modelica user as well as the Modelica developer with getting an overview over the structure of libraries.

The graphical user interface that was completely developed in Java using Swing [9] can be configured to best suite the users' preferences. The user may select color and font for each type of class and also choose to display or hide additional information that is usually not displayed in a class diagram but might be handy when analyzing libraries. Figure 7 shows a screenshot of the current configuration dialog. The user can decide to display parameters, constants and variables for both, the public and the protected section, in the class diagram.

Figure 5 shows a screenshot of a class diagram generated by Modelica CDV. The package "Blocks" from the Modelica Standard Library 2.2 was parsed and displayed for this example. Note that only the inheritance associations are displayed in the example screenshot. The different class types such as blocks (i.e. "LogicalSwitch") or models (i.e. "And") are discriminated by their graphical appearance both in shape and color and according to the user's settings. Information about the variables contained in each class is also displayed. The final version of Modelica CDV will also display connectors as small icons.

The package tree containing all classes of the parsed library is displayed on the left hand side of the class diagram as shown in Figure 5. When selecting a class with the mouse the respective block in the class diagram on the right hand side is highlighted and centered.

An important aspect when using Modelica CDV is the time it takes to parse, layout and display a library. The Modelica Standard Library in the version 2.2 containing about 2500 classes was parsed as a representative example which took about two minutes on a standard laptop computer. While the Modelica Standard Library is very likely much larger than most libraries that will be displayed with Modelica CDV, two minutes still is too long. The user can therefore choose to save the parsed library which cuts down the time requirements to a few

seconds. The user just has to be aware that he is using a pre-computed version of the library when using the saved version. A warning is displayed to remind the user of the current operation status.

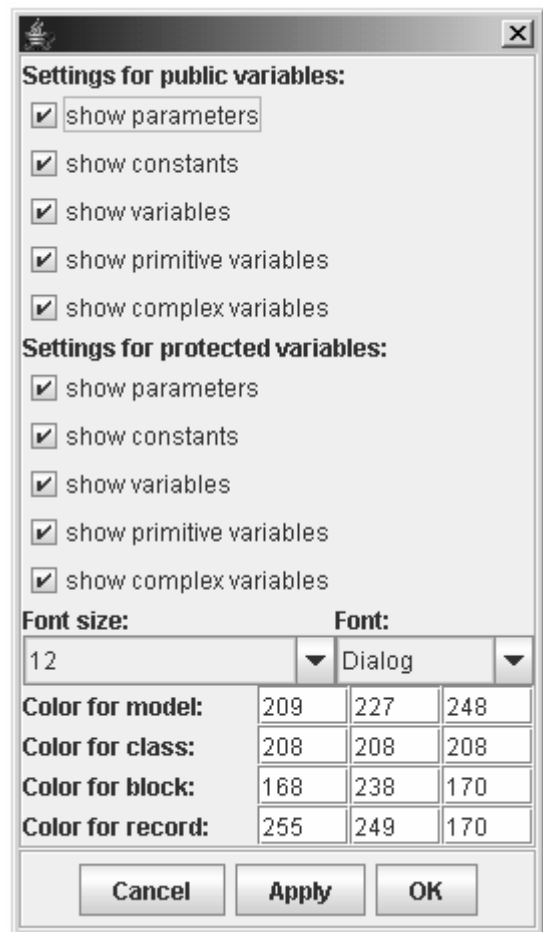


Figure 7: Screenshot of the option dialog in Modelica CDV

6 Conclusions

Modelica CDV is a tool to help Modelica developers and users to better understand the structure of libraries by generating class diagrams closely related to the UML notation. It uses a combination of different layout algorithms to automatically generate a class diagram that is as readable as possible. In the class diagram additional information (i.e. about variables) may be displayed and it can be configured according to the users' preferences.

Visualizing the structure of libraries is the first step towards improving the readability which ensures its sustainability. The developed tool is available as freeware to the Modelica community. The current version only parses and displays information that is required for an object-oriented analysis. Additional

information about equations and algorithms and enhanced analysis that might be of interest is currently neglected but might be included in a future version.

References

- [1] Modelica – Modeling of Complex Physical Systems, Modelica Libraries, <http://www.modelica.org/library/>
- [2] ModelicaAssociation, 2005, Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specifications, Version 2.2
- [3] A.B. Tucker (Ed.), The Computer Science and Engineering Handbook, CRC Press, 1997
- [4] B. Eckel, Thinking in Java, 4th Edition, Prentice Hall, 2002
- [5] A Pop and P. Fritzon, 2004, ModelicaXML: A Modelca XML Representation with Application, Proc. of 4th International Modelica Conference, Hamburg, Germany
- [6] M. Jeckle, C. Rupp, J. Hahn, Barbara Zengler, S. Queins, UML 2 glasklar, Hanser Verlag München, Wien, 2004
- [7] M.Kaufmann, D. Wagner, „Drawing Graphs – Methods and Models“, Springer Verlag, 2001
- [8] Guiseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis, “Graph Drawing – Algorithms for the Visualization of Graphs”, Prentice Hall, 1999
- [9] Sun Microsystems, Java Foundation Classes, <http://java.sun.com/products/jfc/>

Advanced modeling and simulation techniques in MOSILAB: A system development case study

Christoph Nytsch-Geusen¹
 Thilo Ernst¹ André Nordwig¹
 Peter Schwarz² Peter Schneider²
 Matthias Vetter³ Christof Wittwer³
 Andreas Holm⁴ Thierry Nouidui⁴
 Jürgen Leopold⁵ Gerhard Schmidt⁵
 Alexander Mattes⁶

¹Fraunhofer Institute for Computer Architecture and Software Technology
 Kekuléstr. 7, D-12489 Berlin, christoph.nytsch@first.fhg.de

Fraunhofer IIS/EAS², Fraunhofer ISE³, Fraunhofer IBP⁴, Fraunhofer IWU⁵, Fraunhofer IPK⁶

1 Abstract

The design and the optimization of complex technical systems can be supported efficiently by using simulation methods and tools. For this reason, the generic simulation tool MOSILAB (Modeling and Simulation Laboratory) is being developed by a consortium of six Fraunhofer institutes in the GENSIM project. For the modeling process, MOSILAB uses the object- and equation-oriented model description language Modelica®, with a backwards-compatible extension to incorporate elements for describing model structure dynamics [1].

In this article we will use illustrate how MOSILAB's advanced modeling and simulation techniques support the user, with the help of two case studies: a complex energy system and a cutting tool system. Thus, the case studies illustrates very different uses MOSILAB.

2 Case studies

2.1 Complex energy system

The case study of a solar heating system will demonstrate MOSILAB's advanced modeling and simulation techniques, such as model-based development, model structure dynamics, external simulator coupling, or the distributed execution of simulation experiments. The considered system model includes a

solar energy plant model, a building model, a model for the control strategy and an environment model for the climate parameters (see Figure 1).

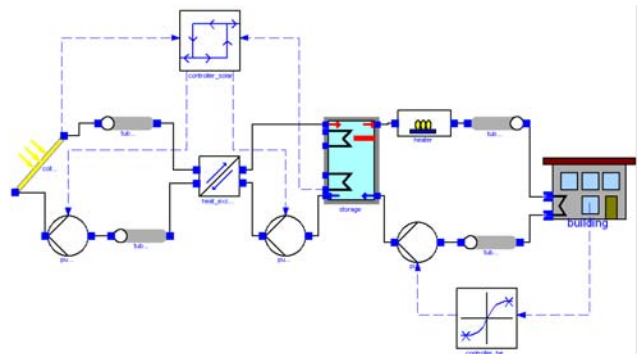


Figure 1: Energy system for solar heating system

The solar energy plant model consists of a primary solar cycle with the collector field, the solar pump and some tubes. The solar energy is transferred by a counterflow heat exchanger to the secondary storage cycle, where a storage pump loads the thermal storage. A discrete two-point controller switches on both mentioned pumps, if the temperature difference between the collector output is higher than the fluid temperature in the lowest point of the storage. The other side of the storage provides the building model with heating energy by a heating cycle. A continuous controller regulates the mass flow between zero and a maximum value, subject to the difference of the current room temperature and the set room temperature. An auxiliary heater delivers additional thermal energy, if the set flow temperature isn't achieved by the storage output temperature.

2.2 Cutting tool system

At present high performance and high precision cutting tools often are designed as modular systems with a complex mechanical behavior. Static and dynamic tool deformations or deflections affect the reliability of the technological process as well as the quality of the workpiece. Tool designers need convenient modeling techniques to predict the tool behavior under working conditions in order to optimize the tool design. Simulation can also help users to choose the most suitable tool and to optimize the cutting conditions.

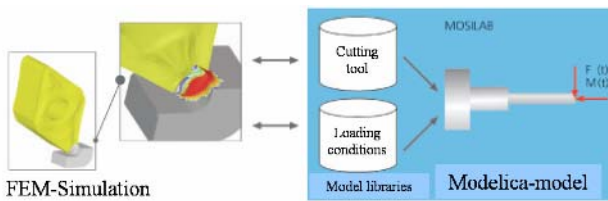


Figure 2: Coupling of FEM-simulators and MOSILAB for cutting tool systems

In the case study of a machining tool, MOSILAB is coupled with two external domain-specific FEM (Finite-Element-Methods) simulators. To simulate the behaviour of high performance cutting machine tools, the machining processes are considered to determine the loading conditions, the tool deformation, the cutting edge displacement and possible malfunctions caused by overloads. Non-linear effects at interfaces between components of a modular cutting tool are also included.

The mechanical and thermal tool loads undergo changes while complex workpiece geometries are machined e.g. dies and molds. Detailed knowledge of the occurring forces and temperatures caused by the chip building for any section of the tool path enables an adjustment of the process parameters to the specific cutting conditions. Thus, the feed rate is optimized by using FEM and analytically based simulation approaches.

This coupled consideration of tool loading and the corresponding tool behaviour enables the choice of the most suitable tool, an estimation of the workpiece quality and provides significant improvements in the efficiency of machining operations.

3 Model-based development with the MOSILAB-IDE

An integrated development environment offers users support at every step of the simulation – from model building to simulation to post-processing [6]. In or-

der to the traditional component diagrams (compare Figure 2), which give overviews about the structures of the plant- or sub-models, further UML^H - diagram types are available. The class diagrams are used to organize classes and their relationships in libraries. Statechart diagrams can be used to model reactive behaviour of components, e.g. the drivers for model structural dynamics. An integrated meta-model ensures model consistency for all diagram types [8].

Thus, the behaviour of a solar thermal plant during „normal operation“ or in different unscheduled states can be represented in an integrated state-dependent structure variable model. Unscheduled states could be the plant behaviour whilst damaged pumps or self-activated pressure control valves, when the solar collector becomes overheated.

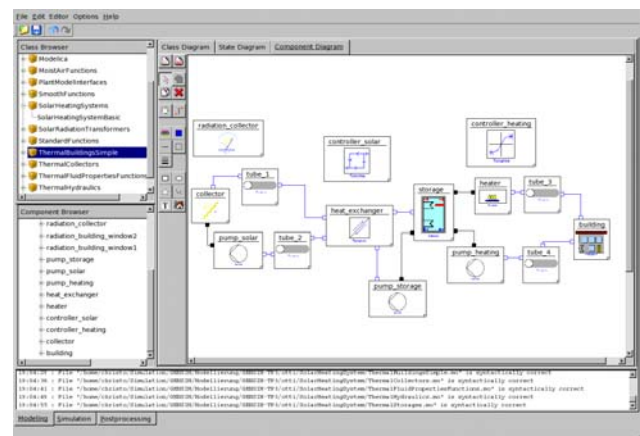


Figure 2: Solar heating system as component Diagram in the MOSILAB-IDE

To support a gap-free model-based development process, a code generator plug-in can be used to produce native embedded system code for controller relevant sub-models.

With this feature, a newly designed controller algorithm can be tested in combination with the virtual model of the controlled system. After successful testing, the same controller algorithm can work on the real controller hardware.

Technically, the description of such controller models uses Modelica’s block and algorithm concepts. Each block implementation can be automatically transformed into controller code for the target operating system. The approach was tested on the embedded Linux derivate BOSS [2].

4 Use of model structural dynamics

Using model structural dynamics [1], MOSILAB is able to adapt the model description, depending on the model state. One example of MOSILAB’s flexi-

bility is its capability to switch between simulation models varying in local resolution. We have chosen the application of a 1D-thermal storage model, embedded in the solar heating system model to illustrate this advantage.

During periods of low collector temperatures or when the storage pump is off, the thermal stratification in the storage can be calculated sufficiently with few numerical nodes ($n_zones = 4$). When hot water enters the storage, it is necessary to use a storage model with substantially more numerical nodes for the thermal gradient calculation ($n_zones = 12$, compare Figure 3).

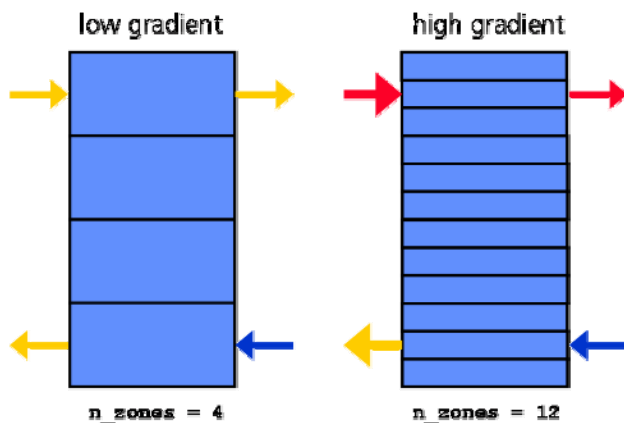


Figure 3: Structural variable storage model, which uses a different number of zones in dependency of the current thermal layering.

The following code fragments of the system model show the implemented strategy for switching between both models. The first part includes the declaration of the component models.

```
model SolarHeatingSystem
...
ThermalCollectorDynamic collector(...);
Pump pump_solar(...), pump_storage(...),
  pump_heating(...);
Tube tube1(...), tube2(...), tube3(...),
  tube4(...);
HeatExchangerCounterflow heat_exchanger(...);
TwoPointController controller_solar(...);
TanhController controller_heating(...);
FlowHeater heater(...);
ThermalBuildingHeatEx building(...);
dynamic Storage storage, tempStorage;
event Boolean finer(start=false);
```

The dynamic parts of the system model are marked with the prefix **dynamic**, in our use case the storage model. Further, the Boolean-variable *finer* has the prefix **event**, which is needed to trigger the replacement from the coarser to the finer storage model.

```
equation
  finer = pre(finier) or
    collector.out.T-pump_storage.in.T > 3.0
  and controller_solar.out > 0;
```

The first equation in the **equation**-section is true, if the difference of the collector temperature and the temperature in the lowest storage zone exceeds 3 K (and the solar pump is on). Then the storage model has to switch from 4 to 12 zones for a better reproduction of the thermal gradient.

The following code illustrates that only the static **connect**-equations are available in the **equation**-section. All dynamic connects between the storage model and its surrounding components are not closed:

```
// controller solar and storage cycle
collector.out.T = controller_solar.in1;
pump_storage.in.T = controller_solar.in2;
pump_solar.alpha = controller_solar.out;
pump_storage.alpha = controller_solar.out;

// controller heating cycle
building.T_air = controller_heating.in2;
273.15 + 20.0 = controller_heating.in1;
pump_heating.alpha = controller_heating.out;
...
// solar circle:
connect(collector.out, tube1.in);
connect(tube1.out, heat_exchanger.in1);
connect(heat_exchanger.out1, tube2.in);
connect(tube2.out, pump_solar.in);
connect(pump_solar.out, collector.in);
// storage solar circle:
// no static connect between
// heat_exchanger.out2 and storage.in_supply1
// no static connect between
// storage.out_supply1 and pump_storage.in
connect(pump_storage.out, heat_exchanger.in2);
// heating circle:
// no static connect between
// storage.out_load_1 and heater.in
connect(heater.out, tube3.in);
connect(tube3.out, building.in);
connect(building.out, tube4.in);
connect(tube4.out, pump_heating.in);
// no static connect between
// pump_heating.out and storage.in_load1
...

```

In the **statechart**-section, which is responsible for the model structure dynamics, the states of the system model (*startState*, *lowResolution*, *highResolution*) are declared and the transitions between the states (*startState* -> *lowResolution*, *lowResolution* -> *highResolution*) are modelled:

```
statechart
  state SolarHeatingSystemBasic
  extends State;
  State lowResolution, highResolution;
  State startState(isInitial = true);

  entry action
    storage := new Storage(n_zones = 4,
                          volume = 30.0,
                          ...);

  end entry;
```

At the beginning of the simulation experiment (*startState* -> *lowResolution*) the storage model is added to the system model and the connections of the storage model to its surrounding components are closed.

```

transition startState -> lowResolution
  add(storage);
  connect(heat_exchanger.out2,
    storage.in_supply1);
  connect(storage.out_supply1,
    pump_storage.in);
  connect(storage.out_load1, heater.in);
  connect(pump_heating.out,
    storage.in_load1);
end transition;

```

If the transition *lowResolution* -> *highResolution* is triggered by the variable *finer* during the simulation experiment, the connections from the storage model are cut by using **disconnect**(*a.p,b.p*) and the old storage model is removed.

```

transition lowResolution -> highResolution
  event finer action
    disconnect(heat_exchanger.out2,
      storage.in_supply1);
    disconnect(storage.out_supply1,
      pump_storage.in);
    disconnect(storage.out_load1, heater.in);
    disconnect(pump_heating.out,
      storage.in_load1);
    remove(storage);

```

Now a new storage model is instantiated with **new** in a higher resolution ($n_zones = 12$). The start values of the new storage model are determined from the current state of the old storage model:

```

tempStorage := new Storage(n_zones = 12,
  volume = 30.0,
  ...);
tempStorage.content.T_zone[1] :=
  storage.content.T_zone[1];
tempStorage.content.T_zone[2] :=
  storage.content.T_zone[1];
tempStorage.content.T_zone[3] :=
  storage.content.T_zone[1];
tempStorage.content.T_zone[4] :=
  storage.content.T_zone[2];
...
tempStorage.content.T_zone[10] :=
  storage.content.T_zone[4];
...

```

Then the new storage model substitutes the old model, must be added to the system model and the connection to its adequate components are closed again.

```

storage := tempStorage;
add(storage);
connect(heat_exchanger.out2,
  storage.in_supply1);
connect(storage.out_supply1,
  pump_storage.in);
connect(storage.out_load1, heater.in);
connect(pump_heating.out,
  storage.in_load1);

```

```

end transition;
end SolarHeatingSystem_SC;
end SolarHeatingSystem;

```

The simulation experiment with MOSILAB for this system model for a summer day is shown in Figure 4. The diagram shows the implemented behaviour. During the morning hours, the solar controller switches the pumps on first (third curve). Two hours later the temperature between the collector output and the temperature in the lowest layer of the storage is greater than 3 K. (This temperature is equal to the input temperature of the storage pump.) As a result, MOSILAB exchanges the coarse storage model with the higher-resolution model ($n_zones = 4$ -> $n_zones = 12$, fourth curve).

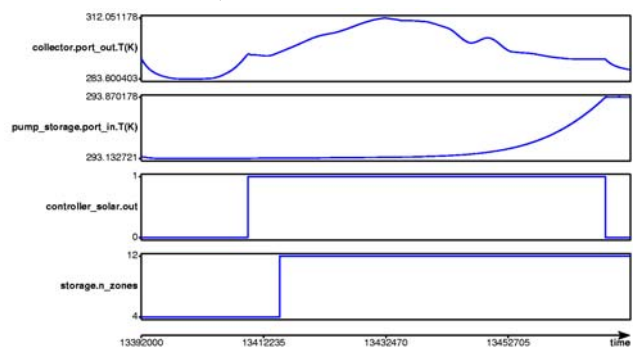


Figure 4: Simulation experiment for a summer day: MOSILAB switches to the detailed model, when hot water enters the storage model and its thermal gradient has to be recalculated in a finer resolution.

5 Numerical coupling with external simulators

Building on the MOSILAB platform, reusable components for simulator coupling have been developed within the GENSIM project. The components support integration with standard tools, such as MATLAB/Simulink or FEMLAB/COMSOL Multiphysics and also domain-specific FEM-Tools such as MARC and DEFORM. This represents a departure from and an improvement upon the typical separate handling of system simulation and FEM (Finite Element Method) simulation.

5.1 MATLAB/Simulink

MOSILAB offers an optional generic interface for MATLAB/Simulink [3]. Thus, it is possible to develop control strategies for embedded systems within MATLAB/Simulink and combine them with a Modelica model of the mixed-continuous discrete system environment. In this scenario each sub-

system is modelled in with the appropriate modelling paradigm within adequate simulation engineering tools.

For a smooth integration of both modelling views, a proxy object is introduced in each view. Within a view, the proxy object represents the wrapped simulator which is realized in the other view. This leads to symmetric model perspectives, which are close to the mental model of the engineer.

In MATLAB/Simulink a generic MOSILAB proxy model can be imported and parameterized via the block parameter dialog. (Compare with Figure 5.)

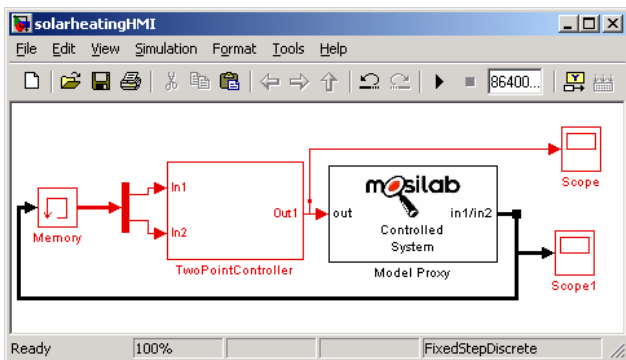


Figure 5: Simulink with an embedded MOSILAB-model

The controlled system model itself (in the case study, the solar energy plant and the building model) is developed using MOSILAB and will be associated with this proxy model, which is shown in the following code fragment:

```
block RemoteModel
  constant Boolean isRemoteModel=true;
  parameter Integer nInp, nOutp;
  input Real inp[nInp];
  output Real outp[nOutp];
end RemoteModel;
```

The constant *isRemoteModel* indicates the presence of a further simulator/driver behind this model. Thus, the numeric algorithms can handle the input and output vectors correctly. The number of input and output variables can be given by *nInp* and *nOutp*. The vectors itself are given by *inp* and *outp*.

The following code illustrates the direct use of this generic remote interface within a Modelica model:

```
model SolarHeatingSystem
  ThermalCollectorDynamic collector
  Pump pump_solar(...);
  StorageSimple storage(...);
  ...
  // the Simulink interface model
  RemoteModel ctrl_solar(nInp=2, nOutp=1);
  ...
equation
  ...
  // controller solar cycle
  collector.port_out.T = ctrl_solar.inp[1];
  storage.content.T_zone[4] = ctrl_solar.inp[2];
  pump_solar.alpha = ctrl_solar.outp[1];
```

```
pump_storage.alpha = ctrl_solar.outp[1];

// solar cycle:
connect(collector.out, tube1.in);
connect(tube1.out, heatexchanger.in1);
connect(heat_exchanger.out1, tube2.in);
connect(tube2.out, pump_solar.in);
connect(pump_solar.out, collector.in);

// storage solar cycle:
connect(heatexchanger.out2, storage.in_supply1);
connect(storage.out_supply1, pump_storage.in);
connect(pump_storage.out, heat_exchanger.in2);
...
end SolarHeatingSystems;
```

In this configuration the simulation is driven by MATLAB/Simulink as the master simulator. Figure 6 illustrates a coupled simulation experiment for the solar heating system during a simulation period of one week in spring. The top screen shows the output signal of the discrete controller, calculated in MATLAB/Simulink. This signal switches the solar pump depending on the temperature difference between the collector output temperature and the temperature in lowest level within the water storage.

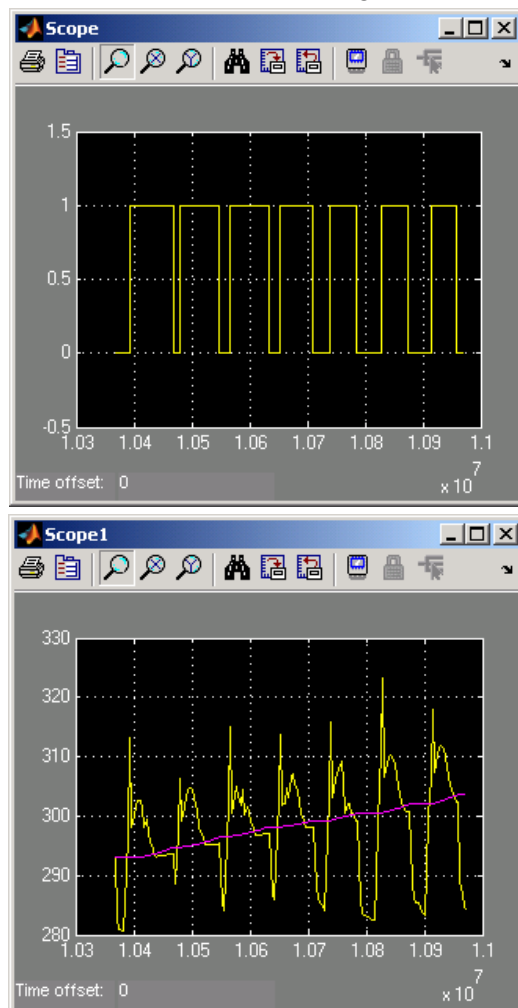


Figure 6: Coupled simulation of MOSILAB with MATLAB/Simulink.

The bottom screen illustrates the dynamic behaviour of the controlled system, calculated in MOSILAB, for the same time period. The curves represent both state variables, which are the input signals of the controller (collector output and storage temperature).

5.2 FEMLAB/COMSOL Multiphysics

One other aspect within the project was the development of a numeric coupling between the simulators MOSILAB and FEMLAB [4]. For simulator couplings which incorporate FEMLAB, two basic principles exist:

1. Coupling within the MATLAB Framework – here the MATLAB engine is used in a C-program or a dedicated coupling model is implemented based on the MEX-interface.
2. FEMLAB is used as a stand-alone simulator – within FEMLAB Java-API models can be loaded, the simulation can be controlled, and the data exchange can be organized.

The second principle is used for this implementation. Figure 7 illustrates the basic structure of the coupling.

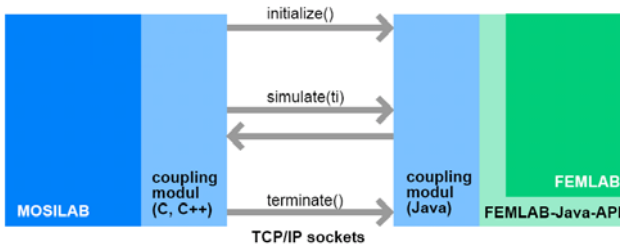


Figure 7: Numerical coupling between MOSILAB and FEMLAB/COMSOL Multiphysics

The communication between the two sides is handled by TCP/IP sockets. This extends the usage of the simulator coupling for a distributed computer environment. Due to a lean coupling implementation, the communication time for the data exchange is much shorter than the simulation time for simple FEM-models. This allows an effective simulation including realistic transient boundary conditions even in combination with models requiring small simulation time steps.

Hence, simulator coupling is suitable for a wide range of applications. This enables the analysis of control systems with a detailed consideration of the controlled process. Furthermore, components in a complex system can be analysed in detail using the MOSILAB-FEMLAB environment, e.g. the multi-dimensional flow within the heat storage as a part of a solar heating system.

5.3 MARC

The finite element code MARC can be used to model complex nonlinear mechanical and thermo-mechanical structures such as machining tools consisting of different components with contact and friction problems. Thus, it is possible to simulate complex system behavior which cannot be adequately described by analytical functions. For example, nonlinear load dependent contact behavior between tool components may result in non-linear tool deformations which require an expensive finite element analysis (FEA). Because of long computing times with FEA the coupling between MOSILAB and MARC will be off-line in most cases. For that purpose a special interface has been developed. The coupling of MOSILAB with MARC will enable to opt between analytical models for relatively simple cutting tools or the more complex FEM-models. That way it is possible to optimize the accuracy of the description of the tool behavior and the expense of the calculations.

Finite element analyses will run outside of MOSILAB and the input and output streams to respectively from the MOSILAB databases will be realized by *readData()* and *writeData()* commands. Using that interface it will also be possible to use predefined FEM-models from a tool model library without special knowledge of finite element modeling.

The loading conditions required by analytical or finite element analyses are provided by the simulation of the cutting process (see chapter 5.4).

5.4 DEFORM

Originally developed for metal forming processes, the FEM-tool DEFORM is also suited for simulation of the chip formation during the machining process. DEFORM is advantageous for an efficient handling of the mesh distortion, which is caused by the high strains within the chip formation zone. Through the remeshing function it is possible to generate a new mesh and to transfer the interpolated values for each node. Thereby the program is able to simulate the mechanical and thermo-mechanical behavior.

In addition, a simplified and fast model for the chip building process in Modelica was developed, which is based on analytical equations (e.g. cutting force calculation according to Kienzle [5]). First of all, the parameters of the simplified Modelica-model have to be calculated with a large number of detailed DEFORM simulations. As a result, the fast Modelica model can be used in the area of validity of these DEFORM calculations.

The cooperation between MOSILAB and DEFORM for the determination of these parameters has been fully automated. First, a routine for automated pre- and post processing for DEFORM was developed. The execution of DEFORM by an external program is possible using the text mode of the software. This enables set up and run of simulations without going through the graphic user interface. The routine needs initial input information about tool geometry and machining parameters provided in a text file. To transfer amongst others, the values for the angles of the cutting wedge or for the feed rate and width of cut from MOSILAB the commands `readData()` and `writeData()` are applied.

```

model DataExchange
  model Kienzle
  ...
  end Kienzle;

  parameter String fname = "inputData.txt" ;
  parameter String fnameOut = "outputData.txt";
  Kienzle k;
  algorithm
    when initial() then
      readData(fname, k);
    end when;
    when terminal() then
      writeData(fnameOut, k);
    end when;
  end DataExchange;

```

These loose coupling of MOSILAB with DEFORM helps to combine the advantages of both methods: First, the short computation time, when solving analytical equations in Modelica and second, the manifold possibilities by analyzing the chip building process through FEM-Analysis.

6 Distributed execution of simulation experiments

Simulators developed using MOSILAB can be generated in various configurations – from a “barebone” variant suitable for constrained environments, such as embedded systems, to a regular desktop application, to a web service for distributed simulation.

6.1 Simulator Services and Interoperability

MOSILAB follows a service-based architectural style. For all configurations except the minimal one, the simulators generated by MOSILAB are created as services communicating through a standard interface. The standard interface is based on the W3C/OASIS web services protocol suite (most importantly, HTTP and SOAP), which allows MOSI-

LAB-developed simulators to be controlled from a wide variety of software environments such as Java, C++, C#.NET, MATLAB, Python, Perl, and Ruby. MOSILAB also supports a more bandwidth-efficient proprietary stream command interface, a direct C++ API, and a Python API. The Python layer abstracts from the underlying transport mechanism; i.e. the same Python experiment script can be used to control a simulator running as a local subprocess and communicating via OS standard I/O pipes, or to control a simulation web service running on a remote machine but having been generated from the same Modelica model (compare figure 8).

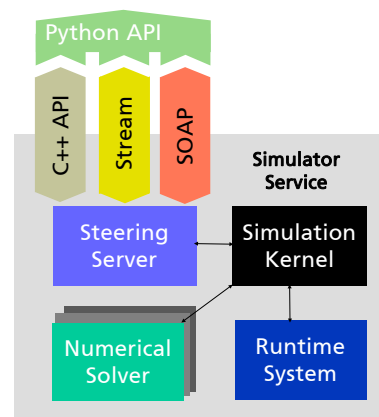


Figure 8: MOSILAB steering interface options

These interfaces are all manifestations of one and the same abstract protocol (called the *MOSILAB unified steering protocol*), which is only expressed in different programming languages. The generic interfaces to other simulators described in section 5 have been developed using these interfacing options specific to MOSILAB, in addition to Modelica’s standard external function interface.

6.2 Speeding up parameter studies by distributed simulation

Often, the system design task at hand requires a large number of simulation runs with differing parameter values, e.g. to obtain knowledge about the system’s behaviour under parameter variations (“robust design”), or to approximate a certain desired property of the system being designed (“optimization”). In the system model from the case study, it makes sense to consider variations of the model parameters “collector area”, “heat store volume” or “building orientation”, as well as parameters of the controller model. The following Figures 9 and 10 illustrate a variation of the collector area parameter.

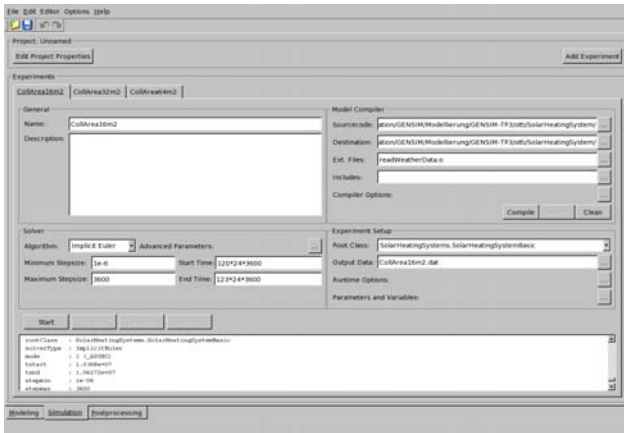


Figure 9: Multiple simulation experiments in the MOSILAB-IDE for varying the collector area

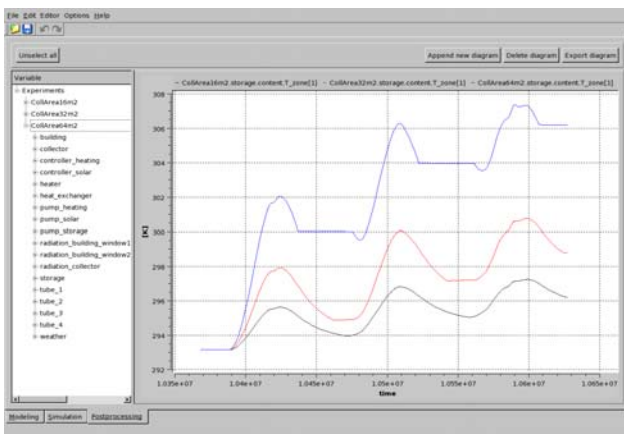


Figure 10: Impact of different collector areas on the storage temperature during a period of 3 days

Variations of multiple parameters lead to multidimensional variant spaces, the size of which (i.e. the total number of simulation runs needed) soon becomes impractical, due to the sheer computation time needed. Statistics-based methods exist to achieve a substantial reduction of the variant space with only a marginal loss of result quality, but even with such methods in place, a large number of necessary simulation experiments are likely to remain. MOSILAB's service-based architecture allows for distribution of simulation experiments as independent, parallel jobs in clusters and computational grids, thus empowering the user to make optimal use of the computational resources available. The individual distributed simulators can nevertheless be interactively controlled and supervised from the MOSILAB-IDE (see Figure 10).

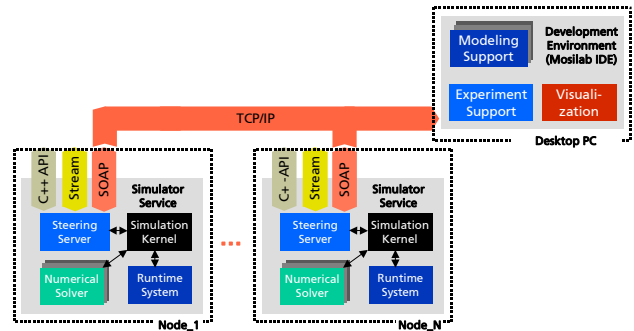


Figure 10: Executing simulator services in the Grid

For very large numbers of parallel experiments, central steering limits scalability, and interactive supervision becomes impractical. In this case, MOSILAB-generated simulators can be distributed in the Grid as independent batch jobs. For more information on MOSILAB and Grid computing, see [7].

References

- [1] Nytsch-Geusen, C. et al. MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. Proceedings of the 4th International Modelica Conference TU Hamburg-Harburg, 2005.
- [2] Nordwig, A. et. al: Codegenerierung aus Simulationsmodellen von heterogenen technischen Systemen am Beispiel einer Pendelsteuerung, VSEK-Report. FKZ01ISC65, 2005.
- [3] Nordwig, A.: Coupling of Modelica and Matlab/Simulink models. Technical Report, Fraunhofer FIRST 2006.
- [4] Clauß, C. et. al: Simulatorkopplung mit FEMLAB. Proceedings of the 1th FEMLAB Conference, Frankfurt am Main, 2005.
- [5] König, W.: Fertigungsverfahren – Band 1: Drehen, Fräsen, Bohren. VDI-Verlag, 1990.
- [6] MOSILAB-Homepage: <http://www.mosilab.de>
- [7] Ernst, T. et al.: MOSILAB: Modelica Simulation from Desktop to Grid. 2. Workshop "Grid-Technologie für den Entwurf technischer Systeme", Dresden, 2006.
- [8] Nordwig, A.: Integration von Sichten für die objektorientierte Modellierung hybrider Systeme, Verlag dissertation.de, ISBN 3-89825-692-8, 2003.

Quantised State System Simulation in Dymola/Modelica using the DEVS Formalism

Tamara Beltrame
VTT, Industrial Systems
PO Box 1000, VM3
02150 Espoo, Finland
Tamara.Beltrame@vtt.fi

François E. Cellier
Institute of Computational Science
ETH Zurich
8092 Zurich, Switzerland
FCellier@Inf.ETHZ.CH

Abstract

Continuous-time systems can be converted to discrete-event descriptions using the Quantised State Systems (QSS) formalism. Hence it is possible to simulate continuous-time systems using a discrete-event simulation tool, such as a simulation engine based on the DEVS formalism.

A new Dymola library, ModelicaDEVS, was developed that implements the DEVS formalism.

DEVS has been shown to be efficient for the simulation of systems exhibiting frequent switching operations, such as flyback converters. ModelicaDEVS contains a number of basic components that can be used to carry out DEVS simulations of physical systems. Furthermore, it is also possible - with some restrictions - to combine the two simulation types of ModelicaDEVS and Dymola (discrete-event and discrete-time simulation) and create hybrid models that contain ModelicaDEVS as well as standard Dymola components.

Keywords: *DEVS formalism; Quantised State Systems; Event-Based Simulation; Numerical Integration*

1 Introduction

Since Dymola/Modelica was primarily designed to deal with continuous physical problems, numerical integration is central to its operation, and therefore, the search for new algorithms that may improve the efficiency of simulation runs is justified.

Toward the end of the nineties, a new approach for numerical integration by a discrete-event formalism has been developed by Zeigler et al. [13]: given the fact that all computer-based continuous system simulations have to undergo a discretisation of one form or another –as digital machines are not able to process raw continuous signals– the basic idea

of the new integration approach was to replace the discretisation of time by a quantisation of state.

The DEVS formalism turned out to be particularly well suited for implementing such a state quantisation approach, given that it is not limited to a finite number of system states, which is in contrast to many other discrete-event simulation techniques.

The Quantised State Systems (QSS) introduced by Kofman [6] in 2001 improved the original quantised state approach of Zeigler by avoiding the problem of ever creating illegitimate models, and hence gave rise to efficient DEVS simulation of large and complex systems.

The simulation of a continuous system by a (discrete) DEVS model comes with several benefits:

When using discretisation of time, variables have to be updated synchronously¹. Thus, the time steps have to be chosen according to the variable that changes the fastest, otherwise a change in that variable could be missed. In a large system where probably very slow but also very fast variables are present, this is critical to computation time, since the slow variables have to be updated way too often. The DEVS formalism however allows for asynchronous variable updates, whereby the computational costs can be reduced significantly: every variable updates at its own speed; there is no need anymore for an adaptation to the fastest one in order not to miss important developments between time steps. This property could be extremely useful in stiff systems that exhibit widely spread eigenvalues, i.e., that feature mixed slow and fast variables.

The DEVS formalism is very well suited for problems with frequent switching operations such as electrical

¹Note that this is not true for methods with dense output. However, the above statement holds for the majority of today's integration methods, since they rarely make use of dense output.

power systems. Given that the problem of iteration at discontinuities does not apply anymore, it even allows for real-time simulation.

For hybrid systems with continuous-time, discrete-time, and discrete-event parts, a discrete-event method provides a “unified simulation framework”: discrete-time methods can be seen as a particular case of discrete-event methods [6], and continuous-time parts can be transformed in a straightforward manner to discrete-time/discrete-event systems.

When using the QSS approach of Kofman in order to transform a continuous system into a corresponding discrete system, there exists a closed formula for the global error bound [2], which allows a mathematical analysis of the simulation.

Since the mid seventies, when Zeigler introduced the DEVS formalism [11], there have emerged several DEVS implementations, most of them designed to simulate *discrete* systems. However, one simulation/modelling software system aimed at simulating *continuous* systems is PowerDEVS [8]: it provides a library consisting of block diagram component models that can be used for modelling any system described by ODE’s (DAE’s), thereby allowing for the simulation of continuous systems.

The implementation of ModelicaDEVS has been kept close to the PowerDEVS simulation software. Hence ModelicaDEVS can be considered a re-implementation of PowerDEVS in Modelica.

2 Continuous System Simulation with DEVS

2.1 The DEVS Formalism

The DEVS formalism has been introduced by Zeigler in 1976 [11]. It was the first methodology designed for discrete-event system simulation that is based on system theory.

A DEVS model has the following structure:

$$M = \langle X, Y, S, \delta_{int}(s), \delta_{ext}(s, e, x), \lambda(s), ta(s) \rangle$$

where the variables have the following meaning (see also [2], Chapter 11):

X represents all possible inputs, Y represents the outputs, and S is the set of states.

The variable e indicates the amount of time the system has already spent in the current state. $\delta_{ext}(s, e, x)$ is the external transition that is executed after an

external event has been received. $\delta_{int}(s)$ is the internal transition that is executed as soon as the system has spent in its current state the time indicated by the time-advance function.

$ta(s)$ is the so-called time advance function that indicates how much time has to pass until the system undergoes the next internal transition. The time-advance function is often represented by variable σ which holds the value for the amount of time that the system has to remain in its current state in the absence of external events.

The λ -function is the output function. It is executed prior to performing an internal transition. External transitions do not produce output.

Figure 1 illustrates the functioning of a DEVS model: the system receives input (top graph) at certain time instants, changes its states according to the internal and external transitions (center graph), and produces output (bottom graph).

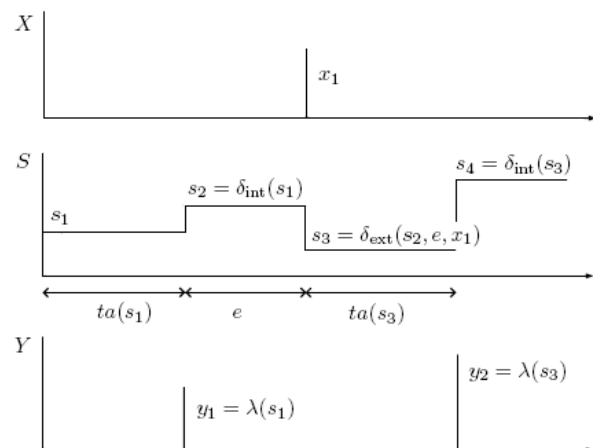


Figure 1: Trajectories in a DEVS model.

In theory, DEVS models can describe arbitrarily complex systems. The only drawback is that the more complex the system is, the more difficult it will be to set up the correct transition functions describing the system. Fortunately, complex systems can be broken down into simpler submodels that are easier to handle. The fact that DEVS is closed under coupling [2] makes such an approach viable.

Figure 2 illustrates this concept: the model N consists of two coupled atomic models M_a and M_b . N can be said to wrap M_a and M_b and is indistinguishable from the outside from an atomic model.

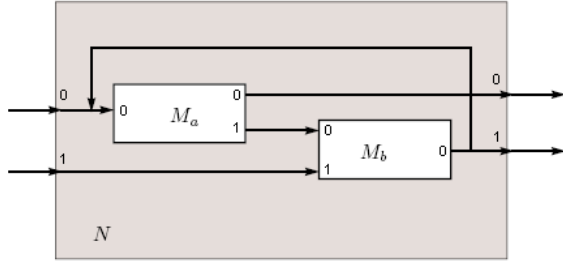


Figure 2: Coupled DEVS models [2].

2.2 Quantised State Systems

For a system to be representable by a DEVS model, it must exhibit an input/output behaviour that is describable by a sequence of events. In other words, the DEVS formalism is able to model any system with piecewise constant input/output trajectories, since piecewise constant trajectories can be described by events [2].

Continuous state variables are being quantised. Consider the following system represented by the state-space description:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$$

where $\mathbf{x}(t)$ is the state vector and $\mathbf{u}(t)$ is the input vector, i.e. a piecewise constant function. The corresponding quantised state system has the following form:

$$\dot{\mathbf{q}}(t) \approx \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t), t)$$

where $\mathbf{q}(t)$ is the (componentwise) quantised version of the original state vector $\mathbf{x}(t)$. A simple quantisation function could be:

$$\mathbf{q}(t) = \text{floor}(\mathbf{x}(t)).$$

Unfortunately, the transformation of a continuous system into its discrete counterpart by applying an arbitrarily chosen quantisation function can yield an illegitimate model². Thus, the quantisation function has to be chosen carefully, such that it prevents the system from switching states with an infinite frequency. This property can be achieved by adding hysteresis to the quantisation function [6], which leads to the notion of a Quantised State System (QSS) as introduced by Kofman [6] providing legitimate models that can be simulated by the DEVS formalism. A hysteretic quantisation function is defined as follows [2]: Let $Q = \{Q_0, Q_1, \dots, Q_r\}$ be a set of real numbers where $Q_{k-1} < Q_k$ with $1 \leq k \leq r$. Let Ω

be the set of piecewise continuous trajectories, and let $x \in \Omega$ be a continuous trajectory. The mapping $b : \Omega \rightarrow \Omega$ is a hysteretic quantisation function if the trajectory $q = b(x)$ satisfies:

$$q(t) = \begin{cases} Q_m & \text{if } t = t_0 \\ Q_{k+1} & \text{if } x(t) = Q_{k+1} \wedge q(t^-) = Q_k \wedge k < r \\ Q_{k-1} & \text{if } x(t) = Q_k - \varepsilon \wedge q(t^-) = Q_k \wedge k < 0 \\ q(t^-) & \text{otherwise} \end{cases}$$

and:

$$m = \begin{cases} 0 & \text{if } x(t_0) < Q_0 \\ r & \text{if } x(t_0) \geq Q_r \\ j & \text{if } Q_j \leq x(t_0) < Q_{j+1} \end{cases}$$

The discrete values Q_i and the distance $Q_{k+1} - Q_k$ (usually constant) are called the quantisation levels and the quantum, respectively. The boundary values Q_0 and Q_r are the upper and the lower saturation values, and ε is the width of the hysteresis window. Figure 3 shows a quantisation function with uniform quantisation intervals.

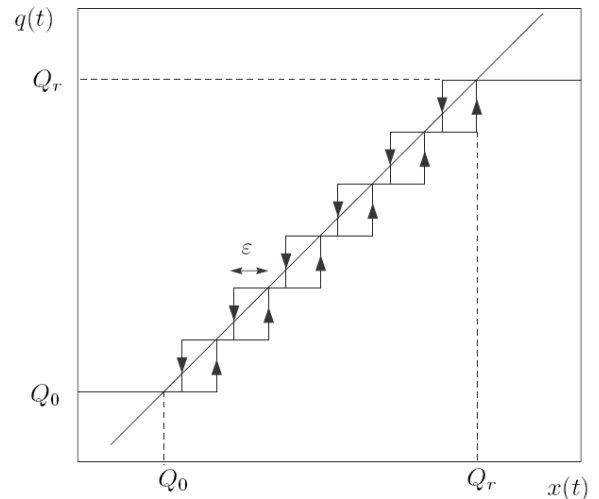


Figure 3: Quantisation function with hysteresis [2].

The QSS described above is a first-order approximation of the real system trajectory. Kofman however has also introduced second- and third-order approximations that may reduce the error made by the approximation. These systems are referred to as QSS2 [7] and QSS3 [9], respectively.

3 ModelicaDEVS

The average block of the ModelicaDEVS library exhibits the following basic structure:

²Definition [2]: "A DEVS model is said to be legitimate if it cannot perform an infinite number of transitions in a finite interval of time." Illustrative examples of illegitimate models can be found in [2] and [6].

```

1  block SampleBlock
2    extends ModelicaDEVS.Interfaces. ... ;
3    parameter Real ... ;
4
5  protected
6    discrete Real lastTime(start=0);
7    discrete Real sigma(start=...);
8    Real e;
9    Boolean dext;
10   Boolean dint;
11   [...other variable declarations...]
12
13  equation
14    dext = uEvent;
15    dint = time>=pre(lastTime)+pre(sigma);
16
17    when {dint} then
18      yVal[1]= ...;
19      yVal[2]= ...;
20      yVal[3]= ...;
21    end when;
22    yEvent = edge(dint);
23
24    when {dint, dext} then
25      e=time-pre(lastTime);
26      if dint then
27        [...internal transition behaviour...]
28      else
29        [...external transition behaviour...]
30      end if;
31      lastTime=time;
32    end when;
33
34  end SampleBlock;

```

The following sections will offer more insight into the reasons for this specific block structure.

In accordance with the PowerDEVS implementation, ModelicaDEVS event ports (connectors) consist of four variables representing the coefficients to the first three terms (constant, linear, and quadratic) of the function's Taylor series expansion, and a Boolean value that indicates whether a block is currently sending an event.

Dense output can then be approximated as:

$$y_{out} = y_0 + y_1 \cdot (t - t_{last}) + y_2 \cdot (t - t_{last})^2$$

whereby the coefficient of the quadratic term of the Taylor series, $y_2 = yVal[3]$, is only used by the third-order accurate method, QSS3, whereas the linear term, $y_1 = yVal[2]$, is used by QSS2 and QSS3.

Let us now consider a small example in order to gain increased insight into the role of the Boolean variable of the port. Let us assume a two-block system consisting of block A and block B, where the only input port of block B is connected to the only output port of block A. Every block features a variable `dext` accompanied

by an equation

```
dext = uEvent;
```

where `uEvent` is the Boolean component of the connector that represents an input event. Suppose now that block A produces an output event at time $t = 3$. At this precise instant, it updates its output vector with the appropriate values (the coefficients of the Taylor series) and sets `A.yEvent` to true:

```

when dint then
  yVal[1]= ...; //new output value 1
  yVal[2]= ...; //new output value 2
  yVal[3]= ...; //new output value 3
end when;
yEvent = edge(dint);

```

Still at time $t = 3$, block B notices that now `B.uEvent` has become true (note that `B.uEvent = A.yEvent` because the two blocks are connected), and therefore `dext` has become true, also. Consequently, Block B is executing its external transition [4].

A DEVS model must contain code to perform internal and external transitions, as well as execute the time advance and output functions at the appropriate instants. All of these functions have to be explicitly or implicitly present in the ModelicaDEVS blocks.

The *time advance function* is normally represented by a variable `sigma`. It is a popular trick in DEVS to represent the current value of the time advance function by `sigma` [2].

The *internal transition* is executed when `dint` is true. An internal transition depends only on `sigma`. Hence the value of `dint` can be calculated as:

```
dint = time >= pre(lastTime) + pre(sigma);
```

where `lastTime` holds the time of the last execution of a transition (internal or external).

The *external transition* is executed when `dext` is true. The variable `dext` is defined as follows:

```
dext = uEvent;
```

The internal and external transitions are represented by a when-statement. The reason for packing the internal and external transitions into a single when-statement instead of having two separate when-statements, one representing the internal transition and the lambda function, the other one representing the external transition, is due to a rule of the Modelica language specification that states that equations in different when-statements may be evaluated simultaneously. Hence, if there are two when-statements each containing an expression for a variable X, X is considered overdetermined. This circumstance would cause a syntactical problem with variables that have to be updated both during the internal and the

external transition and thus would have to appear in both when-statements. For this reason, we need to have a when-statement that is active if either `dint` or `dext` becomes true. Subsequently, an additional discrimination is done *within* the when-statement, determining whether it was an internal (`dint` is true) or an external transition (`dext` is true) that made the when-statement become active, and as the case may be, updating the variables with the appropriate value. The *lambda function* is executed right before an internal transition. Lines 17-22 of the “block basic structure” code (beginning of Section 3) constitute the typical lambda function part, containing a when-statement and a separate instruction for the `yEvent` variable. The right hand side of the equations in the lambda function normally depends on `pre()` values of the used variables. This is due to the fact that the lambda function has to be executed *prior to* the internal transition. The variable `yEvent` has to be true in the exact instant when an internal transition is executed and false otherwise. This behaviour is obtained by using the Modelica `edge()` operator.

There is one particular situation that can occur in a model that requires special attention: let us assume two connected blocks, where both block A and block B have to execute an internal transition simultaneously (Figure 4). Whereas block A simply

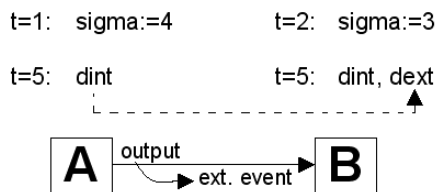


Figure 4: Concurrent events at block B.

executes its internal transition, block B is confronted with the problem of concurrent events: from block A it receives an external event, but at the same time, it was about to undergo its own internal transition. Which event should be processed first? This question is to be answered by the priority settings between the two blocks.

In our simple two-block example there are only two possible priority orderings with the following consequences: either block A is prior to block B, and block A will produce the output event before block B executes the internal transition (block B will first execute an external transition triggered by the external event it received from block A), or block B is prior

to block A, such that block B will first undergo its internal transition and receive the external event right afterwards, when A will be allowed to execute its internal transition.

The problem of block priorities can be solved in two ways: by an explicit, absolute ordering of all components in a model (e.g., a list), or by letting every block determine itself whether it processes the external or the internal event first, in case both of them occur simultaneously. ModelicaDEVS implements the latter approach. As can be seen in the “block basic structure” code, internal transitions take always priority over external transitions (line 26: the code checks first whether `dint` is true).

The reason for this choice is quite simple. As internal events are processed before external events, and since internal events are accompanied by output events, the variable `yEvent` can be computed as a function of `dint` alone. If we were to force external events to be processed before internal events, we would need to make sure that `yEvent` is only set true in the case that the internal event is not accompanied by a simultaneous external event. Thus `yEvent` would now be a function of both `dint` and `dext`. Yet, `dext` is a function of `uEvent`. Thus, if ModelicaDEVS blocks were connected in a circular fashion, as this is often the case, an algebraic loop in discrete (Boolean) variables would be created, which would get the Dymola compiler into trouble.

By forcing the internal events to always take preference over external events, ModelicaDEVS blocks can be interconnected in an arbitrary fashion without ever creating algebraic loops in the Boolean event-indication variables.

Note that since Dymola/Modelica is already aimed at object-oriented modelling, which includes the reuse of multi-component models as parts of larger models, the issue of hierarchically coupled models did not require any special treatment in ModelicaDEVS.

Dymola can trigger two types of events: *state events* that require iteration to locate the event time, and *time events* that make Dymola “jump” directly to the point in time when the time event takes place.

The only expressions responsible for activating the when-statements in the models, namely:

```
dext = uEvent;
```

and:

```
dint = time >= pre(lastTime) + pre(sigma);
```

both trigger time events and hence avoid the computationally more expensive state events.

An earlier version of ModelicaDEVS used an approach that triggered mostly state events. Inspired by the book of Fritzon [4], a number of small modifications have been applied that converted all state events to time events. Performance comparisons carried out between the two versions showed that the time-event approach is roughly four times faster than an equivalent approach triggering state events.

4 Results

4.1 Efficiency

In order to compare the run-time efficiency of ModelicaDEVS to other simulation software systems (PowerDEVS and standard Dymola), a system with frequent switching operations was modelled using each of the three tools (PowerDEVS, ModelicaDEVS and Dymola), and the execution times of the three codes were compared against each other.

The chosen system is the flyback converter example presented in [5].

The flyback converter can be used to transform a given input voltage to a different output voltage. It belongs to the group of DC-DC converters.

A very simple electrical circuit with a voltage source connected to the primary winding of the converter and a load to its secondary winding looks as shown in Figure 5.

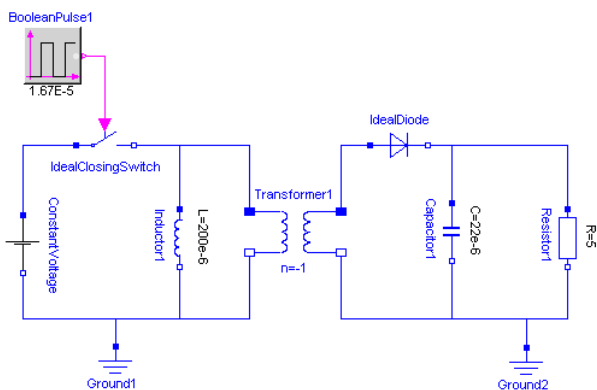


Figure 5: The flyback converter in Dymola.

Figure 6 shows the first two milliseconds of a simulation run of the flyback converter circuit given in Figure 5. The rapid switching is a result of the high switching rate of the ideal switch.

The flyback converter is described by a set of acausal equations in Dymola. However, in order to be able to model the flyback converter in either ModelicaDEVS

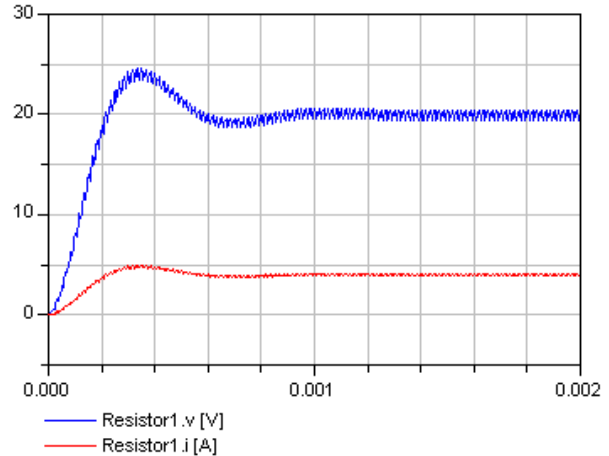


Figure 6: The flyback converter output.

or PowerDEVS, the behaviour of the converter needs to be converted to a causalised block diagram³, which then can be modelled using component models of the PowerDEVS/ModelicaDEVS libraries.

Figure 7 shows the flyback converter model built in ModelicaDEVS. The structure of this block diagram is also valid for the PowerDEVS model.

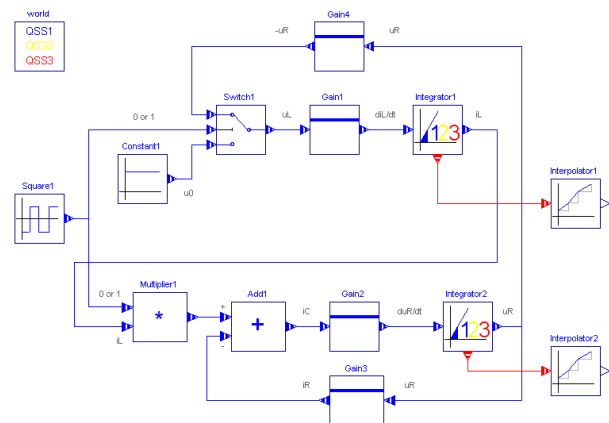


Figure 7: The ModelicaDEVS flyback converter.

Table 1 provides the average simulation CPU time for a simulation of 0.002 seconds of the flyback converter model in standard Dymola, ModelicaDEVS, and PowerDEVS, respectively. The Dymola and ModelicaDEVS model were simulated setting the numerical integration method to LSODAR⁴. Testing has been carried out on an IntelCeleron 2.6 GHz Laptop with 256MB RAM. The resulting CPU time may vary from

³For more details on the causalising process in the flyback converter example, see [1].

⁴Although ModelicaDEVS does not make use of LSODAR directly, the event handling behaviour of Dymola is somewhat influenced by the selection of the numerical integration algorithm.

one computer system to another, but the relative ordering is expected to remain the same.

Table 1: Execution efficiency comparison.

		CPU time [s]	time events	result points
Dymola		0.062	239	738
M-DEVS	QSS1	3.55	6363	11829
	QSS2	0.688	958	2299
	QSS3	0.656	833	2164
P-DEVS	QSS1	0.064	N/A	N/A
	QSS2	0.019	N/A	N/A
	QSS3	0.018	N/A	N/A

Table 1 shows a clear ordering of the three different systems in terms of performance: PowerDEVS is faster than Dymola, which in turn is faster than ModelicaDEVS.

First, it needs to be remarked that standard Dymola simulates this model very efficiently. The switching (BooleanPulse) block leads to time events only, whereas the diode should lead to state events. Yet, this is not the case.

Switching at the input leads immediately to a switching of the diode as well. Since Dymola iterates after each event to determine a consistent set of initial conditions, the switching of the diode is accomplished at once without need of first triggering a state event.

Second, the model is quite trivial. The execution time is almost entirely dictated by the number of time events handled. What happens in between events is harmless in comparison.

Standard Dymola performs exactly one time event per switching. In contrast, ModelicaDEVS performs considerably more time events. Time events take here the role of integration steps.

Figure 8 shows the constant term of the Taylor series expansion of the load voltage as a function of time for QSS1 and QSS3. QSS1 requires a new time event as soon as the constant output no longer represents the true output, whereas QSS3 requires an event only, when the second-order accurate Taylor series expansion no longer approximates the true output. QSS1 requires roughly eight times as many events as QSS3, and is therefore between five and six times slower. Yet, even QSS3 requires roughly three times as many events as standard Dymola. In addition, the ModelicaDEVS model contains roughly three times as many variables as the standard Dymola model. All of these variables are being stored at every event. Consequently, QSS3 is roughly nine times slower than standard Dymola.

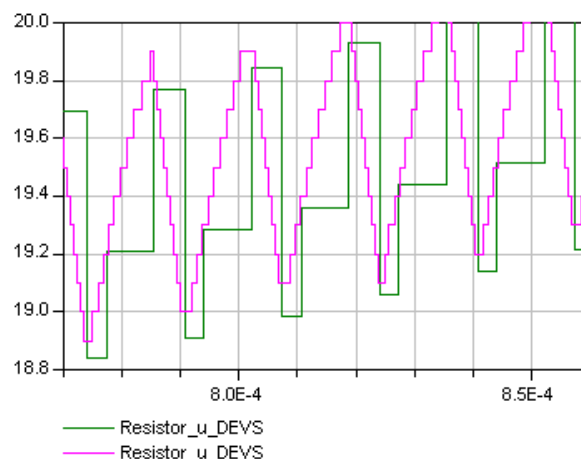


Figure 8: QSS3 simulation vs. QSS1 simulation.

Yet, QSS3 in PowerDEVS is roughly three times *faster* than standard Dymola for comparable accuracy. A comparison between PowerDEVS and ModelicaDEVS is not straightforward. PowerDEVS implements Zeigler's hierarchical simulator [12], whereas ModelicaDEVS operates on simultaneous equations and synchronous information flow [10]. Consequently, PowerDEVS suffers from requiring message passing to implement the communication between blocks, but enjoys the advantage of only having to process those equations that are directly involved with the event being handled. In contrast, ModelicaDEVS needs to visit *all* equations of *all* blocks whenever an event takes place. Which variables are to be updated in each case is decided by Boolean expressions associated with the various when-statements.

Yet the true difference in speed has probably more to do with the event handling itself. Dymola has been designed for optimal speed in the simulation of continuous models and for optimal robustness in handling hybrid models.

The algorithms implemented in Dymola for robust event handling are important in the context of hybrid modelling. In the context of a pure discrete-event simulation, these algorithms are an overkill. For example, in a pure discrete-event simulation there is no need for iteration after each event to determine a new consistent set of initial conditions. In Dymola, many variables are being stored internally in order to allow LSODAR to integrate continuous state equations correctly across discontinuities. In a pure discrete-event simulation, variables need to be stored for output only.

4.2 Mixed Systems

Mixed systems contain both Dymola and Modelica blocks. Figure 9 shows an example of a simple electrical circuit modelled in Dymola, and in a mixed version with a ModelicaDEVS capacitor. Figure 10 illus-

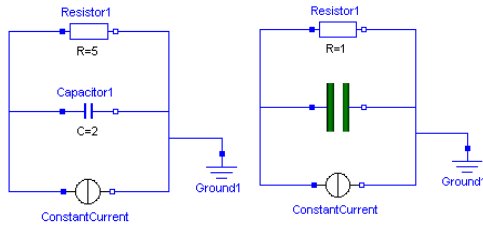


Figure 9: Two versions (Dymola and Dymola/ModelicaDEVS) of a simple electrical circuit.

trates the implementation of the ModelicaDEVS capacitor. On its outside, this block looks like a normal electrical Dymola component, but internally it consists of ModelicaDEVS blocks that model the behaviour of a capacitor: The Gain block multiplies the incoming signal by the value of $\frac{1}{C}$, where C is specified by a parameter, and passes it on to the Interpolator. Taken as a whole, the ModelicaDEVS blocks constitute nothing more than the well known capacitor formula $v = \frac{1}{C} \int i dt$.

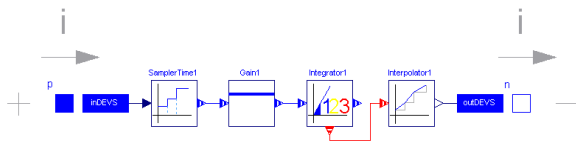


Figure 10: The internal structure of the ModelicaDEVS capacitor.

Unfortunately, it is not as straightforward as it may seem at first glance to replace a component from the Dymola standard electrical library by its ModelicaDEVS equivalent: since the electrical components do not assume a certain data flow direction (they are described by acausal equations), whereas the ModelicaDEVS components do (DEVS components feature input and output ports), the ModelicaDEVS capacitor must turn acausal equations into causal ones. It assumes the capacitive current i to be given, and hence computes the capacitive voltage v . Note that such a capacitor would not work anymore correctly if we were to connect it to a voltage source instead of a current source.

An even more severe problem is caused by the SamplerTime block applying the `der()` operator to the signal that it receives through its input port:

```

du=der(u);

when sample(start,period) then
  yVal[1]=u;
  yVal[2]=if method>1 then du else 0;
  yVal[3]=if method>2 then der(du) else 0;
end when;
    
```

Given that the input of the SamplerTime block depends algebraically on the output of the Interpolator in the DEVS capacitor, Dymola would have to differentiate discrete variables, which it is unable to do.

An attempt to solve this problem was made using Dymola’s “User specified Derivatives” feature described in the Dymola User’s Manual [3]: functions for the first and second derivatives have been inserted into the Interpolator, but due to unknown reasons, this did not resolve the issue either.

In order to be able to perform mixed simulations nonetheless, another trick has been applied: supplementary to the standard ModelicaDEVS SamplerTime block that uses the Modelica `der()` operator, an additional block has been programmed: the SamplerTimeNumerical block avoids the problem caused by the `der()` operator by means of the `delay()` function that is used to differentiate the input variable numerically. Instead of the first and second derivatives of the input signal, the SamplerTimeNumerical returns a numerical approximation:

```

Du = delay(pre(u),D);
D2u= delay(pre(u),2*D);

yVal[1]= pre(u);
yVal[2]= if method>1 then
  (pre(u)-Du)/D else 0;
yVal[3]= if method>2 then
  (pre(u)-2*Du+D2u)/(D*D) else 0;
    
```

Using the new sampler block, the mixed simulation could be carried out without any problems, and the results differ only slightly from the simulation with conventional Dymola components (see Figure 11).

4.3 Hybrid Systems

Hybrid systems contain mixed integration methods: standard Modelica integrators and ModelicaDEVS Integrator blocks. An example of a hybrid system is for instance an electrical circuit with at least one ModelicaDEVS capacitor/inductor (using the ModelicaDEVS Integrator block) and at least one Dymola capacitor/inductor (using the Modelica `der()` operator). The flyback converter of Section 4.1, where the capacitor in the secondary winding is replaced by an equivalent ModelicaDEVS capacitor, may serve as an

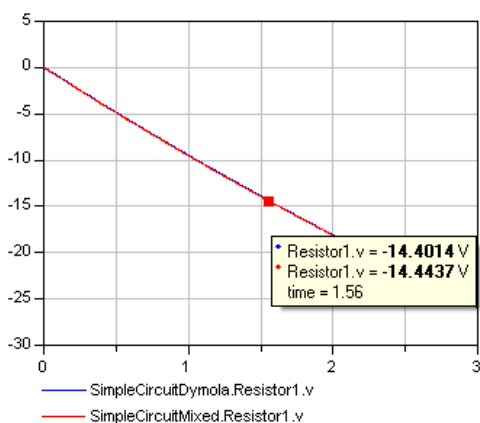


Figure 11: Standard Dymola (blue) and mixed (red) simulation of the simple electrical circuit (Figure 9).

example of a hybrid system.

Note that the ModelicaDEVS capacitor applies numerical differentiation in order not to obtain “DAE index reduction” error messages (see previous section).

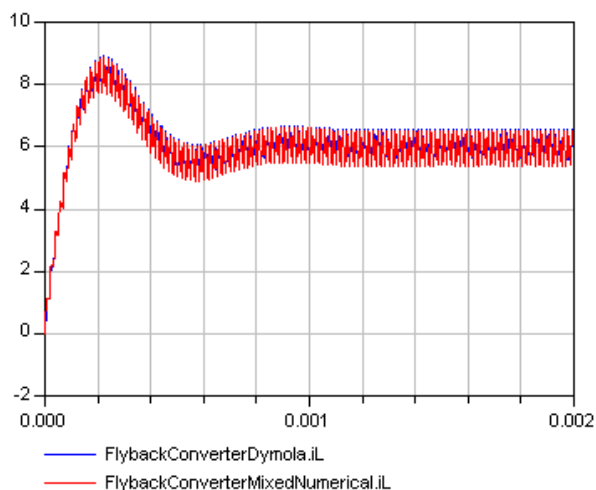


Figure 12: Standard Dymola (blue) and mixed (red) simulation of the flyback converter.

Figure 12 shows the output of the mixed simulation compared to the result of the standard Dymola simulation. Just as it was the case with the simpler example of Section 4.2, the output of the hybrid simulation differs only slightly from the Dymola simulation. Thus, it is also possible to perform not only accurate⁵ mixed simulations, but also hybrid simulations.

⁵Note that due to the numerical differentiation used in the SamplerTimeNumerical block, the result is not as accurate as if analytical differentiation had been used. However, the accuracy is sufficient for most purposes, and also adjustable through selection of the width parameter, D.

5 Conclusions

A new Dymola/Modelica library implementing a number of Quantised State System (QSS) simulation algorithms has been presented. ModelicaDEVS duplicates the capabilities of PowerDEVS. The graphical user interfaces of both tools are practically identical. However, the underlying simulators are very different. Whereas PowerDEVS implements Zeigler’s hierarchical DEVS simulator, ModelicaDEVS operates on simultaneous equations and synchronous information flows.

The embedding of ModelicaDEVS within the Dymola/Modelica environment enables users to mix DEVS models with other modelling methodologies that are supported by Dymola and for which Dymola offers software libraries.

Unfortunately, ModelicaDEVS is much less efficient in run-time performance than PowerDEVS. The loss of run-time efficiency is probably caused by Dymola’s event handling algorithms that have been designed for optimal robustness in the context of hybrid system simulation rather than run-time efficiency in the context of pure discrete-event system simulation.

Although ModelicaDEVS offers a full implementation of a DEVS kernel and can therefore be used for the simulation of arbitrary discrete-event systems, the modelling blocks that have been made available so far in ModelicaDEVS are geared towards the simulation of continuous systems using QSS algorithms.

References

- [1] Beltrame, T. (2006), *Design and Development of a Dymola/Modelica Library for Discrete Event-oriented Systems using DEVS Methodology*, MS Thesis, Institute of Computational Science, ETH Zurich, Switzerland.
- [2] Cellier, F.E. and E. Kofman (2006), *Continuous System Simulation*, Springer-Verlag, New York.
- [3] Dynasim AB (2006), *Dymola Users’ Manual, Version 6.0*, Lund, Sweden.
- [4] Fritzson, P. (2004), *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-Interscience, New York.
- [5] Glaser, J.S., F.E. Cellier, and A.F. Witulski (1995), “Object-Oriented Switching Power Con-

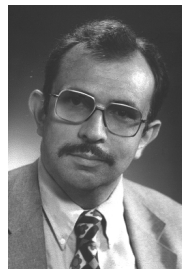
verter Modeling Using Dymola With Event-Handling,” *Proc. OOS’95, SCS Object-Oriented Simulation Conference*, Las Vegas, NV, pp.141-146.

- [6] Kofman, E. and S. Junco (2001), “Quantised State Systems: A DEVS Approach for Continuous Systems Simulation,” *Transactions of SCS*, **18**(3), pp.123-132.
- [7] Kofman, E., “A Second Order Approximation for DEVS Simulation of Continuous Systems,” *Simulation*, **78**(2), pp.76-89.
- [8] Kofman, E., M. Lapadula, and E. Pagliero (2003), *PowerDEVS: A DEVS-based Environment for Hybrid System Modeling and Simulation*, Technical Report LSD0306, LSD, Universidad Nacional de Rosario, Argentina.
- [9] Kofman, E., “A Third Order Discrete Event Method for Continuous System Simulation,” *Latin American Applied Research*, **36**(2), pp.101-108.
- [10] Otter, M., H. Emqvist, and S.E. Mattsson (1999), “Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle,” *CACSD’99, IEEE Symposium on Computer-Aided Control System Design*, Hawaii, pp.151-157.
- [11] Zeigler, B.P. (1976), *Theory of Modeling and Simulation*, John Wiley & Sons, New York.
- [12] Zeigler, B.P. (1984), *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, London.
- [13] Zeigler, B.P. and J.S. Lee (1998), “Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment,” *SPIE Proceedings*, Vol. 3369, pp.49-58.

Biographies



Tamara Beltrame received her MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. She recently started working at VTT (Finland), where she deals with problems of simulation aided automation testing.



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 1d

Mechanical Systems and Applications 1

The DLR FlexibleBodies library to model large motions of beams and of flexible bodies exported from finite element programs

Andreas Heckmann*, Martin Otter*, Stefan Dietz[◇] and José Díaz López[‡]
*German Aerospace Center (DLR), Institute of Robotics and Mechatronics
Oberpfaffenhofen, 82234 Wessling, Germany
[◇]INTEC GmbH, Argelsrieder Feld 13, 82234 Wessling, Germany
[‡]Dynasim AB, Ideon Research Park, SE-223 70 Lund, Sweden

Abstract

The new DLR FlexibleBodies library enables and supports the object-oriented and mathematically efficient modelling of flexible bodies as components of multi-body and of arbitrary physical systems. It provides Modelica model classes to model (a) beams and (b) general flexible bodies exported from finite element programs. The motion of a flexible structure is defined by superposition of a in general large, non-linear motion of a reference frame with small elastic deformations. This paper gives an overview on the background, concepts and ideas on which the library is based and how the Modelica user may take advantage of it.

Keywords: Flexible body, modal representation, standard input data (SID), floating frame of reference

1 Introduction

The DLR FlexibleBodies library is based on the "Standard Input Data of flexible bodies" (SID) which is an object-oriented data structure that was developed at the DLR-Institute of Robotics and Mechatronics to generally describe the properties of elastic bodies, see [13]. It facilitates the use of data which may originate from a finite element description or from continuum models and has been used by various multibody codes, especially by SIMPACK [10], in industrial applications since the early 1990's. The implementation of this general, stable and well-established approach offers new possibilities for multiphysical modelling tasks in Modelica and is nevertheless open for further development and improvements, e.g. concerning multifield problems [6].

2 Modelling Capabilities

The DLR FlexibleBodies library is a commercial Modelica package. It provides two basic Modelica model classes: the Beam model and a general ModalBody model, see Figure 1.

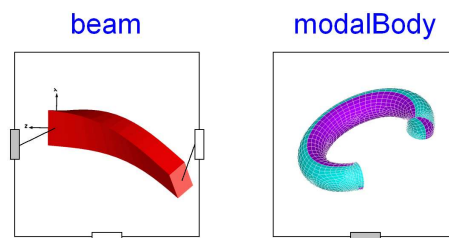


Figure 1: Icons of Beam and ModalBody models.

If a Beam object is instantiated, a dialogue menu, see Fig. 2, supports the definition of the geometrical and physical properties of a straight, homogenous and isotropic beam. For the specification of the cross section an additional menu offers predefined cross section profiles such as tubes, U-pipes or T-beams, see Fig. 3. The parameters defined there are also used for animation purposes. The choice *general* in Fig. 3 enables the direct input of the mechanical essentials, i.e. the geometrical moments of inertia of the cross section.

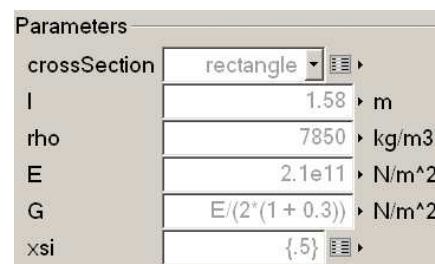


Figure 2: Cutout of the user interface to specify parameters of the Beam model.

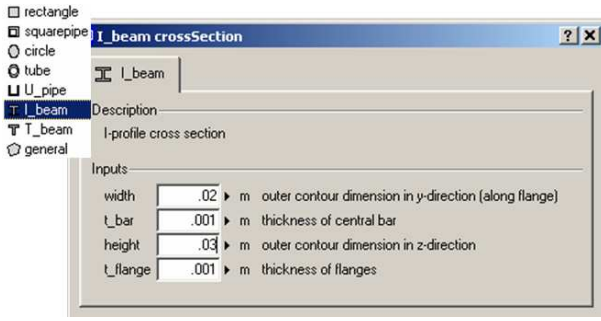


Figure 3: Input for predefined cross section profiles of the Beam model.

The beam model takes bending in two planes, lengthening and torsional deflections into account. The deformations are discretised by eigenforms, i.e. the analytical solutions of the eigenvalue problem of the Euler-Bernoulli beam, see Sec. 3.

Only those eigenforms and eigenfrequencies respectively that are specified by their ordinal numbers are considered in the model. In Fig. 4 the first, third and fifth eigenmode are selected, whereas a damping coefficient of 0.02 is assigned to the first, 0.03 to the second and 0.01 to the third eigenfrequency. This feature gives the experienced modeller the possibility to reduce the number of degrees of freedoms, since some eigenmodes might not contribute to the modelling problem.

Additionally, appropriate boundary conditions that constrain the deformation field of the beam with respect to its floating frame of reference have to be defined. As a general rule, the boundary condition should correspond to the joint where the beam is attached to.

If a ModalBody is instantiated, a file name has to be specified by the user in which the SID structure is present. This SID-file has to be generated in a prepro-

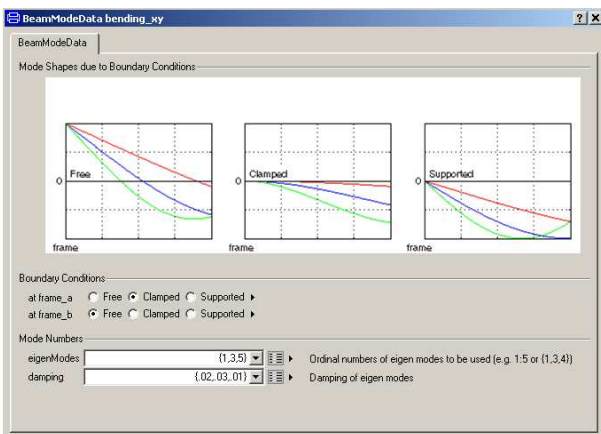


Figure 4: Menu to specify modes of the beam model.

cessing step from finite element data of the body. We recommend the preprocessor FEMBS, see Sec. 4 and [5], from the DLR spin-off company INTEC GmbH that is capable to prepare data from the FE-programs **ABAQUS, ANSYS, MSC.Nastran, NX Nastran, I-DEAS, PERMAS**.

For both the Beam and the ModalBody model the frame connectors of the Modelica.Mechanics.MultiBody library are used to define the connection of the flexible body instance to other system components such as joints, force or sensor elements. Besides the two frames at both ends an arbitrary number of intermediate frames may be defined as parameters of the Beam model. E.g. the parameter $xsi=\{0.5\}$ in Figure 2 specifies an additional frame at the center of the beam. The ModalBody model has one vector of frame connectors that are associated with nodes of the finite element model.

3 Mechanical Background

The mechanical description is based on the floating frame of reference approach, i.e. the absolute position $\mathbf{r} = \mathbf{r}(\mathbf{c}, t)$ of a specific body particle is subdivided into three parts: the position vector $\mathbf{r}_R = \mathbf{r}_R(t)$ to the body's reference frame, the initial position of the body particle within the body's reference frame, i.e. the Lagrange coordinate $\mathbf{c} \neq \mathbf{c}(t)$, and the elastic displacement $\mathbf{u}(\mathbf{c}, t)$:

$$\mathbf{r} = \mathbf{r}_R + \mathbf{c} + \mathbf{u}, \quad (1)$$

where all terms are resolved w.r.t. the body's floating frame of reference (R). That's why the angular velocity of the reference frame $\boldsymbol{\omega}_R$ have to be taken in account when the kinematic quantities velocity \mathbf{v} and acceleration \mathbf{a} of a particle are derived:

$$\mathbf{v} = \dot{\boldsymbol{\omega}}_R \mathbf{r} + \dot{\mathbf{r}} = \mathbf{v}_R + \dot{\boldsymbol{\omega}}_R (\mathbf{c} + \mathbf{u}) + \dot{\mathbf{u}}, \quad (2)$$

$$\mathbf{a} = \mathbf{a}_R + (\ddot{\boldsymbol{\omega}}_R + \dot{\boldsymbol{\omega}}_R \boldsymbol{\omega}_R) (\mathbf{c} + \mathbf{u}) + 2\dot{\boldsymbol{\omega}}_R \dot{\mathbf{u}} + \ddot{\mathbf{u}}, \quad (3)$$

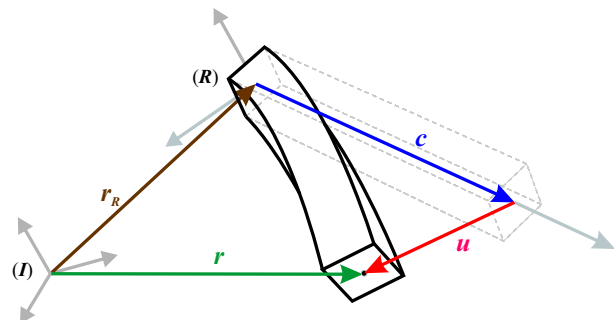


Figure 5: Vector chain of the floating frame of reference.

where the $(\tilde{\cdot})$ -operator is used to replace the vector cross product using the appropriate skew-symmetric matrix instead so that e.g. the identity $\boldsymbol{\omega} \times \mathbf{c} = \tilde{\boldsymbol{\omega}} \mathbf{c}$ holds.

The decomposition in (1) makes it possible to superimpose a large non-linear overall motion of the reference frame with small elastic deformations.

The displacement field is approximated by a **second order** Taylor expansion with space-dependent mode shapes $\Phi(\mathbf{c}) \in \mathbb{R}^{3,n}$, $\Phi_x(\mathbf{c}), \Phi_y(\mathbf{c}), \Phi_z(\mathbf{c}) \in \mathbb{R}^{n,n}$ and time-dependent modal amplitudes $\mathbf{q}(t) \in \mathbb{R}^n$ [13]:

$$\mathbf{u} = \Phi \mathbf{q} + \frac{1}{2} \begin{bmatrix} \mathbf{q}^T \Phi_x \\ \mathbf{q}^T \Phi_y \\ \mathbf{q}^T \Phi_z \end{bmatrix} \mathbf{q}. \quad (4)$$

The focus of the second order expansion is not to depict large deformations e.g. for crash analysis but the incorporation of stress stiffening and softening effects, e.g. the weak bending behaviour of a slender beam under the influence of a large axial thrust force, see Sec. 6.2.

The kinematic quantities together with the vector of applied forces \mathbf{f}_e are inserted into Jourdain's principle of virtual power:

$$\delta \mathbf{v}^T \int_{body} (d\mathbf{f}_e - \mathbf{a} dm) = 0. \quad (5)$$

Subsequently, the equations of motion of an unconstrained flexible body are formulated neglecting deflection terms of higher than first order [13, (38)]:

$$\begin{pmatrix} m\mathbf{I}_3 & & sym. \\ m\tilde{\mathbf{d}}_{CM} & \mathbf{J} & \\ \mathbf{C}_t & \mathbf{C}_r & \mathbf{M}_e \end{pmatrix} \begin{bmatrix} \mathbf{a}_R \\ \dot{\boldsymbol{\omega}}_R \\ \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{K}_e \mathbf{q} + \mathbf{D}_e \dot{\mathbf{q}} \end{bmatrix} + \mathbf{h}_e, \quad (6)$$

where the following quantities and symbols appear:

m	body mass
\mathbf{I}_3	3×3 identity matrix
$\mathbf{d}_{CM}(\mathbf{q})$	position of center of mass
$\mathbf{J}(\mathbf{q})$	inertia tensor
$\mathbf{C}_t(\mathbf{q})$	inertia coupling matrix
$\mathbf{C}_r(\mathbf{q})$	inertia coupling matrix
$\mathbf{h}_\omega(\boldsymbol{\omega}, \mathbf{q}, \dot{\mathbf{q}})$	gyroscopic and centripetal forces
\mathbf{h}_e	external forces
\mathbf{M}_e	structural mass matrix
\mathbf{K}_e	structural stiffness matrix
\mathbf{D}_e	structural damping matrix

If, for the sake of demonstration, the body is assumed to be rigid, those rows and columns in (6) vanish that are associated with the generalised elastic acceleration $\dot{\mathbf{q}}$. Since (6) is formulated in terms of the translatory and angular acceleration of the floating frame of reference, such reduction leads to the classical Newton-Euler equations of a rigid body. Therefore, SHABANA calls (6) the generalised Newton-Euler equations of an unconstrained deformable body in [11, Sec. 5.5].

On the other hand, if the motion of the reference frame is constrained to be zero, (6) is reduced to the classical structural equation (see Sec. 6.2 and (12) for the definition of \mathbf{f}_q):

$$\mathbf{M}_e \ddot{\mathbf{q}} + \mathbf{D}_e \dot{\mathbf{q}} + \mathbf{K}_e \mathbf{q} = \mathbf{f}_q. \quad (7)$$

Applying the classical deformation assumptions of RAYLEIGH and BERNOULLI, it is possible to describe the displacement field of a beam up to second order terms analytically, see [2, (4.104)]:

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} \int_0^x (v'^2 + w'^2) d\bar{x} \\ -\int_0^x \int_0^{\bar{x}} \theta w'' d\bar{x} d\bar{x} + \int_0^x u' v' d\bar{x} \\ \int_0^x \int_0^{\bar{x}} \theta v'' d\bar{x} d\bar{x} + \int_0^x u' w' d\bar{x} \end{bmatrix}, \quad (8)$$

where u denotes the lengthening deformation, v the bending deflection in xy -plane, w the bending deflection in xz -plane and θ the torsional deformation of a point on the beam's x -axis. The $(\cdot)'$ -operation complies with the partial derivation w.r.t. the x -coordinate.

Eq. (8) may be formulated in the manner of (4), if a Rayleigh-Ritz approach such as

$$v = \Theta_v \mathbf{q}_v, \quad (9)$$

is made not only for v but for all four deformation types u, v, w and θ . The deformation state of the beam may then be characterised by $\mathbf{q} = [\mathbf{q}_u^T \mathbf{q}_v^T \mathbf{q}_w^T \mathbf{q}_\theta^T]^T$.

The DLR FlexibleBodies library uses the analytical solutions Θ_i of the spatial problem to the eigenvalue $\tau_i l$ of an Euler-Bernoulli beam with length l :

$$\Theta_i = \begin{bmatrix} \cosh(\tau_i x) \\ \sinh(\tau_i x) \\ \cos(\tau_i x) \\ \sin(\tau_i x) \end{bmatrix}^T \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}_i \quad (10)$$

to form a set of spatial shape functions for each of the deformation coordinates. Hereby, c_1 to c_4 represent

constants associated to specific boundary conditions, see [12, Ch. V].

To summarise, the Beam model class is based on an analytical continuum description of beams via (8) to (10). Contrary, the ModalBody class is supposed to represent bodies with an arbitrary geometry derived from a finite element description. The FE-model of the body is transformed and utilised in such a way that (1) to (6) also hold for the ModalBody class.

We recommend the use of the pre-processor FEMBS from INTEC GmbH that enables this transformation of FE-data as a reliable and sophisticated process, controlled via its own graphical user interface.

4 FEMBS an Interface between MBS and FEA-tools

FEMBS the time-tested interface between the multi-body system code SIMPACK and the most important finite element codes can also be used as a pre-processor for Modelica.

The number of degrees of freedom of multi-body systems is small, when compared with finite element models. Therefore, the main task of flexible body integration is the reduction of the number of degrees of freedom. Currently, this is to be done in two steps.

First, standard reduction schemes like Guyan reduction or Craigh-Bampton reduction are used to export the model from the finite element code into FEMBS's flexible body input file.

Reduction in finite element codes is performed based upon a user defined set of degrees of freedom at nodes which are to be retained. When using the Craigh-Bampton method proper dynamic behaviour of the reduced finite element model can be guaranteed within the frequency range of the application. This additionally requires the specification of a number of so-called dynamic degrees of freedom. They are the eigenmodes of the flexible body, whose retained degrees of freedom were fixed by constraints.

Once the time consuming reduction has been finished the finite element model has less than 1500 degrees of freedom. This first reduction step is to be done in order to keep FEMBS's flexible body input file small. The contents of the *FEMBS* input file are:

- The mass and stiffness matrix of the reduced model.
- the retained nodes

- The eigenmodes which are to be obtained by a modal analysis of the reduced finite element model.
- Geometric stiffening matrices which are to be obtained by extra static analyses of the reduced finite element model. These analyses consider the pre-stresses which are relevant for the application.
- The mesh of finite element model. This data is taken from the FEA input file that contains the elements and nodes of the original, non-reduced model.

Based on this data, the second reduction step [9] in FEMBS is to be performed by selecting modes for multi-body simulation. The user is recommended to select all eigenmodes which correspond to the frequency range which is important for the current application.

Additionally the user should calculate so-called frequency response modes in FEMBS in order to improve the accuracy of the modal representation [3]. Frequency response modes represent local deformations which may occur near the attachment points of the flexible body, where it is connected by force elements, constraints and joints with the surrounding multi-body system.

To generate the frequency response modes (11) in FEMBS the user selects first the nodes and then the directions of the forces and moments which may be transmitted by the force elements or constraints. The frequency response modes \mathbf{u}_i

$$(\mathbf{K} - \Omega^2 \mathbf{M}) \mathbf{u}_i = \mathbf{p}_i \quad (11)$$

are based upon the unit load cases \mathbf{p}_i for each coupled degree of freedom i , the mass matrix \mathbf{M} the stiffness matrix \mathbf{K} and an excitation frequency Ω which is set to the half of the minimum eigenfrequency of interest.

Frequently, only a subset of the frequency response modes has significant influence on the flexible body deformation. The superfluous frequency response modes may be detected by their large frequencies, which follow from a modal analysis which is automatically performed in FEMBS. Thus, FEMBS can automatically select the important frequency response modes based on a cut-off frequency, whose default value is five times greater than the maximum frequency of interest.

As described, the finite element model with say millions of degrees of freedom is transformed to a

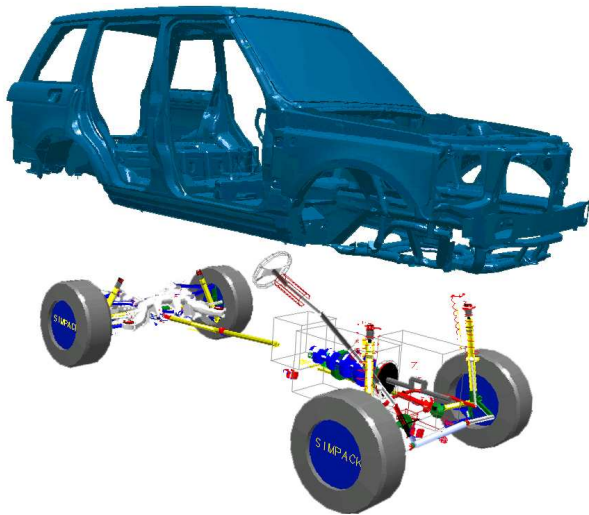


Figure 6: The multi-body system model with the flexible chassis

modal representation of typically about fifty to hundred degrees of freedom. This modal representation is stored in the standard input data file [9] that is input for SIMPACK and Modelica, respectively.

The static deflection of many different flexible bodies such as railway and automotive car bodies, chassis, engine parts, subframes etc. was calculated in the multibody system code SIMPACK. They were compared with the results of static analyses performed in the finite element code based upon the original, non-reduced model. For the static deflection a difference less than one percent can be expected between the results of the finite element model and the multi-body system. Also the eigenfrequencies of the flexible body calculated in SIMPACK are very close to the eigenfrequencies, which are obtained by corresponding finite element analyses.

Thus FEMBS provides efficient and accurate input for multi-body system analyses.

Detailed FEM models of a sport utility vehicle's chassis and also its front and rear subframes were integrated into the multi-body system model which was created using SIMPACK for comfort analyses, see Figure 6. The chassis consisted of about 2.4 millions degrees of freedom. All finite element models were dynamically reduced within NASTRAN. Different sets with 30 to 40 modes consisting of eigenmodes and frequency response modes were used for multi-body system analyses. CAD files of the finite element mesh were also generated by FEMBS and used for the graphical representation.

Fig. 7 shows the good correlation of the model to measured data in the frequency domain up to 25 Hz.

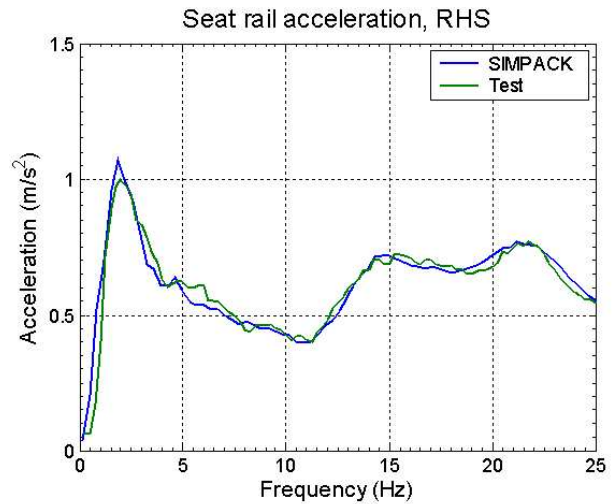


Figure 7: Correlation of the model to measured data: Seat rail acceleration at the right hand side

5 Animation of Modal Bodies

In this section, we describe briefly some visualisation aspects of the ModalBody. The visualiser object used in FlexibleBodies library has been designed to meet the specific requirements of flexible bodies animation in MBS context. It can be accessed from a Modelica model via a built-in function provided by the Modelica modelling and simulation environment Dymola [4].

5.1 Interpolation scenario

Consider a body Ω . The typical animation scenario for Ω is depicted in Fig. 8. As stated in (1), the numerical simulation of this modal body provides the position vector \mathbf{r}_R and the orientation of the body's reference frame as well as the elastic displacements $\mathbf{u}(\mathbf{c}_i, t)$ of a finite set of nodes \mathbf{c}_i on the body. The set of space points that can be obtained from this information and setting $\mathbf{u}(\mathbf{c}, t) \equiv 0$ is called *simulation points* set.

On the other hand, the preprocessor FEMBS provides a Wavefront file [14] that contains the mesh definition of the FE description in the undeformed state with respect to the reference frame of the body. This data is the basis for the flexible body animation. The set of node points of this description is called *animation points* in this section.

The basic interpolation problem is discussed at hand of the simple rectangular plate shown in Fig. 8. The animation set consists of the points in the grid. The simulation points are only the four corners of Ω , that is,

$$\mathbf{c}_1 = (0, 0, 0), \mathbf{c}_2 = (1, 0, 0),$$

$$\mathbf{c}_3 = (0, 1.3, 0), \mathbf{c}_4 = (1, 1.3, 0)$$

and we define as deformation field $\mathbf{u}(\mathbf{c}_i, t)$ the following

$$\mathbf{u}_1 = (0, 0, 0), \mathbf{u}_2 = (0, 0, 0),$$

$$\mathbf{u}_3 = (0, 0, 0), \mathbf{u}_4 = (0, 0, \sin 20\pi t)$$

This is the typical scenario for animation of flexible bodies. The discretisation of a flexible body, in the general case, ends up with a set of nodes such that the error of the FE discretisation is minimised. Then, a modal analysis is performed. For some special geometries, the resulting modes can be deduced in closed form on \mathbf{c} , but it is a difficult task in general, [9]. That is why in general, each mode is defined in discretised form, i.e. by the related displacements that are given at the node points only.

With these constraints in mind, the visualiser used in FlexibleBodies meets the following design criteria

- The solution of the equations of motion results in the displacements \mathbf{u}_i at simulation points \mathbf{c}_i for $t \geq t_0$. The visualiser has to determine the absolute position of the animation points whereas the instantaneous positions of the simulation points and the positions the animation points in the undeformed configuration are given.
- The topological information is just provided for the animation points. That is, the simulation point set is just a set of non-structured points in \mathcal{R}^3 .
- The ModalBodyVisualizer uses a special interpolation technique that results in visually appealing images under the assumption that the simulation and animation point sets describe an elastic deformation field. No general interpolation technique (like polynomial interpolation or splines) is used. Instead, an approach inspired by potential theory applied to elasticity is used, [8].

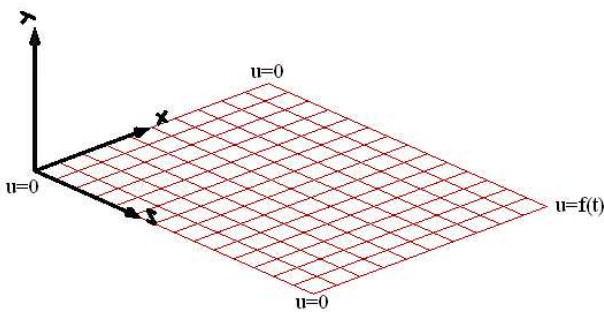


Figure 8: Interpolation problem setup with the rectangle plate

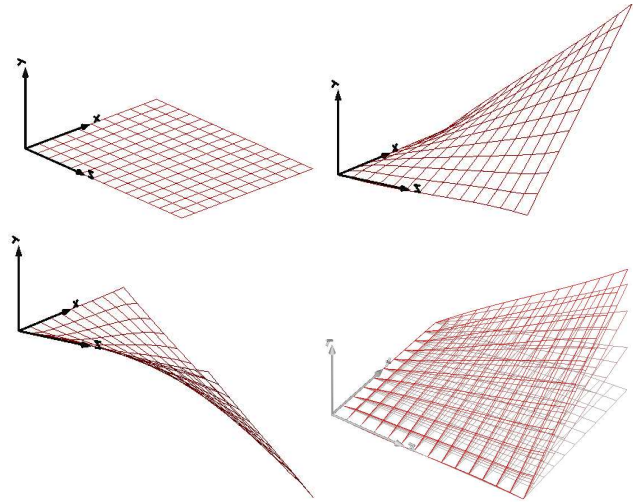


Figure 9: Interpolation result using the ModalBodyVisualizer

5.2 ModalBodyVisualizer in FlexibleBodies library

To show how ModalBodyVisualizer works, we applied it to Ω in the setup described before. The results are depicted in Fig. 9. The body Ω is originally defined in the plane xy , The outer most right point (\mathbf{c}_4) is now moved up and down.

The initial position is depicted in the left upper figure. Then, when the elongation is maximally positive in the y -axis in the right upper figure. After some time, the deformation is maximally negative, depicted in the left lower figure. Finally, and to give some feeling about the animation, we present a frame with 4 past positions.

In this example, 165 nodes are interpolated from the information in the four corners of the rectangle. More general interpolation techniques are not so suitable for elastic deformation fields. The main reason is the lack of smoothness and loss of connectivity in the single body, making artificial cracks, peaks or weird artifacts to be shown as part of the simulation.

Taking just the simulation point set for animation can make hard to imagine the body behaviour, so a larger set of animation points is needed for good visualisation.

6 Example Models

The DLR FlexibleBodies library contains several examples that demonstrate the use of the provided capabilities.

6.1 Slider Crank Model

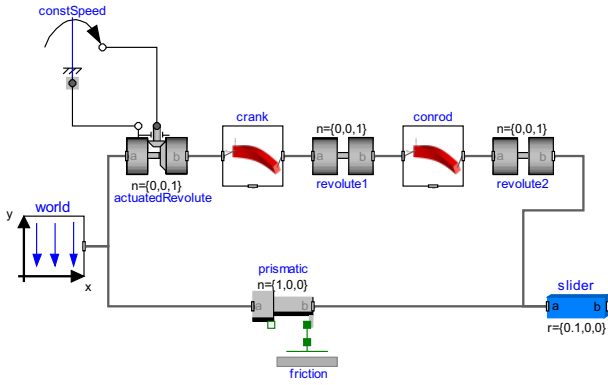


Figure 10: Diagram layer of the slider crank model.

One of these examples is a model of a slider crank mechanism with two beam instances that represent the crank and the conrod. The crank rotates with constant angular velocity.

Gravity forces are applied to all bodies of the system and an additional Coulomb friction force acts on the slider that has one translational degree of freedom.

The bending behaviour of the conrod is represented by one mode that is related to 1 Hz eigenfrequency, while the crank bending mode corresponds to 3.9 Hz eigenfrequency. The model aligns with an example from literature and may therefore also be used for verification purposes [9, Sec. 6.5.5].

Figure 10 depicts the graphical set-up of this model in Modelica. The model 3D animation is shown in Figure 11, where the menu option to show an additional exaggerated displacement field beside the exact in-scale deformations is activated.

Fig. 12 shows the simulation results of this plane, closed-loop mechanism with discontinuously applied friction force. For the integration of the 10-s-scenario 0.671 CPU-s were spent with 10^{-6} integration tolerance on a 1.6 GHz Intel Pentium Laptop with WindowsXP.

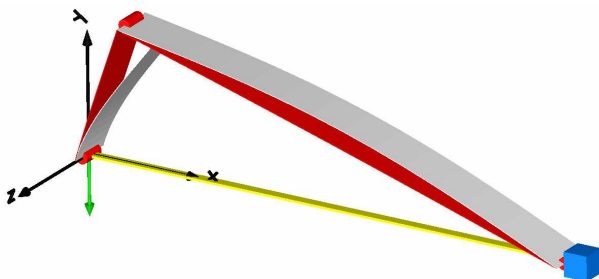


Figure 11: Animation of the slider crank model. The grey animations are scaled versions of the red animations and exaggerate the deformations of the beams.

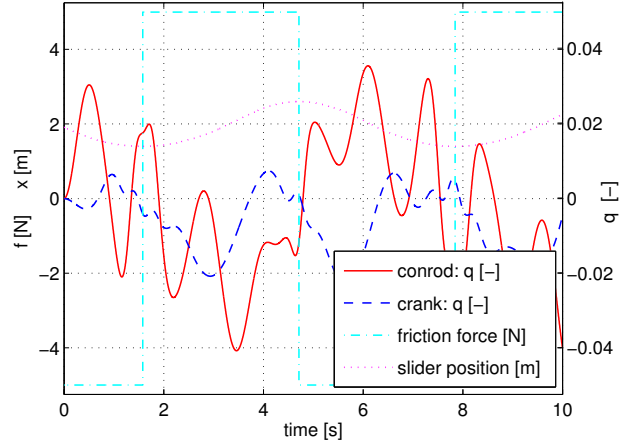


Figure 12: Slider Crank simulation results.

6.2 Buckling of a beam

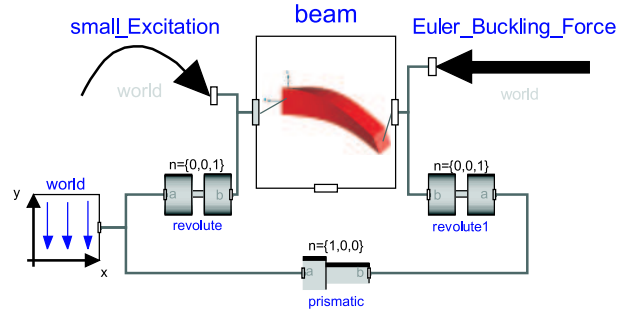


Figure 13: Set-up of the model Beam Buckling.

Considered is a bending beam with 18.8 Hz eigenfrequency that is supported but not clamped on both ends. At one end an external force is applied and is increased linear in time until the classical Euler buckling force (in this case: 16.6 kN) is reached. This is the scenario of the model Beam Buckling of Fig. 13. It is supposed to document that the present approach is capable to cover the bending behaviour of the beam until buckling occurs, see Fig. 14. This requires to include stress stiffening terms in (6) that originate from the second order displacement field description, here.

Consider the physical force vector $\mathbf{f} \in \mathbb{R}^3$ that is applied on the structure at the point \mathbf{c}_f . The equations of the model Beam Buckling then get the form of (7) whereas \mathbf{f}_q has to be defined using (4) and the Jacobian \mathbf{J} :

$$\mathbf{f}_q = \mathbf{J}^T \mathbf{f} \quad (12)$$

$$\text{with } \mathbf{J} := \frac{\partial \mathbf{u}(\mathbf{c}_f, t)}{\partial \mathbf{q}} = \Phi_{\mathbf{c}_f} + \begin{bmatrix} \mathbf{q}^T \Phi_x \\ \mathbf{q}^T \Phi_y \\ \mathbf{q}^T \Phi_z \end{bmatrix}_{\mathbf{c}_f} .$$

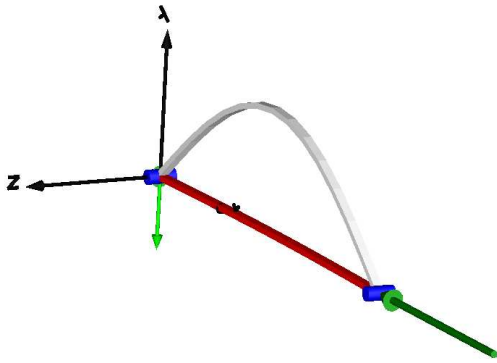


Figure 14: Animation of the model Beam Buckling in the instant the bifurcation occurs.

Consequently, an increasing force f may be additionally amplified by the increasing deflection q , so that the stability of the system may be affected. This is in particular the case for axial loads on slender beams as it is given in this model.

Since the deformation behaviour is not determined beyond the bifurcation point, an additional small harmonic excitation torque with 1 Nm and 1 Hz is applied at the other end at the beam. This is necessary to ensure that the buckling actually occurs and can be animated as it is done in Fig. 14.

The results of the simulation are shown in Fig. 15. At the beginning the harmonic excitation hardly influences the state of the beam. With increasing thrust force the bending behaviour is weakened until the buckling occurs after about 10 s. These results perfectly correspond to the theoretical prediction. The nonlinear characteristics of the beam are reproduced until buckling. However, it should be pointed out that beyond this point the model is no more valid.

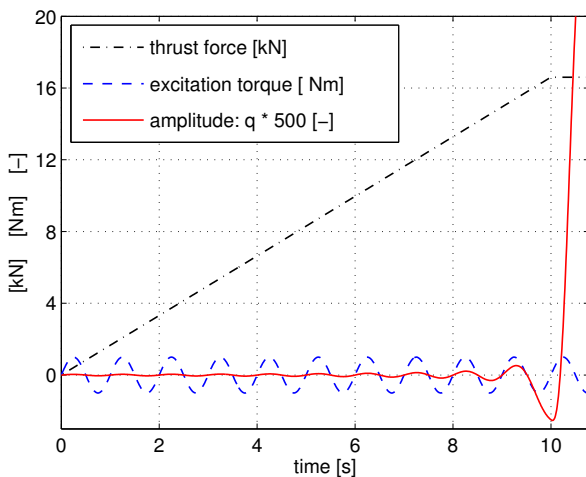


Figure 15: Simulation results: Buckling of a beam.

6.3 Helicopter Rotor

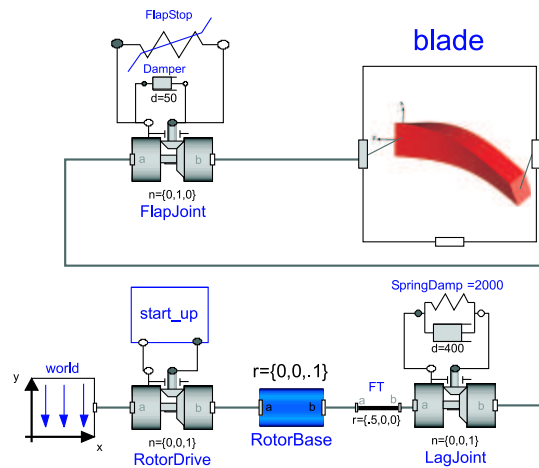


Figure 16: Set-up of the model Helicopter Rotor.

The model Helicopter Rotor in Fig. 16 mainly consists of a rheonom driven, cylindrical rotor base, two joints and one blade. The rotor base rotates around its cylinder axis that coincides with the global z -axis, while the lag joint allows for a rotation around the local z -axis at the outer radius of the rotor base. The flap joint defines a angular motion around the local y -axis at the circumference of the rotor base.

In its initial state the rotor base does not move and the flap stop, a bump stop modelled as a nonlinear spring, applies the torque to counterbalance the gravity of the blade. A linear spring-damper element actuates according to the state of the lag joint. The 6 m long blade is modelled as a flexible beam with 7 bending modes in its xz -plane and 2 bending modes in its xy -plane so that a frequency range up to 270 Hz is covered.

Fig. 17 visualises the initial, static deformation state of the blade which is dominated by the first xz -

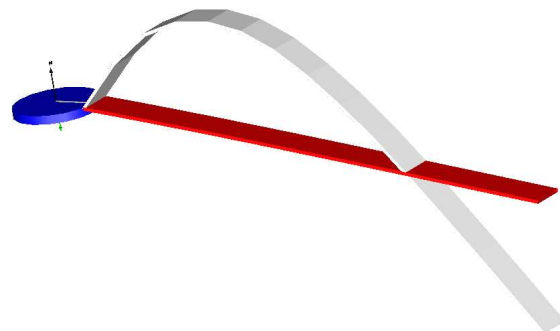


Figure 17: 3D-View: Helicopter Rotor in its initial state with red in-scale and grey exaggerated deflection.

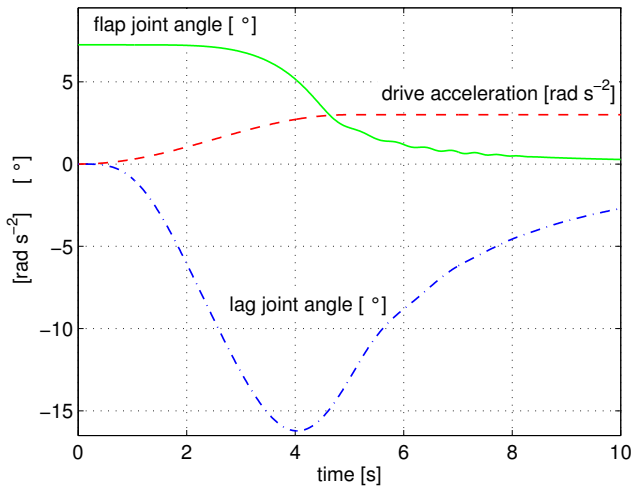


Figure 18: Time plot of joint variables of the model Helicopter Rotor.

bending mode at 7.9 Hz. The design, the geometrical and physical parameters of the model correspond to a typical helicopter configuration of the 1960's, see [7].

The result plot Fig. 18 shows the applied, smooth rotor drive acceleration of the simulated start-up scenario. The flap joint angle tends against zero with increasing rotor rotation so that the blade moves towards its vertical alignment. The instantaneous lag joint angle is associated to the torque applied by the linear spring-damper element which transmits the drive torque to the blade.

Fig. 19 presents the dominance of 7.9 Hz mode at the initial static configuration. The plot of xy -plane bending modes clearly correspond to the lag joint plot which is again due to the torque applied by the lin-

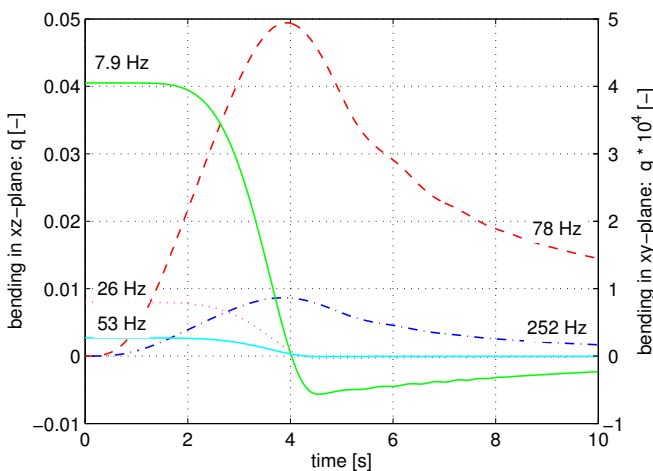


Figure 19: Plot of modal deformation amplitudes $q(t)$ for 2 bending modes in xy -plane (78 Hz, 252 Hz) and 3 bending modes in xz -plane (7.9 Hz, 26 Hz, 53 Hz).

ear spring-damper element. Generally, all modal amplitudes decrease with the increasing angular velocity and clarify its stabilising influence. Note that the stabilising effect again relies on the second order description (4). Neglecting these terms would lead to completely wrong results for this scenario.

6.4 Piston Rod

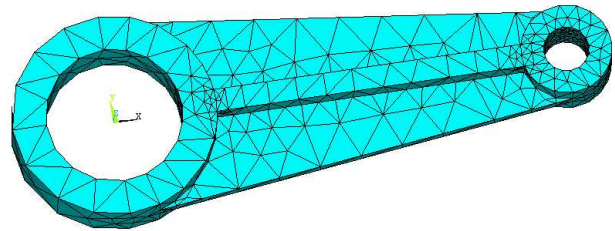


Figure 20: ANSYS finite element model of a piston rod.

In order to demonstrate the use of DLR FlexibleBodies library with body data that originate from finite element data, a simplified piston rod was modelled in Ansys with 2.788 tetrahedral elements of type Solid95 [1] and 15.492 degrees of freedom, see Fig. 20.

The model was preprocessed in FEMBS and three eigenmodes with 809 Hz, 1040 Hz and 1244 Hz eigenfrequency were selected to represent the deformation field of the piston rod on the multibody side. 97 of the

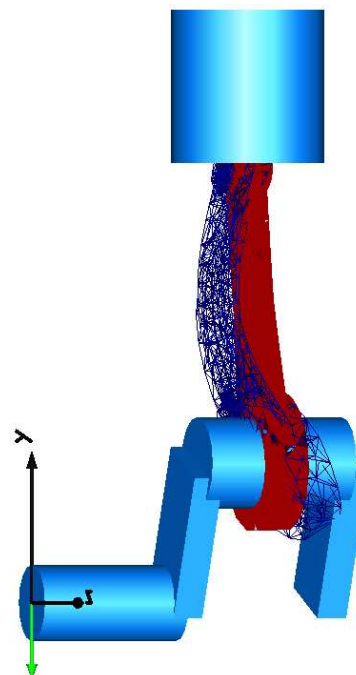


Figure 21: Modelica 3D-View of one cylinder combustion engine with flexible piston rod.

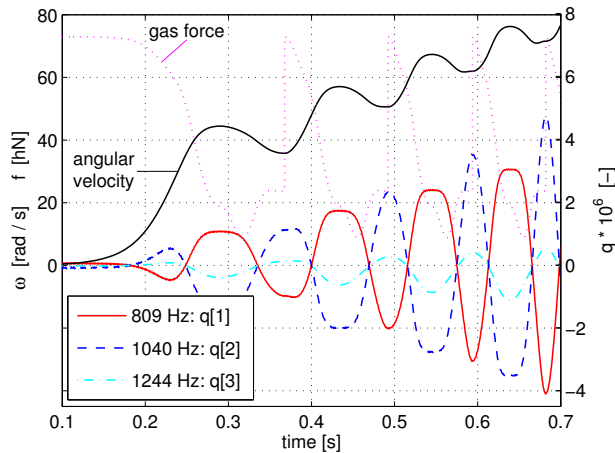


Figure 22: one-cylinder-combustion engine: simulation results.

5164 finite element nodes were selected in *FEMBS* so that the animation scheme has to map the motion of 97 simulation nodes on 5164 animation nodes.

Fig. 21 gives a 3D-View of a one cylinder combustion engine where the flexible piston rod is assembled. The dark-red, solid structure depicts the unscaled deformation state while the blue mesh exaggerate the deflections of the piston rod.

A start-up manoeuvre of this engine has been simulated applying a simplified gas force and no load. The plot of the gas force and the increase of the angular velocity of the crank shaft are given in Fig. 22. For each of the eigenmodes the related modal amplitude as function of time is plotted there as well.

7 Conclusion

This paper presents the new DLR FlexibleBodies library, that is based on the “Standard Input Data” (SID) file format for flexible bodies. The library provides the intrinsic capability to generate this data for beam-like bodies. For more general bodies, this data has to be supplied by an external tool on the basis of an available finite element model. This preprocessing is performed using the *FEMBS*-Software of the *INTEC* GmbH in Wessling, Germany, that supports all major FE programs.

The animation of general flexible bodies is based on new features provided in Dymola. Especially, wavefront files are supported in Dymola and a sophisticated algorithm is included to map the movement of simulation to animation nodes.

Furthermore, the mechanical background and the graphical user interface of the FlexibleBodies library

have been presented. In several application examples, the usage of the model classes *Beam* and *ModalBody* have been demonstrated.

As a final conclusion it may be stated that this library opens new chances for the set-up of multibody models in Modelica since flexible components may now be included and so that future application fields are numerous.

8 Acknowledgements

A first preliminary version of the *ModalBody* model was implemented by Gerhard Schillhuber with advice by Professor Dr. Oskar Wallrapp.

We would like to thank Hans Olsson from Dynasim for helping with some implementation details.

This work was partly funded by the Bayerisches Staatsministerium für Wirtschaft, Verkehr und Technologie which supported the Conceptual Design Laboratory (CDL) project. The authors are grateful for this support.

References

- [1] ANSYS® Release 7.1 Theory Reference, ANSYS, Inc., Canonsburg, USA, 2003.
- [2] H. Bremer and F. Pfeiffer: *Elastische Mehrkörpersysteme*, Teubner-Verlag, Stuttgart, 1992.
- [3] S. Dietz: *Vibration and Fatigue Analysis of Vehicle Systems Using Component Modes*, Fortschritt-Berichte VDI Reihe 12, Nr. 401, VDI-Verlag Düsseldorf, 1999.
- [4] Dynasim: *Dymola* Version 6. Dynasim AB, Lund, Sweden, 2006, <http://www.dynasim.se/>
- [5] *FEMBS*: http://www.simpack.com/downloads/pdf/datasheet_fembs.pdf
- [6] A. Heckmann. *The modal multifield approach in multibody dynamics*. Fortschritt-Berichte VDI Reihe 20, Nr. 398. VDI-Verlag, Düsseldorf, 2005.
- [7] W. Just: *Hubschrauber und Vertikalstartflugzeuge*, Verlag Flugtechnik Stuttgart, Stuttgart, 1963.
- [8] Perdomo, Y. *Reproducing kernels and Potential theory for the Bergman Spaces* Doctorate thesis in Mathematics, Lund University, 2004.

- [9] R. Schwertassek and O. Wallrapp: Dynamik flexibler Mehrkörpersysteme, Vieweg Verlag, Braunschweig, 1999.
- [10] SIMPACK: <http://www.simpack.com>.
- [11] A.A. Shabana: Dynamics of Multibody Systems, Cambridge University Press, Cambridge, 2nd ed., 1998.
- [12] S. Timoshenko: Vibration Problems in Engineering. D. Van Nostrand, Princeton, 1955.
- [13] O. Wallrapp: Standardization of Flexible Body Modeling in Multibody System Codes, Part 1: Definition of Standard Input Data, Mechanics of Structures and Machines 22(3):283-304, 1994.
- [14] Wavefront: <http://www.fileformat.info/format/wavefrontobj/egff.htm>.

3D Flexible Multibody Thin Beam simulation in Modelica with the Finite Element Method

Xabier Murua, Felix Martinez, Aron Pujana, Jon Basurko, Juan Manuel Pagalday
Ikerlan S.Coop.

P. JM. Arizmendiarieta N°2, Mondragón (Basque Country), Spain
{xmurua, Felix.Martinez, APujana, JBasurko, JMPagalday}@ikerlan.es

Abstract

This paper presents the development, simulation and validation of 3 dimensional flexible beam models using *Modelica*. The models are based on finite element method (FEM) application, following mathematical calculations proposed by Shabana [2], and are 3D extensions to the 2D model developed by F. Schiavo [3]. The element formulation is independent of applied boundary conditions, making the element suitable for any 3D multibody simulation.

All models use standard connectors defined in the *Modelica* multibody library, thereby guaranteeing full compatibility with library components. Mathematical modelling details are fully analyzed, indicating motion equation development. Models also feature a graphical interface, and visualization of simulation outcomes using the same 3D environment as the multibody library, providing the user with immediate visual feedback. Finally, models are analyzed and validated by means of selected simulation experiments, with reference to theoretical predictions and comparison with results obtained with commercial FEM code.

Keywords: flexible beam; Multibody; FEM; finite element formulation

1 Introduction

Whilst 2D flexible model development is widely covered by literature [2], its application to 3 Dimensional models normally only goes as far as the initial stages of the mathematical development. This paper presents the application of 2D model redevelopment and implementation [3] to 3D models using *Modelica*. At times shall descriptions made by Schiavo [3] are used.

With the aim of creating a reusable parametric element, is has been considered only the modelling of

one particular scenario: a beam containing two connection points, one at each end.

In this paper the linear elasticity theory for thin beam modelling are considered, ignoring shear deformation effects and assuming uniform cross sectional properties throughout the element length. Cross sectional dimensions compared to element length rigid configuration deflection are also assumed to be small.

Taking into account the object-oriented capability of the *Modelica* language, some parametric features have been implemented in the model, for example: cross sectional beam shape (rectangular or cylindrical), hollow or full section, and mesh length density (from 1 to N elements). This provides a parametric mesh of the element enabling computation of points along the element length and precision-computational cost control.

The implementation of other shape sections can be easily carried out, but are not included in the model in order to simplify the user interface menu.

It has been also developed a model that for importing mass and stiffness matrix values from condensed FE models to 12 degrees of freedom (dof).

2 Degrees of Freedom (dof)

Consider a generic multibody system (Figure 1). The position, in body coordinates, of a point in a specific deformable body is expressed as follows:

$$u = u_0 + u_f \quad (1)$$

where u_0 is the “undeformed” (i.e., rigid) position vector and u_f is the deformation contribution to position (i.e., the deformation field).

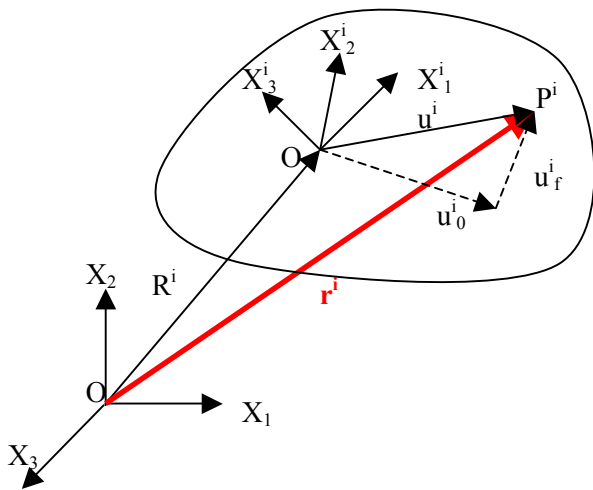


Figure 1: Flexible body reference systems

The mathematical description of a body's generic deformation requires that the deformation field belong to an infinite dimensional functional space, requiring, in turn, an infinite number of deformation degrees of freedom.

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

$$u_f = Sq_f \quad (2)$$

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

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

$$r = R + Au = R + A(u_0 + Sq_f) = R + Au_0 + ASq_f \quad (3)$$

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

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

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

where R and θ represents unreformed body position and orientation angles and q_f is a vector containing flexible degrees of freedom.

3 Motion Equations

The equations are solved using a classical Lagrangian approach. The equation for flexible element motion, in body axes, can be expressed as [2],[3] (a general demonstration to motion equations in [2] and in more detail in [3]):

$$\begin{bmatrix} m_{RR} & \bar{S}_t^T & \bar{S} \\ & I_{\theta\theta} & I_{\theta f} \\ & & m_{ff} \end{bmatrix} \cdot \begin{bmatrix} \ddot{R} \\ \ddot{\alpha} \\ \ddot{q}_f \end{bmatrix} = \begin{bmatrix} O_3 \\ O_3 \\ -K_{ff}q_f \end{bmatrix} + \begin{bmatrix} Q_v^R \\ Q_v^\theta \\ Q_v^f \end{bmatrix} + \begin{bmatrix} \bar{Q}_e^R \\ \bar{Q}_e^\theta \\ \bar{Q}_e^f \end{bmatrix} \quad (5)$$

Equations are valid for a general deformable body, though many of the quantities involved (e.g., the K_{ff} matrix) depend on specific body characteristics such as the shape and material properties, but do not depend on element deformation.

The mass matrix obtained using the formulation of [3], takes the following form:

$$M^i = \begin{bmatrix} M_{RR}^i & M_{R\theta}^i & M_{Rf}^i \\ M_{\theta R}^i & M_{\theta\theta}^i & M_{\theta f}^i \\ M_{fR}^i & M_{f\theta}^i & M_{ff}^i \end{bmatrix} \quad (6)$$

Where mass matrix components (6) are calculated in the following manner, assuming that the body is a 3D elastic continuum, with constant cross-sectional properties, isotropic material behaviour and is perfectly elastic.

$$M_{RR}^i = \int_{V^i} \rho^i dV^i \quad (7)$$

$$M_{R\theta}^i = M_{\theta R}^i{}^T = -A^i \left[\int_{V^i} \rho^i \bar{u}^i dV^i \right] \bar{G}^i \quad (8)$$

$$M_{Rf}^i = A^i \int_{V^i} \rho^i S^i dV^i = A^i \bar{S}^i \quad (9)$$

$$M_{\theta\theta}^i = \int_{V^i} \rho^i (A^i \tilde{u}^i \bar{G}^i)^T (A^i \tilde{u}^i \bar{G}^i) dV^i \quad (10)$$

$$M_{\theta f}^i = \bar{G}^{iT} \int_{V^i} \rho^i \tilde{u}^i S^i dV^i \quad (11)$$

$$M_{ff}^i = \int_{V^i} \rho^i S^{iT} S^i dV^i = \bar{S}_{11}^i + \bar{S}_{22}^i + \bar{S}_{33}^i \quad (12)$$

Where the 3D shape matrix is:

$$S^{ijT} = \begin{bmatrix} 1-\xi & 0 & 0 \\ 6[\xi - (\xi)^2] \eta & 1-3(\xi)^2 + 2(\xi)^3 & 0 \\ 6[\xi - (\xi)^2] \zeta & 0 & 1-3(\xi)^2 + 2(\xi)^3 \\ 0 & -(1-\xi)\eta\zeta & -(1-\xi)\eta\eta \\ [1-4\xi+3(\xi)^2] \zeta & 0 & [-\xi+2(\xi)^2 - (\xi)^3] \\ [1-4\xi+3(\xi)^2] \eta & [\xi-2(\xi)^2 + (\xi)^3] & 0 \\ \xi & 0 & 0 \\ 6[-\xi + (\xi)^2] \eta & 3(\xi)^2 - 2(\xi)^3 & 0 \\ 6[-\xi + (\xi)^2] \zeta & 0 & 3(\xi)^2 - 2(\xi)^3 \\ 0 & -l\xi\zeta & l\xi\eta \\ [-2\xi+3(\xi)^2] \zeta & 0 & [(\xi)^2 - (\xi)^3] \\ [-2\xi+3(\xi)^2] \eta & [-(\xi)^2 + (\xi)^3] & 0 \end{bmatrix}^{ij} \quad (13)$$

To be able to perform the calculation at the right beam extremity (point B, Figure 2) it has been supposed that it is calculated exactly at the average point of the element in order to simplify matrices that intervene in the mass matrix calculation.

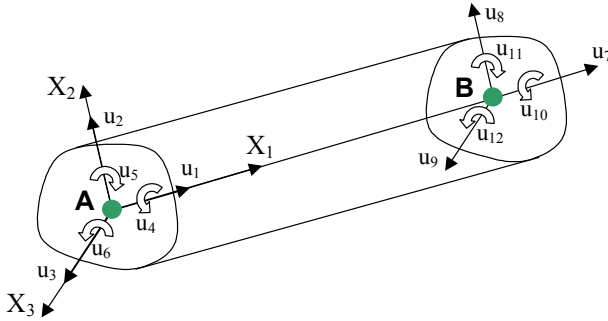


Figure 2: Beam extremities

By means of this assumption the general matrices are simplified applying the following assumption:

$$\xi^{ij} = \frac{x_1^{ij}}{l^{ij}} = 1 \quad (14)$$

$$\eta^{ij} = \frac{x_2^{ij}}{l^{ij}} = 0 \quad (15)$$

$$\zeta^{ij} = \frac{x_3^{ij}}{l^{ij}} = 0 \quad (16)$$

In the end, by using these assumptions, the beam is simplified to a line due to the fact that the length of

the beam and the height and width they are not taken into account.

It also implies that the following terms vanish in the mass matrix formulation (see appendix B)

$$Q_{\eta}^{ij} = \left[\int_a \rho \eta da \right]^{ij} = 0 \quad (17)$$

$$Q_{\zeta}^{ij} = \left[\int_a \rho \zeta da \right]^{ij} = 0 \quad (18)$$

$$I_{\zeta}^{ij} = \left[\int_a \rho (\eta)^2 da \right]^{ij} = 0 \quad (19)$$

$$I_{\eta}^{ij} = \left[\int_a \rho (\zeta)^2 da \right]^{ij} = 0 \quad (20)$$

$$I_{\eta\zeta}^{ij} = \left[\int_a \rho \eta \zeta da \right]^{ij} = 0 \quad (21)$$

Following the procedure described in [3] and the matrix in appendix B, one can verify that the integrals that appear in the expression of S_{kl}^{ij} are given by:

$$\left[\int_V \rho S^T S_i dV \right]^{ij} = \begin{bmatrix} \frac{m}{3} & 0 & 0 & 0 & 0 & 0 & \frac{m}{6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{m}{6} & 0 & 0 & 0 & 0 & 0 & \frac{m}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{ij} \quad (22)$$

$$\left[\int_V \rho S_2^T S_2 dV \right]^{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{13m}{35} & 0 & 0 & 0 & \frac{11ml}{210} & 0 & \frac{9m}{70} & 0 & 0 & 0 & -\frac{13ml}{420} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{11ml}{210} & 0 & 0 & 0 & \frac{m(l)^2}{105} & 0 & \frac{13ml}{420} & 0 & 0 & 0 & -\frac{m(l)^2}{140} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{9m}{70} & 0 & 0 & 0 & \frac{13ml}{420} & 0 & \frac{13m}{35} & 0 & 0 & 0 & -\frac{11ml}{210} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{13ml}{420} & 0 & 0 & 0 & -\frac{m(l)^2}{140} & 0 & -\frac{11ml}{210} & 0 & 0 & 0 & \frac{m(l)^2}{105} \end{bmatrix}^{ij} \quad (23)$$

$$\left[\int_V \rho S_3^T S_3 dV \right]^{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{55} & 0 & -\frac{11ml}{210} & 0 & 0 & \frac{9m}{70} & 0 & \frac{13ml}{420} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{11ml}{210} & 0 & \frac{m(l)^2}{105} & 0 & 0 & -\frac{13ml}{420} & 0 & -\frac{m(l)^2}{140} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{9m}{70} & 0 & -\frac{13ml}{420} & 0 & 0 & \frac{13m}{35} & 0 & \frac{11ml}{210} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{420} & 0 & -\frac{m(l)^2}{140} & 0 & 0 & \frac{11ml}{210} & 0 & \frac{m(l)^2}{105} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{ij} \quad (24)$$

$$\left[\int_V \rho S_1^T S_3 dV \right]^{ij} = \begin{bmatrix} 0 & 0 & \frac{7m}{20} & 0 & -\frac{ml}{20} & 0 & 0 & 0 & \frac{3m}{20} & 0 & -\frac{ml}{30} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3m}{20} & 0 & -\frac{ml}{30} & 0 & 0 & \frac{7m}{20} & 0 & \frac{ml}{20} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{ij} \quad (25)$$

$$\left[\int_V \rho S_2^T S_3 dV \right]^{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{35} & 0 & -\frac{11ml}{210} & 0 & 0 & \frac{9m}{70} & 0 & \frac{13ml}{420} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{11ml}{210} & 0 & -\frac{m(l)^2}{105} & 0 & 0 & \frac{13ml}{420} & 0 & \frac{m(l)^2}{140} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{9m}{70} & 0 & -\frac{13ml}{420} & 0 & 0 & \frac{13m}{35} & 0 & \frac{11ml}{210} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{13ml}{420} & 0 & \frac{m(l)^2}{140} & 0 & 0 & -\frac{11ml}{210} & 0 & -\frac{m(l)^2}{105} & 0 & 0 \end{bmatrix}^{ij} \quad (26)$$

$$\left[\int_V \rho S_1^T S_2 dV \right]^{ij} = \begin{bmatrix} 0 & \frac{7m}{20} & 0 & 0 & 0 & \frac{ml}{20} & 0 & \frac{3m}{20} & 0 & 0 & 0 & -\frac{ml}{30} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{3m}{20} & 0 & 0 & 0 & \frac{ml}{30} & 0 & \frac{7m}{20} & 0 & 0 & 0 & -\frac{ml}{20} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{ij} \quad (27)$$

It can be seen that the mass matrix depends on element deformation, and needs to be re-calculated for each simulation instance.

The speed quadratic vector is a non-linear function of the widespread system of coordinates and speeds and includes the Coriolis effect and the effect of centrifugal forces.

$$\begin{aligned} (Q_v^i)_R &= -A^i \left[(\bar{w}^i)^2 S_i^i + 2\bar{w}^j \bar{S}^j q_f^i \right] \\ (Q_v^i)_\theta &= -2\bar{G}^{iT} \bar{I}_{\theta\theta}^i \bar{w}^i - 2\bar{G}^{iT} \bar{I}_{\theta f}^i q_f^i - \bar{G}^{iT} \bar{I}_{\theta\theta}^i \bar{w}^i \\ (Q_v^i)_f &= -\int_V \rho^i \left\{ S^{iT} \left[(\bar{w}^i)^2 \bar{u}^i + 2\bar{w}^i \bar{u}_f^i \right] \right\} dV^i \end{aligned} \quad (28)$$

3.1 The element point of view

The finite element method is based upon a discretization of the beam into N elements. It is then possible to define the local dimensionless *abscissa* as $\xi=x/l$, where x is the longitudinal local coordinate and l is the element length.

For a single element, the generic equations of motion (5) can be expanded as follows:

$$\bar{u}_{f,el} = \begin{bmatrix} \bar{u}_{f1,el} \\ \bar{u}_{f2,el} \\ \bar{u}_{f3,el} \end{bmatrix} = S_{el} q_{f,el} \quad (29)$$

$$q_{f,el}^T = \begin{bmatrix} q_{f1,el} \\ q_{f2,el} \\ q_{f3,el} \\ q_{f4,el} \\ q_{f5,el} \\ q_{f6,el} \\ q_{f7,el} \\ q_{f8,el} \\ q_{f9,el} \\ q_{f10,el} \\ q_{f11,el} \\ q_{f12,el} \end{bmatrix}$$

$$S_{el} = \begin{bmatrix} S_{el1} \\ S_{el2} \\ S_{el3} \end{bmatrix}$$

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

Figure 2 depicts the element coordinate systems associated with the deformation degrees of freedom: $q_{f1,el}$ and $q_{f7,el}$ are associated with axial compression, $q_{f2,el}$ and $q_{f8,el}$ with transversal displacement, $q_{f3,el}$ and $q_{f9,el}$ with Z axis displacement, $q_{f4,el}$ and $q_{f10,el}$ with beam extremities rotation around the X axis, $q_{f5,el}$ and $q_{f11,el}$ with beam extremities rotation around the Y axis and $q_{f6,el}$ and $q_{f12,el}$ with beam extremity rotation around the Z axis.

3.2 Finite Element Method Equation Assembly

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

Let then m and L be the mass and length of the entire beam, and N the number of elements used, so that $l=L/N$. With \hat{X} indicating the reference system unit vector along the beam axis, the expression of the generic position u_j of a point of element j can be expressed as:

$$\bar{u}_j = \bar{u}_{0j} + S_{el} B_j q_f = [\xi_j l + (j-1)l] \hat{X} + S_{el} B_j q_f \quad (30)$$

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

$$B_j = [O_{6,3(j-1)} \quad I_6 \quad O_{6,3(N-j)}], \forall j=1, \dots, N \quad (31)$$

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

$$q_{f,el_j} = B_j q_f \quad (32)$$

4 Modelica Implementation

The finite element model formulation has been implemented using the *Modelica* language, creating thus a new component, called *FlexBeamFem3D* (Figure 4). The component interfaces are two standard mechanical flanges from the new *MultiBody* library [4]. The choice of connector makes the component fully compatible with the *Modelica Multi-body* library, so that it is possible to directly connect the flexible beam component to the predefined models, such as mechanical constraints (revolute joints, prismatic joints, etc.), parts (3D rigid bodies) and force elements (springs, dampers, forces, torques).



Figure 4: Component icon

The models also have a graphical interface, with a visualization of simulation outcomes within the same 3D environment used in the *Multibody* library, providing the user with immediate visual feedback.

Mathematical modelling details are partially indicated below in Modelica characteristic language:

```
parameter SI.Density rho=7800 "Material Volume Density";
parameter SI.Length L=0.2 "Beam Length";
parameter SI.Height a=0.005 "Height of section";
parameter SI.Breadth b=0.02 "Breath of section";
parameter SI.ModulusOfElasticity E=210e9 "Material Youngs modulus";
parameter SI.ShearModulus G=8.077e10 "Material Shear modulus";
```

```
parameter SI.SecondMomentOfArea J1=7.025e-10
"Cross sectional inertia 1";
parameter SI.SecondMomentOfArea J2=3.33333e-9
"Cross sectional inertia 2";
parameter SI.SecondMomentOfArea J3=2.08333e-10
"Cross sectional inertia 3";
parameter Real Alpha=1e-3 "Rayleigh structural
damping proportional to mass [sec^-1]";
parameter Real Beta=5e-5 "Rayleigh structural
damping proportional to stiffness [sec]";
parameter Integer N(min=2) = 2 "Number of Elements";
```

Dynamic equations for the 3D flexible beam model are as follows:

```
[m*identity(3), transpose(StbarCross), Sbar]*[aa - g_0; za; ddqf] = QvR + matrix(fa + fb_a);
```

```
[StbarCross, Ithth_bar, Ithf_bar]*[aa - g_0; za; ddqf] = QvAlpha + matrix(ta + tb_a + cross({L,0,0} + S1*B[N, :, :] * qf, fb_a));
```

```
[transpose(Sbar), transpose(Ithf_bar), mff]*[aa - g_0; za; ddqf] = Qvf + Qef - matrix(Kff*qf) - matrix((Alpha*mff + Beta*Kff)*dqf);
```

Using the degrees of freedom q_f and the derivate of degrees of freedom dq_f , information can be passed from *Frame B* to *Frame A* as indicated in the following equations:

```
FrameB.R=Modelica.Mechanics.MultiBody.Frames.absoluteRotation(FrameA.R, R_rel);
```

```
R_rel=Modelica.Mechanics.MultiBody.Frames.axesRotations({1,2,3},{qf[(6*N)-2],qf[(6*N)-1],qf[6*N]},{dqf[(6*N)-2],dqf[(6*N)-1],dqf[6*N]});
```

5 Simulations

The different flexible beam models have been validated by several simulation analyses performed in the *Dymola* simulation environment [1] and compared with a general purpose commercial FEM code (ANSYS).

5.1 First example (Static)

As a preliminary result, a 3D simulation has been performed applying movement to the free extremity of a cantilever beam (Figure 5).

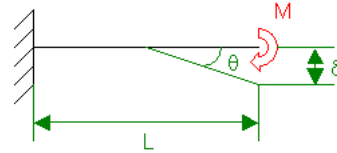


Figure 5: Beam 3D

Beam displacement and rotation analytical form calculation follows:

$$\delta = \frac{ML^2}{2EI} = \frac{2\pi EI}{L} L^2 = \pi L \quad (33)$$

$$\theta = \frac{ML}{EI} = \frac{2\pi EI}{L} L = 2\pi \quad (34)$$

Variables used in the simulation are expressed in the following table:

Variable	Value
Rho (kg/m ³)	7800
L (m)	1
a (m)	0.02
b (m)	0.06285
E (Pa)	210e9
G (Pa)	8.077e10
J1 (m ⁴)	1.33242e-7
J2 (m ⁴)	4.138e-7
J3 (m ⁴)	4.19e-8
Alpha	1e-3
Beta	5e-5
N	2
Applied moment M (N.m)	2*π*E*I/L

The simulation has been performed by connecting 10 elements in series (Figure 6), to overcome the assumption of small displacements in the internal development of the element.

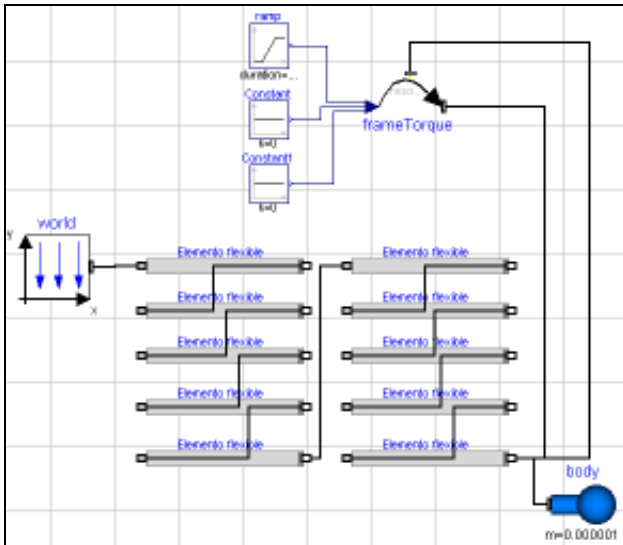


Figure 6: Modeling in Dymola

Figure 7 shows the results obtained where the 3D position of the beam's extremities is plotted at the end of the simulation. As expected, X and Y positions describe a circle since as this was verified in the theoretical analysis. The same results are obtained, irrespective of the direction of movement applied.

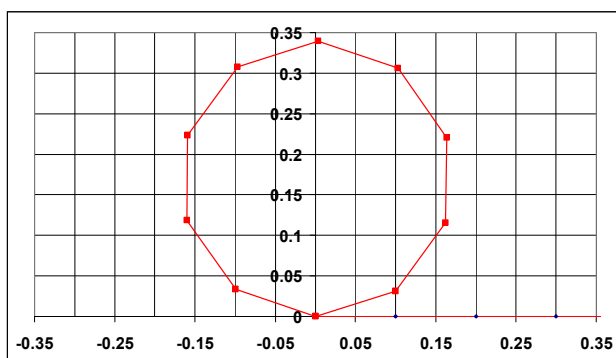


Figure 7: Results obtained

By means of this simulation correct element response to static flexion has been obtained.

5.2 Second example (Dynamic)

In this example (Figure 8) the dynamical operation of a flexible pendulum articulated to one of its ends has been analysed. The initial position of the flexible element is horizontal (X direction) and gravity is applied in the vertical direction (Z direction).

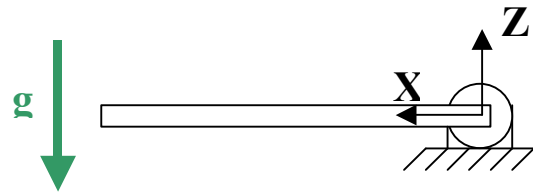


Figure 8: Articulated pendulum

Figure 9 shows the comparison of the results obtained with Dymola, ANSYS code and considering a rigid element. The mechanical properties of the beam are the same as in the previous example except L being 0.2. This simulation can be performed with one or more elements because the large displacement of the end is mainly governed by the revolute joint, the flexibility acting only in the longitudinal direction of the beam.

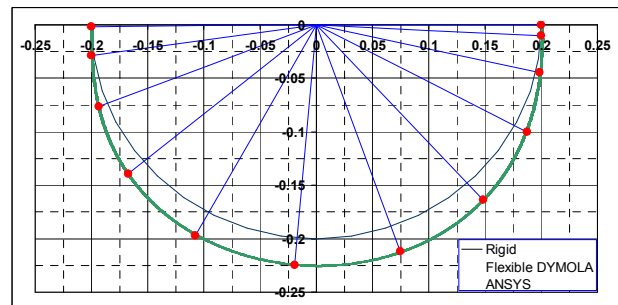


Figure 9: Results obtained with different programs

Between the rigid model and the flexible model there is a lack of coordination in frequency where the rigid element is advanced with regard to the flexible model. The results concur with those obtained in ANSYS.

6 Generalization and model versions

In the general and parametric 3D model created, modifications have been performed in order to introduce mass and stiffness (matrices calculated for example using ANSYS). In this case the matrix of mass becomes constant along the simulation as well as it happens with the matrix of stiffness and besides Coriolis's terms though they appear are not updated.

By means of this simplification the computational cost decreases because the reduction in the calculation of the equations of movement. It must be pointed out that a calculation error is made, however this error could be fully undertaken.

7 Conclusions

To date, there has been no evidence of a Dymola model that can simulate flexible mechanisms in 3D and therefore the importance of the model created, as its correct operation has been verified, both statically and dynamically.

This development in flexible element simulation, based on *Modelica* programming language, enables the simulation of flexible 3D mechanisms and the integration of these model into other disciplines, for example, control tasks.

The only drawback to the implemented model is the large number of equations required to solve the problem, this slows down simulation and requires powerful simulation equipment. Although, compared to other simulation programs, the 3D flexible model created *Dymola* is about 100 times faster than the same model simulated in ANSYS.

To overcome this limitation a simplified model has been implemented that considers a fixed mass matrix and a single element (no discretisation), that speeds up simulation, and is suitable for preliminary modeling steps.

The developed model can be easily implemented for any type of constant shape along its length.

Future work will include non-constant shape beams and the development of a model based on the assumed modes theory [3] considering free boundaries.

This flexible model will be used for the development of a wearable robotics where the mechanism mass is key.

A Structural Stiffness Matrix

$$K_{ij} = \begin{bmatrix} \frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_x}{(l)^3} & 0 & 0 & 0 & \frac{6EI_x}{(l)^2} & 0 & -\frac{12EI_x}{(l)^3} & 0 & 0 & 0 & \frac{6EI_x}{(l)^2} & 0 \\ 0 & 0 & \frac{12EI_z}{(l)^3} & 0 & -\frac{6EI_z}{(l)^2} & 0 & 0 & 0 & -\frac{12EI_z}{(l)^3} & 0 & \frac{6EI_z}{(l)^2} & 0 & 0 \\ 0 & 0 & 0 & \frac{GJ_t}{l} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{GJ_t}{l} & 0 \\ 0 & 0 & -\frac{6EI_x}{(l)^2} & 0 & \frac{4EI_x}{l} & 0 & 0 & 0 & \frac{6EI_x}{(l)^2} & 0 & \frac{2EI_x}{l} & 0 & 0 \\ 0 & \frac{6EI_x}{(l)^2} & 0 & 0 & 0 & \frac{4EI_x}{l} & 0 & -\frac{6EI_x}{(l)^2} & 0 & 0 & 0 & \frac{2EI_x}{l} & 0 \\ -\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI_x}{(l)^3} & 0 & 0 & 0 & \frac{6EI_x}{(l)^2} & 0 & \frac{12EI_x}{(l)^3} & 0 & 0 & 0 & -\frac{6EI_x}{(l)^2} & 0 \\ 0 & 0 & -\frac{12EI_z}{(l)^3} & 0 & \frac{6EI_z}{(l)^2} & 0 & 0 & 0 & \frac{12EI_z}{(l)^3} & 0 & \frac{6EI_z}{(l)^2} & 0 & 0 \\ 0 & 0 & 0 & -\frac{GJ_t}{l} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{GJ_t}{l} & 0 & 0 \\ 0 & 0 & \frac{6EI_x}{(l)^2} & 0 & \frac{2EI_x}{l} & 0 & 0 & 0 & \frac{6EI_x}{(l)^2} & 0 & \frac{4EI_x}{l} & 0 & 0 \\ 0 & \frac{6EI_x}{(l)^2} & 0 & 0 & 0 & \frac{2EI_x}{l} & 0 & -\frac{6EI_x}{(l)^2} & 0 & 0 & \frac{4EI_x}{l} & 0 & 0 \end{bmatrix}$$

B Mass Matrix Components

$$\int_V \rho S_1^T S_1 dV = \begin{bmatrix} \frac{m}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{Q_z l}{5} & \frac{6I_z l}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{Q_z l}{5} & \frac{6I_z l}{5} & \frac{6I_x l}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{I_x(l)^2}{5} & 0 & \frac{2I_x(l)^3}{15} & 0 & 0 & 0 & 0 & 0 & 0 \\ Q_z(l)^2 & -\frac{I_{xc}(l)^2}{12} & -\frac{I_{yc}(l)^2}{10} & 0 & \frac{2I_{xc}(l)^3}{15} & 0 & \frac{2I_{yc}(l)^3}{15} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{I_{xc}(l)^2}{12} & \frac{I_{yc}(l)^2}{10} & 0 & -\frac{2I_{xc}(l)^3}{15} & 0 & \frac{2I_{yc}(l)^3}{15} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{m}{6} & \frac{I Q_x}{2} & \frac{Q_z l}{2} & 0 & -\frac{Q_z(l)^2}{12} & \frac{Q_z(l)^2}{15} & \frac{m}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{Q_z l}{2} & \frac{6I_x l}{5} & -\frac{6I_{xc} l}{5} & 0 & \frac{I_{xc}(l)^2}{10} & -\frac{I_{xc}(l)^2}{10} & -\frac{Q_z l}{2} & \frac{6I_z l}{5} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{6I_x l}{5} & -\frac{6I_{xc} l}{5} & 0 & \frac{I_{xc}(l)^2}{10} & -\frac{I_{xc}(l)^2}{10} & -\frac{Q_z l}{2} & -\frac{6I_{xc} l}{5} & \frac{6I_z l}{5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{I_{xc}(l)^2}{10} & \frac{I_{yc}(l)^2}{10} & \frac{Q_z(l)^2}{12} & -\frac{I_{xc}(l)^2}{10} & -\frac{I_{yc}(l)^2}{10} & 0 & 0 & 0 & 0 \\ Q_z(l)^2 & -\frac{I_{xc}(l)^2}{12} & -\frac{I_{yc}(l)^2}{10} & 0 & -\frac{I_{xc}(l)^3}{30} & \frac{I_{yc}(l)^3}{30} & \frac{Q_z(l)^2}{12} & \frac{I_{xc}(l)^2}{10} & \frac{I_{yc}(l)^2}{10} & 0 & \frac{2I_{xc}(l)^3}{15} & 0 & 0 \\ 0 & \frac{I_{xc}(l)^2}{12} & \frac{I_{yc}(l)^2}{10} & 0 & \frac{I_{xc}(l)^3}{30} & -\frac{I_{yc}(l)^3}{30} & -\frac{Q_z(l)^2}{12} & -\frac{I_{xc}(l)^2}{10} & -\frac{I_{yc}(l)^2}{10} & 0 & -\frac{2I_{xc}(l)^3}{15} & 0 & 0 \end{bmatrix}$$

$$\int_V \rho S_2^T S_2 dV = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{13m}{35} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{7(l)^2}{20} Q_z & 0 & \frac{(l)^3}{3} I_{\eta} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{11ml}{210} & 0 & -\frac{(l)^3}{20} Q_z & 0 & \frac{m(l)^2}{105} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{9m}{70} & 0 & -\frac{3(l)^2}{20} Q_z & 0 & \frac{13ml}{420} & 0 & 0 & \frac{13m}{35} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{3(l)^2}{20} Q_z & 0 & \frac{(l)^3}{6} I_{\eta} & 0 & -\frac{(l)^3}{30} Q_z & 0 & -\frac{7(l)^2}{20} Q_z & 0 & 0 & \frac{(l)^3}{3} I_{\eta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{13ml}{420} & 0 & \frac{(l)^3}{30} Q_z & 0 & -\frac{m(l)^2}{140} & 0 & -\frac{11ml}{210} & 0 & 0 & \frac{(l)^3}{20} Q_z & 0 & \frac{m(l)^2}{105} \end{bmatrix}$$

$$\int_V \rho S_3^T S_3 dV = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{55} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{7(l)^2}{20} Q_{\eta} & \frac{(l)^3}{3} I_{\zeta} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{11ml}{210} & -\frac{(l)^3}{20} Q_{\eta} & \frac{m(l)^2}{105} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{9m}{70} & \frac{3(l)^2}{20} Q_{\eta} & -\frac{13ml}{420} & 0 & 0 & 0 & \frac{13m}{35} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3(l)^2}{20} Q_{\eta} & \frac{(l)^3}{6} I_{\eta} & -\frac{(l)^3}{30} Q_{\eta} & 0 & 0 & 0 & \frac{7(l)^2}{20} Q_{\eta} & \frac{(l)^3}{3} I_{\zeta} & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{420} & \frac{(l)^3}{30} Q_{\eta} & -\frac{m(l)^2}{140} & 0 & 0 & 0 & \frac{11ml}{210} & \frac{(l)^3}{20} Q_{\eta} & \frac{m(l)^2}{105} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\left[\int_V \rho S_1^T S_2 dV \right]^y =$$

$$= \begin{bmatrix} 0 & \frac{7m}{20} & 0 & -\frac{(l)^2}{3} Q_\zeta & 0 & \frac{ml}{20} & 0 & \frac{3m}{20} & 0 & -\frac{(l)^2}{6} Q_\zeta & 0 & -\frac{ml}{30} \\ 0 & \frac{l}{2} Q_\eta & 0 & -\frac{(l)^2}{2} I_{\eta\zeta} & 0 & \frac{(l)^2}{10} Q_\eta & 0 & \frac{l}{2} Q_\eta & 0 & -\frac{(l)^2}{2} I_{\eta\zeta} & 0 & -\frac{(l)^2}{10} Q_\eta \\ 0 & \frac{l}{2} Q_\zeta & 0 & -\frac{(l)^2}{2} I_\eta & 0 & \frac{(l)^2}{10} Q_\zeta & 0 & \frac{l}{2} Q_\zeta & 0 & -\frac{(l)^2}{2} I_\eta & 0 & -\frac{(l)^2}{10} Q_\zeta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{(l)^2}{10} Q_\zeta & 0 & -\frac{(l)^3}{12} I_\eta & 0 & 0 & 0 & -\frac{(l)^2}{10} Q_\zeta & 0 & -\frac{(l)^3}{12} I_\eta & 0 & -\frac{(l)^2}{60} Q_\zeta \\ 0 & -\frac{(l)^2}{10} Q_\eta & 0 & \frac{(l)^3}{12} I_{\eta\zeta} & 0 & 0 & 0 & \frac{(l)^2}{10} Q_\eta & 0 & -\frac{(l)^3}{12} I_{\eta\zeta} & 0 & -\frac{(l)^2}{60} Q_\eta \\ 0 & \frac{3m}{20} & 0 & -\frac{(l)^2}{6} Q_\zeta & 0 & \frac{ml}{30} & 0 & \frac{7m}{20} & 0 & -\frac{(l)^2}{3} Q_\zeta & 0 & -\frac{ml}{20} \\ 0 & -\frac{l}{2} Q_\eta & 0 & \frac{(l)^2}{2} I_{\eta\zeta} & 0 & -\frac{(l)^2}{10} Q_\eta & 0 & -\frac{l}{2} Q_\eta & 0 & \frac{(l)^2}{2} I_{\eta\zeta} & 0 & \frac{(l)^2}{10} Q_\eta \\ 0 & -\frac{l}{2} Q_\zeta & 0 & \frac{(l)^2}{2} I_\eta & 0 & -\frac{(l)^2}{10} Q_\zeta & 0 & -\frac{l}{2} Q_\zeta & 0 & \frac{(l)^2}{2} I_\eta & 0 & \frac{(l)^2}{10} Q_\zeta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{(l)^2}{10} Q_\zeta & 0 & \frac{(l)^3}{12} I_\eta & 0 & -\frac{(l)^3}{60} Q_\zeta & 0 & \frac{(l)^2}{10} Q_\zeta & 0 & -\frac{(l)^3}{12} I_\eta & 0 & 0 \\ 0 & \frac{(l)^2}{10} Q_\eta & 0 & -\frac{(l)^3}{12} I_{\eta\zeta} & 0 & \frac{(l)^3}{60} Q_\eta & 0 & -\frac{(l)^2}{10} Q_\eta & 0 & \frac{(l)^3}{12} I_{\eta\zeta} & 0 & 0 \end{bmatrix}^y$$

References

- [1] Dymola. *Dynamic Modelling Laboratory*. Dynasim AB, Lund, Sweden.
- [2] A. A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, 1998.
- [3] F. Schiavo, G. Ferreti, L. Viganò. *Object-Oriented Modelling and Simulation of Flexible Multibody Thin Beams in Modelica with the Finite Element Method*. In 4th International Modelica Conference, Hamburg, March 7-8, 2005.
- [4] M. Otter, H. Elmqvist, and S. E. Mattsson. *The new Modelica multibody library*. In 3rd Modelica Conference, Linköping, Sweden, November 3-4, 2003.

$$\left[\int_V \rho S_1^T S_3 dV \right]^y =$$

$$= \begin{bmatrix} 0 & 0 & \frac{7m}{20} & \frac{(l)^2}{3} Q_\eta & -\frac{ml}{20} & 0 & 0 & 0 & \frac{3m}{20} & \frac{(l)^2}{6} Q_\eta & -\frac{ml}{30} & 0 \\ 0 & 0 & \frac{l}{2} Q_\eta & \frac{(l)^2}{2} I_\zeta & -\frac{(l)^2}{10} Q_\eta & 0 & 0 & 0 & \frac{l}{2} Q_\eta & \frac{(l)^2}{2} I_\zeta & \frac{(l)^2}{10} Q_\eta & 0 \\ 0 & 0 & \frac{l}{2} Q_\zeta & \frac{(l)^2}{2} I_{\eta\zeta} & -\frac{(l)^2}{10} Q_\zeta & 0 & 0 & 0 & \frac{l}{2} Q_\zeta & \frac{(l)^2}{2} I_{\eta\zeta} & \frac{(l)^2}{10} Q_\zeta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{(l)^2}{10} Q_\zeta & \frac{(l)^3}{12} I_{\zeta\eta} & 0 & 0 & 0 & 0 & -\frac{(l)^2}{10} Q_\zeta & -\frac{(l)^3}{12} I_{\zeta\eta} & -\frac{(l)^2}{60} Q_\zeta & 0 \\ 0 & 0 & -\frac{(l)^2}{10} Q_\eta & -\frac{(l)^3}{12} I_\zeta & 0 & 0 & 0 & 0 & \frac{(l)^2}{10} Q_\eta & \frac{(l)^3}{12} I_\zeta & \frac{(l)^2}{60} Q_\eta & 0 \\ 0 & 0 & \frac{3m}{20} & \frac{(l)^2}{6} Q_\eta & -\frac{ml}{30} & 0 & 0 & 0 & \frac{7m}{20} & \frac{(l)^2}{3} Q_\eta & \frac{ml}{20} & 0 \\ 0 & 0 & -\frac{l}{2} Q_\eta & -\frac{(l)^2}{2} I_\zeta & \frac{(l)^2}{10} Q_\eta & 0 & 0 & 0 & -\frac{l}{2} Q_\eta & -\frac{(l)^2}{2} I_\zeta & -\frac{(l)^2}{10} Q_\eta & 0 \\ 0 & 0 & -\frac{l}{2} Q_\zeta & -\frac{(l)^2}{2} I_{\eta\zeta} & \frac{(l)^2}{10} Q_\zeta & 0 & 0 & 0 & -\frac{l}{2} Q_\zeta & -\frac{(l)^2}{2} I_{\eta\zeta} & -\frac{(l)^2}{10} Q_\zeta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{(l)^2}{10} Q_\zeta & -\frac{(l)^3}{12} I_{\eta\zeta} & \frac{(l)^3}{60} Q_\zeta & 0 & 0 & 0 & \frac{(l)^2}{10} Q_\zeta & \frac{(l)^3}{12} I_{\eta\zeta} & 0 & 0 \\ 0 & 0 & \frac{(l)^2}{10} Q_\eta & \frac{(l)^3}{12} I_\zeta & -\frac{(l)^3}{60} Q_\eta & 0 & 0 & 0 & \frac{(l)^2}{10} Q_\eta & -\frac{(l)^3}{12} I_\zeta & 0 & 0 \end{bmatrix}^y$$

$$\left[\int_V \rho S_2^T S_3 dV \right]^y =$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{13m}{35} & \frac{7(l)^2}{20} Q_\eta & -\frac{11ml}{210} & 0 & 0 & 0 & \frac{9m}{70} & \frac{3(l)^2}{20} Q_\eta & \frac{13ml}{420} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{7(l)^2}{20} Q_\zeta & -\frac{(l)^3}{3} I_{\eta\zeta} & \frac{(l)^3}{20} Q_\zeta & 0 & 0 & 0 & -\frac{3(l)^2}{20} Q_\zeta & -\frac{(l)^3}{6} I_{\eta\zeta} & -\frac{(l)^3}{30} Q_\zeta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{11ml}{210} & \frac{(l)^3}{20} Q_\eta & -\frac{m(l)^2}{105} & 0 & 0 & 0 & \frac{13ml}{420} & \frac{(l)^3}{30} Q_\eta & \frac{m(l)^2}{140} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{9m}{70} & \frac{3(l)^2}{20} Q_\eta & -\frac{13ml}{420} & 0 & 0 & 0 & \frac{13m}{35} & \frac{7(l)^2}{20} Q_\eta & \frac{11ml}{210} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3(l)^2}{20} Q_\zeta & -\frac{(l)^3}{6} I_{\eta\zeta} & \frac{(l)^3}{30} Q_\zeta & 0 & 0 & 0 & -\frac{7(l)^2}{20} Q_\zeta & -\frac{(l)^3}{3} I_{\eta\zeta} & -\frac{(l)^2}{20} Q_\zeta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{13ml}{420} & -\frac{(l)^3}{30} Q_\eta & \frac{m(l)^2}{140} & 0 & 0 & 0 & -\frac{11ml}{210} & -\frac{(l)^2}{20} Q_\eta & -\frac{m(l)^2}{105} & 0 \end{bmatrix}^y$$

A Modelica Library for Space Flight Dynamics

Tiziano Pulecchi Francesco Casella Marco Lovera
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract

The Modelica Space Flight Dynamics Library has been developed as a unified environment to be used throughout the design cycle of the Attitude and Orbit Control System (AOCS) for a generic multibody, possibly flexible, spacecraft. The library architecture has been recently redesigned, exploiting at its best Modelica's reusability, flexibility and modularity features. In this contribution the main benefits of the Space Flight Dynamics Library are discussed with special emphasis posed upon flexibility and advantage of exploiting Modelica's nonlinear dynamic inversion capability for preliminary assessment of disturbance torques in nominal orbit and attitude.

Keywords: space flight dynamics; replaceable model; flexibility; simulation

1 Introduction

There is an increasing need for efficient design tools in every domain involved in spacecraft design, and particularly in the area of control oriented modelling and simulation. Specific tools have to be developed for the design of both the system architecture and the Attitude and Orbit Control System (AOCS), bearing in mind the principles of reusability, flexibility and modularity. The main issue in the development of such tools should be to try and work out a unified environment to be used throughout the design cycle of the AOCS, namely, the mission analysis stage, the preliminary and detailed design and simulation phases, the generation and testing of the on-board code, the development of the AOCS Electrical Ground Support Equipment (EGSE) and the post-launch data analysis activities. A number of commercial tools are available to support one or more of the above mentioned phases in the development of AOCS subsystems, however none of them seems capable of providing complete coverage of the whole development cycle in a sufficiently flexible way. Moreover, the spacecraft architecture is

traditionally designed using domain specific software packages that best solve their tasks with respect to the different disciplines involved, e.g., flight mechanics, propulsion, controls; as a drawback, it is quite cumbersome to link together the different model components and exploit their reusability in the framework of future missions.

The Modelica Space Flight Dynamics Library provides a systematic approach to simulation of spacecraft dynamics based on modern acausal object-oriented modelling techniques. The development of simulation tools for satellite attitude and orbit dynamics within the object-oriented paradigm has been the subject of previous work (see [14], where an overview of the existing tools for AOCS modeling is presented). Surprisingly enough, however, while the use of Modelica for aerospace applications has led to the development of a library for flight dynamics (see [8]), very little activity in the spacecraft domain has been reported. Some preliminary results in the development of a Modelica spacecraft modeling library have been presented in [4, 11, 5]). More recently, the model components presented in the cited references have been revised in order to take advantage of the Modelica Multi-Body library (see [9]) which turns out to be extremely suitable to serve as a basis for the development of the basic model components for the mechanical parts of spacecraft models. In particular, a recent extension of the above mentioned library (see [2, 13]) is proving specially beneficial for the simulation of spacecraft with flexible appendages (see [12]).

In this paper the new Space Flight Dynamics Library is described, emphasizing its flexibility and showing the advantage of exploiting Modelica's nonlinear dynamic inversion capability for the preliminary assessment of disturbance torques acting upon a spacecraft in nominal orbit and attitude. The paper is organized as follows: first the Space Flight Dynamics Library will be described in detail in Sections 2-3; subsequently, the library flexibility will be exploited in Section 4 in a preliminary analysis of the external disturbance

torques acting on a spacecraft in its nominal orbit and attitude. Finally, in Section 5 concluding remarks will be outlined.

2 The Space Flight Dynamics Library

Modelica turns out to be specially suited for the modelling of spacecraft dynamics under many respects:

- Coordinate frames can be simply included in the model in terms of connectors, describing kinematic transformations from one coordinate system to another.
- Spacecraft dynamics can be modelled by extending suitable classes available in the MultiBody Library.
- Specific Modelica constructs are available to deal with the modelling of physical fields and environmental quantities. This feature turns out to be extremely useful in modelling the space environment and representing the interaction between the environment and the spacecraft. In particular, with a suitable choice of the environment interfaces, models of increasing complexity for each of the relevant environmental fields can be implemented.
- Sensors and actuators can also be easily represented in the Modelica paradigm. For instance, a component for the simulation of magnetic torques is modelled in terms of the interaction with the geomagnetic field, while the momentum exchange between spacecraft and wheels is modelled via a simple mechanical connector allowing one rotational degree of freedom ¹.
- Packages of data sheets for each class can be constructed and components easily modified within each spacecraft model, using Modelica's advanced features (see, e.g., [10]).
- *C code* can be easily linked to Modelica models, allowing the designer to reuse, for instance, a wide range of available specific algorithms and routines he is confident with, without going through all the trouble of re-implementing them in Modelica code.

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

- Finally, as the components of the library are independent from each other, one can exploit this flexibility in order to build a simulation model of increasing complexity and accuracy according to the needs associated with each phase of the AOCS development process.

In addition, the availability of the Modelica MultiBody Library (see [9]) leads to further advantages, since the MultiBody components can be extensively reused. Furthermore, recent developments to the library allowing the modeling and simulation of flexible multibody systems (see [2, 13]) make it possible to deal with the dynamics of spacecraft with flexible appendages such as gravity gradient booms, antennas or solar panel arrays.

The Space Flight Dynamics Library encompasses all necessary utilities to ready a reliable and quick-to-use scenario for a generic space mission, providing a wide choice of most commonly used models for AOCS sensors, actuators and controls. The Space Flight Dynamics Library's model reusability is such that, as new missions are conceived, the library can be used as a base upon which readily and easily build a simulator. This goal can be achieved simply by interconnecting the standard Space Flight Dynamics Library objects, possibly with new components purposely designed to cope with specific mission requirements, regardless of space mission scenario in terms of either mission environment (e.g., planet Earth, Mars, solar system), spacecraft configuration or embarked on board systems (e.g., sensors, actuators, controls).

Section 3 deals with a thorough description of the main Space Flight Dynamics Library components.

3 Basic model components

The generic spacecraft simulator will consist of an extended **World** model and one or more **Spacecraft** models:

1. Extended **World** model: a new **World** model, extending **Modelica.MultiBody.World** has been defined. It provides all the functions needed for a complete representation of the space environment as seen by an Earth orbiting spacecraft:
 - Gravitational field models;
 - Geomagnetic field models;
 - Atmospheric models;
 - Solar radiation and eclipse models;

- Models for Sun and Moon ephemeris;

Such an extension to the basic **World** model as originally provided in the MultiBody library plays a major role in the realistic simulation of the dynamics of a spacecraft as the linear and angular motion of a satellite are significantly influenced by its interaction with the space environment.

2. **Spacecraft** model: a completely reconfigurable spacecraft including components:

- (a) **SpacecraftDynamics**: this component has been defined by extending the rigid body model on the basis of the already available **Modelica.MultiBody.Parts.Body** component. The main modifications reside in the selectable evaluation of the interactions between the spacecraft and the space environment and on the additional initialization option for the simulation via selection of a specific orbit for the spacecraft. Data for custom orbits and spacecraft inertial properties and geometry (influencing both aerodynamic and solar radiation behavior) are stored in dedicated library packages. The **SpacecraftDynamics** interface consists of the standard Modelica library mechanical connector.

- (b) **SensorBlock**: this *replaceable* model consists in a reconfigurable set of attitude sensors to be chosen among custom Space Flight Dynamics Library **SensorBlock** implementations. The model *replaceable* feature is active on all levels, such that, for instance, the same basic **Spacecraft** model can be instantiated as having a custom star tracker sensor (corresponding to a specified supplier's serial number), model (such as ideal measure, measure corrupted by simple white noise and bias, optional time delay and availability bit) and configuration (defined by star trackers number, location and orientation with respect to the spacecraft's reference frame).

The Space Flight Dynamics Library encompasses mathematical models of different degree of complexity for star sensors, gyroscopes, magnetometers and GPS receivers. The **SensorBlock** interface consists of a standard Modelica library mechanical connector toward model **SpacecraftDynamics**

and of the expandable connector (see [1]) **SensorBus** toward replaceable model **ControlBlock**.

- (c) **ActuatorBlock**: this *replaceable* model consists in a reconfigurable set of attitude control actuators to be chosen among custom Space Flight Dynamics Library **ActuatorBlock** implementations. Mathematical models of different degree of complexity for commonly employed actuators and actuators set have been implemented in the Space Flight Dynamics Library, including momentum and reaction wheels, magnetic torquers and cold gas thrusters. The **ActuatorBlock** interface consists of a standard Modelica library mechanical connector toward model **SpacecraftDynamics** and of the expandable connector **ActuatorBus** toward replaceable model **ControlBlock**.

Note, in passing, that sensor and actuator models have been developed in a control-oriented framework, i.e., at the current level of refinement they are not based on physical models of the measurement process. More advanced models can however be included in the considered framework if needed.

- (d) **ControlBlock**: this *replaceable* model implements the spacecraft Attitude Control System (ACS), including blocks supervising the basic attitude determination, attitude control and control allocation functions. The **ControlBlock** interface consists of two expandable connectors, **SensorBus** and **ActuatorBus**, toward replaceable models **SensorBlock** and **ActuatorBlock** respectively.

Note that the spacecraft can be either modeled as a rigid body or as a multibody system, possibly with flexible appendages (see [12] for details).

A short description of the Space Flight Dynamics Library's **World**, **Spacecraft** and **SpacecraftDynamics** models is given in the following subsections; **SensorBlock**, **ActuatorBlock** and **ControlBlock** models are omitted for brevity.

3.1 Extended World model

The user interface for Extended World Model component is shown in Figure 1; as can be seen from the Figure, the user can define the initial date and time of

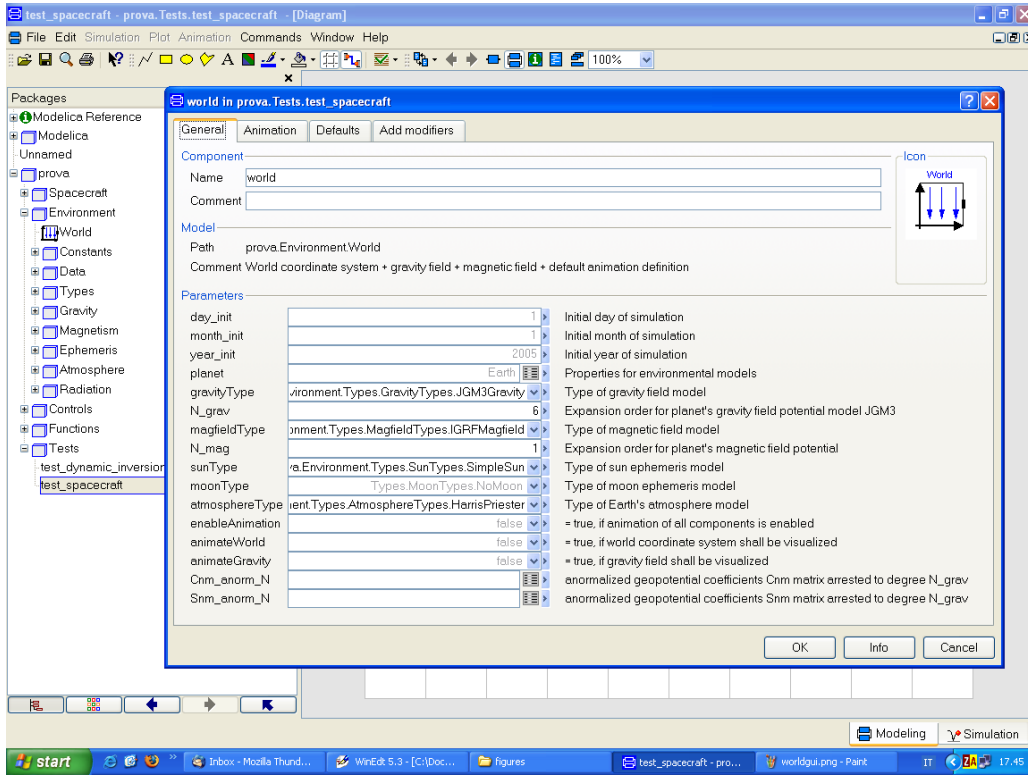


Figure 1: User interface for the Extended World model.

the simulation and choose among the available models for the Earth's gravity field (J_2 , J_4 or the more general $JGM-3$ model for the Earth's gravitational potential, see [7]), for the Earth's magnetic field (dipole, quadrupole and the IGRF model, see [16]), for the atmospheric density model and for the Sun and Moon ephemeris tables.

As is well known, the Earth's gravitational potential U_g may be described by the function

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

where P_n^m are the Legendre polynomials

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

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

R_e is the mean equatorial Earth radius, r, θ and λ are the point's spherical coordinates and coefficients J_n , C_n^m , S_n^m are the zonal, sectoral and tesseral coefficients. Depending on the mission characteristics and on the purpose of attitude control simulations, a satisfactory

approximation can be obtained by choosing the order of the expansion in a suitable way. The Earth gravitational field components (expressed in spherical coordinates) are then given by

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

Similarly, the geomagnetic potential U_m , is described by the function

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

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

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

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

The coefficients for the geomagnetic potential adopted in the simulation environment correspond to the so-called International Geomagnetic Reference Field (IGRF) model for the Earth's magnetic field (see [16]). The components of the geomagnetic field (expressed

in spherical coordinates) are then given by

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

Similar models for the atmospheric density and the Sun and Moon position have been implemented, according to [7, 15].

3.2 Spacecraft model

The **Spacecraft** model is structured according to the diagram in Figure 2: the component associated with the (perturbed) linear and angular dynamics of the satellite (described in Figure 3) is connected to the actuators and sensors blocks via a standard Modelica mechanical connector, whilst the interconnection among sensors, actuators and control blocks is realized via suitably defined data buses. For instance, the default choice for the replaceable model **SensorBlock**, comprising a single star tracker, gyroscope, GPS receiver and magnetometer, is depicted in Figure 4. As can be seen from the Figure, models for each of the on-board sensors are included; in particular, each sensor is characterized by a mechanical interface, corresponding to the physical mounting of the instrument on the satellite body (taking into account the definition of the local sensor reference frame via a suitable change of coordinates) and by a signal interface. The sensors data bus is therefore defined by the collection of output signals coming from each of the available sensors (using Modelica expandable connectors, see [1]).

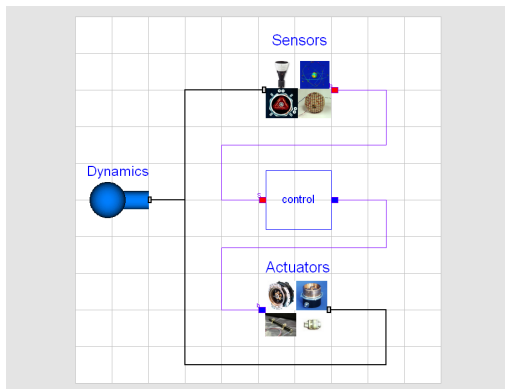


Figure 2: Structure of the Spacecraft model.

Note that the sensor and actuator models have been defined by taking full advantage of the object orientation of the modeling language: the core definition for each sensor/actuator model is at the interface level; mathematical models of increasing complexity are available, ranging from ideal sensors providing

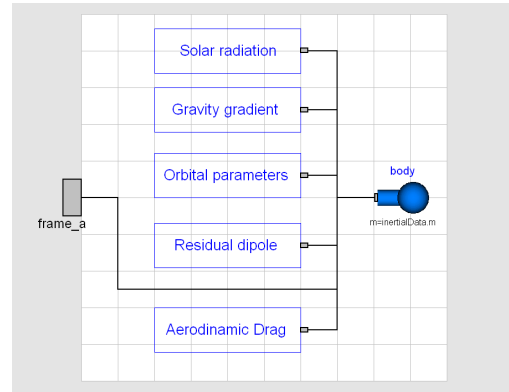


Figure 3: Layout of the SpacecraftDynamics model.

ideal, continuous-time measurements to more refined models taking into account measurement errors and the actual sampling rate of the sensors.

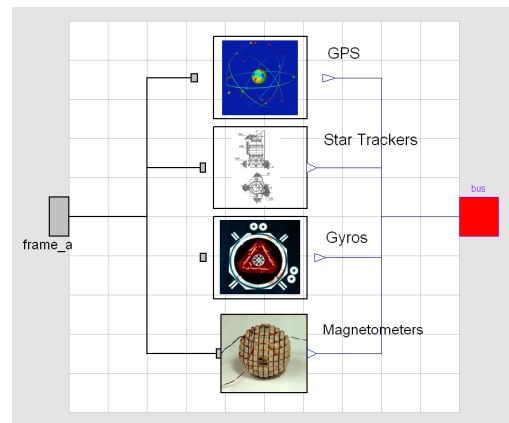


Figure 4: Default choice for replaceable model **SensorBlock**, comprising a single star tracker, gyroscope, GPS receiver and magnetometer.

Let us stress the point that the Space Flight Dynamics Library **Spacecraft** model is completely customizable for what concerns actuators, sensors and controls, which can be selected among standard library models via dedicated popup menus from the model **Spacecraft** graphical user interface (see Figure 5). The possible customization are virtually countless: Limiting the discussion only to possible sensors customization, the present Space Flight Dynamics Library implementations allows for choosing among all possible combinations arising from:

- 3 star tracker models ;
- 2 star tracker configurations ;
- 2 GPS models;
- 1 magnetometers models;

- 2 gyroscope models;
- 4 sensors block instances.

Figure 6 provides an idea of the customization made possible by the Space Flight Dynamics Library for what attains the **SensorBlock** only.

More freedom yet is available within the **SpacecraftDynamics** model for what attains the interaction of the spacecraft with the space environment as defined in the extended **World** model, via the options for activation/deactivation of magnetic residual dipole, aerodynamic and solar radiation disturbance forces and torques.

The **Spacecraft** model extreme flexibility was achieved by setting Dymola provided annotation option **choicesAllMatching** to true for all the replaceable models. In such a way, a model is a candidate for re-declaration of a given replaceable model if and only if it extends a suitable base interface **thisModelFamilyInterface**. If a new model candidate is designed, **Dymola** will automatically update the candidate model list with the new entry, provided it is an extension of the proper interface. Finally, the **choicesAllMatching** annotation adoption prevents the user from unsuitable redeclarations.

3.3 SpacecraftDynamics model

This new component uses the `Modelica.MultiBody.Parts.Body` model to account for the interaction between the spacecraft and the environment. In particular, the following disturbance forces and torques can be selectively included in the spacecraft model:

- Gravity gradient torques;
- Magnetic torques, arising from the presence of a non zero spacecraft's residual magnetic dipole;
- Aerodynamic forces and torques, produced by the interaction with the planet's atmosphere;
- Solar radiation pressure originated disturbance forces and torques.

Specifically, the latter contributions to the disturbance forces and torques requires the definition of the interaction between the spacecraft geometry, defined as an assembly of planar and possibly cylindrical surfaces, and the average solar radiation pressure Φ . When the spacecraft is fully illuminated, the force acting upon

a single exposed surface is given by the momentum exchange law

$$\frac{dq_{sc}}{dt} = p_{\odot} \frac{(1AU)^2}{\|R\|^2} \sum_{i=1}^{\#surf} A_i \cos(\theta_i) \cdot [(1 - \varepsilon_i)(-\hat{R}) + 2\varepsilon_i \cos(\theta_i)(-n_{ext,i})]$$

where q_{sc} is the spacecraft's momentum, $p_{\odot} = 4.56e^{-6} Jm^{-2}$ is the mean solar radiation pressure at 1 Astronomic Unit (AU), R is the relative position vector from the spacecraft center of mass to the Sun, A_i , ε_i and $n_{ext,i}$ are the single surface area, reflectivity coefficient and external surface unit vector, respectively. Finally, $\theta_i = \arccos(\hat{R} \cdot n_{ext,i})$.

When a body interposes between the spacecraft and the Sun, the former is partially or totally eclipsed, and the force reduces accordingly by a factor²

$$v(r_{sun}, r_{s/c}) = 1 - \left(\frac{\alpha a^2 + \beta b^2 - ac \sin \alpha}{\pi b^2} \right)$$

where

$$\alpha = \arcsin\left(\frac{b \sin \beta}{a}\right), \quad \beta = \arccos\left(\frac{c^2 + b^2 - a^2}{2bc}\right)$$

a , b are the Sun and occulting body apparent radii respectively, and c is the apparent distance between the geometrical centers of Sun and occulting body.

Thus, the overall force acting on the spacecraft is

$$\frac{dq_{sc}}{dt} = v \cdot p_{\odot} \frac{(1AU)^2}{\|R\|^2} \sum_{i=1}^{\#surf} A_i \cos(\theta_i) \cdot [(1 - \varepsilon_i)(-\hat{R}) + 2\varepsilon_i \cos(\theta_i)(-n_{ext,i})]$$

while the associated torque is computed accordingly, once the center of pressure of each and every surface composing the spacecraft geometry is defined.

The layout of the **SpacecraftDynamics** model is depicted in Figure 3. As can be seen from the Figure, the core of the component is the Body component of the Modelica MultiBody library, which describes the linear and angular motion of a rigid body. The component interface is constituted by a mechanical connector, to which mathematical models for the forces and torques arising from the interaction with the space environment are attached. Finally, a function for the computation of classical orbit parameters from the cartesian representation of the spacecraft position and velocity is included in the component model.

²The shadow function $v(r_{sun}, r_{s/c})$ is derived under the assumption of occulting body infinitely far from the spacecraft

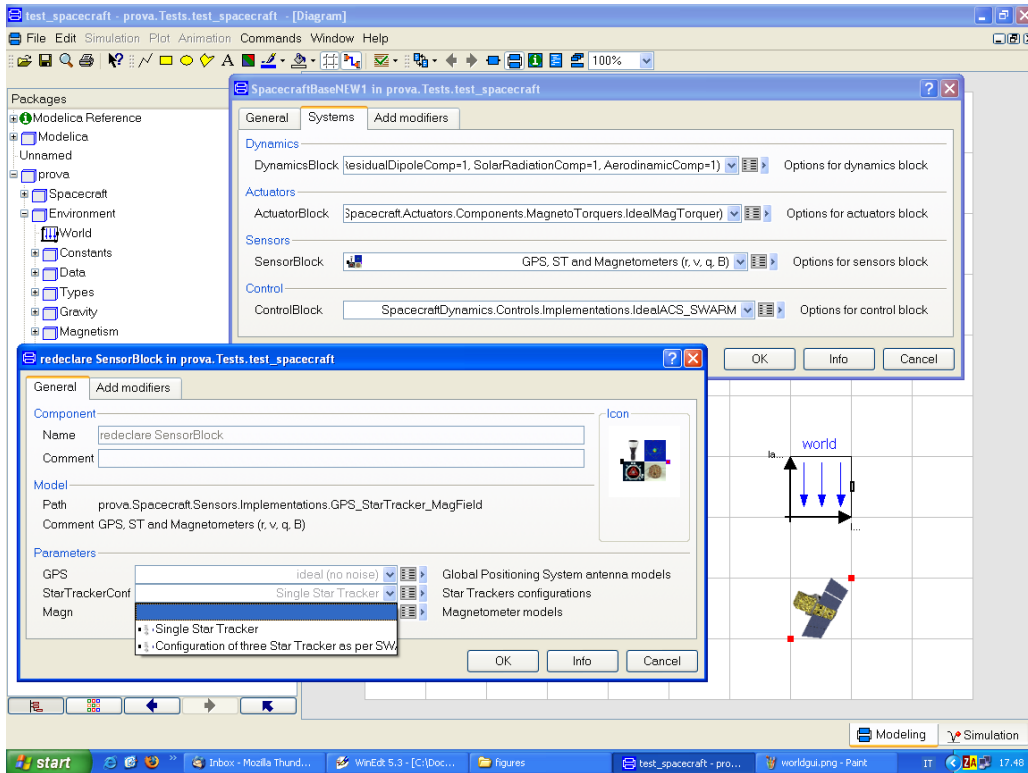


Figure 5: Spacecraft graphical user interface.

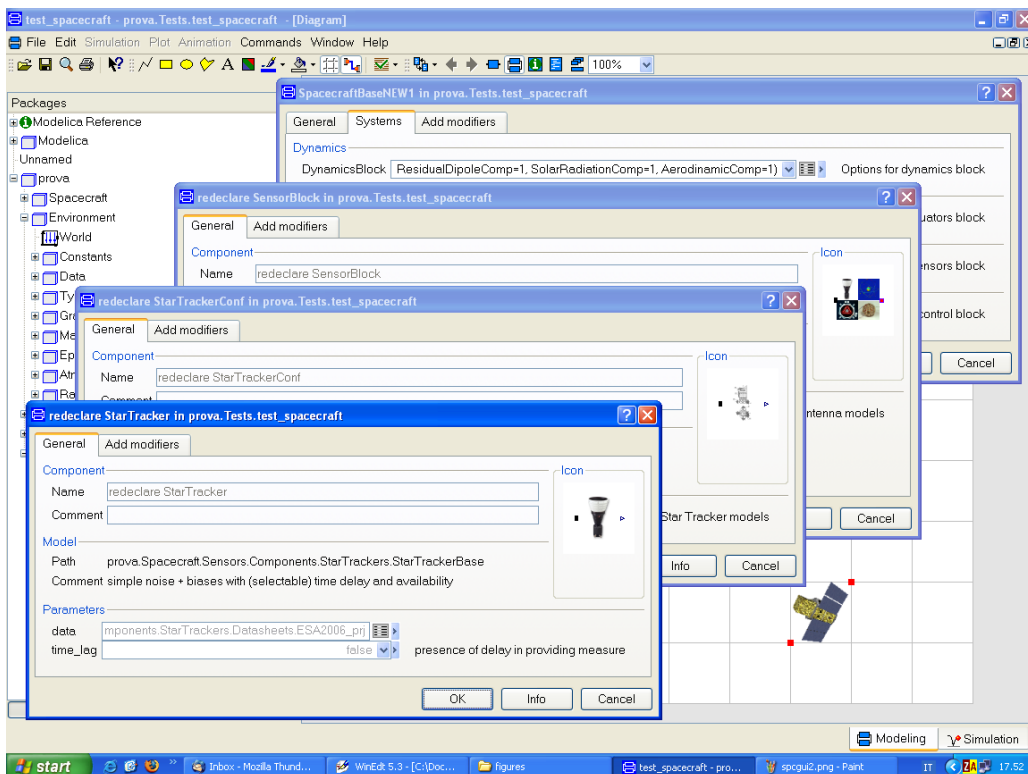


Figure 6: Selection of star tracker instance for **SensorBlock** replaceable model.

The spacecraft data (i.e., inertial properties, geometry, surface reflectivity, material, etc.) can be easily retrieved from appropriate spacecraft records. Moreover, to cope with classic space mission requirements, two initialization options are allowed:

- standard `Modelica.Mechanics.MultiBody.Body` initialization;
- a new initialization based on current simulation Universal Time (set within the extended World model), nominal orbit (specified by six orbital parameters, retrieved from appropriate records), angular rate and Modelica Orientation object relative to the orbital reference frame.

4 A case study

The Spacecraft Dynamics Library has been already used in a number of spacecraft modelling and simulation problems (see, e.g., [6] and the references therein). In this paper, the focus will be on the application of the library in the AOCS preliminary design stage, i.e., when the spacecraft architecture is not yet completely defined and different options are being evaluated, depending on the specific mission profile. At this stage, it is indeed convenient to maintain for the spacecraft a *higher level* structure (i.e., one where only requirements are specified, not specific equipments), to evaluate its interaction with the space environment for different mission profiles and to be concerned only afterwards about the choice of specific equipment to be embarked. One of the main tasks in this stage is to evaluate the external disturbance forces and torques acting on the spacecraft, depending on the mission profile. This task can be performed in many different ways: the simplest approach would be to rely on simple worst case formulas such as the ones given in [3]; on the other hand, one could think of running a closed-loop simulation using a simple attitude control algorithm to maintain the satellite near its nominal operating conditions (e.g., Earth pointing attitude). Clearly, the former approach will introduce a significant conservatism in the analysis, while the latter requires a preliminary design of the attitude control law, which may be time-consuming.

A better way of dealing with this task can be devised by taking advantage of the acausal nature of Modelica models: Dymola's symbolic dynamic inversion capability will be exploited for the preliminary assessment of the disturbance torques acting upon the spacecraft in its nominal orbit and attitude.

Taking advantage of the Space Flight Dynamics Library features, it is an easy task to derive a customization of the base **Spacecraft** model which can be used to perform this preliminary analysis: it is sufficient to *assemble* a new spacecraft model with no **ControlBlock** nor **ActuatorBlock**, to define an unknown *control torque* to be applied as input torque to the spacecraft and assign the desired spacecraft's angular rate time-history. Dymola will then take care of solving the resulting system of nonlinear equations to derive the control torques time history necessary to keep the spacecraft in its nominal attitude.

As an example, Figure 7 shows the computed disturbance torques experienced by an Earth-pointing satellite aligned with its orbital reference frame. The considered satellite is assumed to operate on a near polar orbit ($i = 86.9^\circ$ inclination), eccentricity $e = 0$, altitude of 450 Km and a corresponding orbital period of 5614.8 seconds. The satellite inertial properties are:

- Satellite mass $m = 500 \text{ kg}$
- Satellite inertia matrix [kgm^2]

$$I = \begin{bmatrix} 30 & 2 & -18 \\ 2 & 1080 & -0.1 \\ -18 & -0.1 & 1070 \end{bmatrix}$$

A default cubic geometry was assumed for the satellite, comprising six surfaces, each with 1 m^2 surface area, reflectivity coefficient $\varepsilon = 0.02$ and center of pressure located at the surface geometric center.

For simulation purpose, aerodynamic drag, solar radiation pressure and a residual magnetic dipole of 1 Am^2 upon each spacecraft's body axis were selected as disturbance torques, while default choices were selected for geomagnetic and gravity fields (i.e., magnetic dipole and J_2 respectively), Sun ephemeris and atmosphere model (i.e., Harris-Priester). The simulation was initialized at GMT 12 : 00, March 21, 2007. Note that the solar radiation disturbance torque experiences a sudden drop to zero when the Earth interposes between the spacecraft and the Sun, and takes back a nonzero value as soon as the spacecraft gets full Sun illumination.

5 Concluding remarks

In this paper the main issues related to the modelling and simulation of spacecraft dynamics have been described, the results obtained so far in developing Modelica tools for spacecraft simulation have been pre-

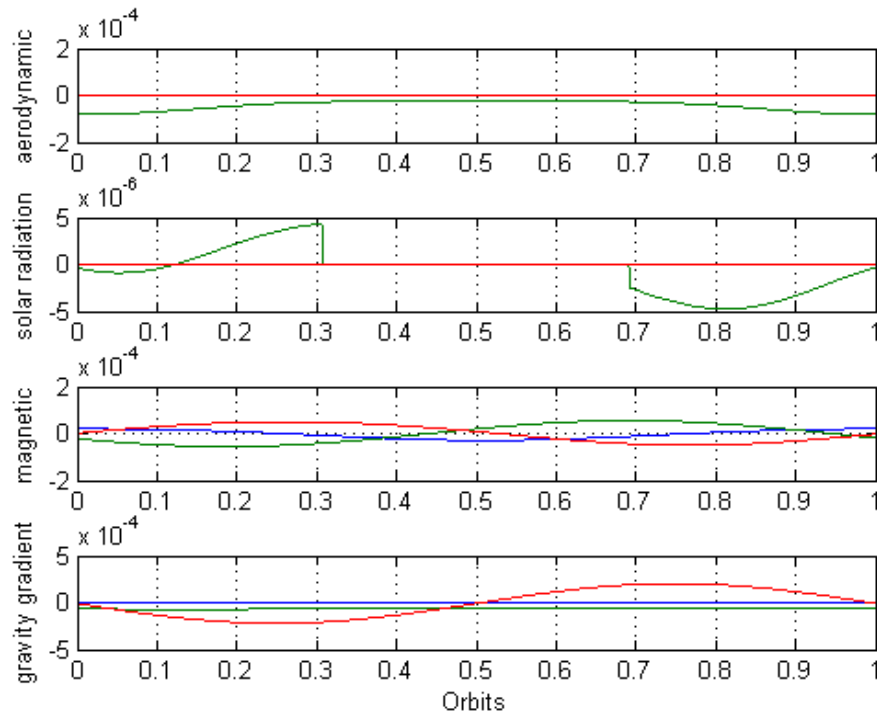


Figure 7: Disturbance torques experienced by the satellite in its nominal orbit and attitude.

sented and a case study demonstrating Modelica's usefulness and flexibility as a design tool has been discussed.

References

- [1] Modelica - a unified object-oriented language for physical systems modelling. Language specification. Technical report, Modelica Association, 2002.
- [2] G. Ferretti, F. Schiavo, and L. Viganò. Object-Oriented Modelling and Simulation of Flexible Multibody Thin Beams in Modelica with the Finite Element Method. In *4th Modelica Conference*, Hamburg-Harburg, Germany, March 7-8, 2005.
- [3] W.J. Larson, and J.R. Wertz, editors. *Space mission analysis and design*. Kluwer Academic Publisher, 1992.
- [4] M. Lovera. Object-oriented modelling of spacecraft attitude and orbit dynamics. In *54th International Astronautical Congress, Bremen, Germany*, 2003.
- [5] M. Lovera. Control-oriented modelling and simulation of spacecraft attitude and orbit dynamics. *Journal of Mathematical and Computer Modelling of Dynamical Systems, Special issue on Modular Physical Modelling*, 12(1):73–88, 2006.
- [6] M. Lovera and T. Pulecchi. Object-oriented modelling for spacecraft dynamics: a case study. In *IEEE International Symposium on Computer-Aided Control System Design, Munich, Germany*, 2006.
- [7] O. Montenbruck and E. Gill. *Satellite orbits: models, methods, applications*. Springer, 2000.
- [8] D. Moorman and G. Looye. The Modelica flight dynamics library. In *Proceedings of the 2nd International Modelica Conference, Oberpfaffenhofen, Germany*, 2002.
- [9] M. Otter, H. Elmqvist, and S. E. Mattsson. The new Modelica multibody library. In *Proceedings of the 3rd International Modelica Conference, Linköping, Sweden*, 2003.
- [10] M. Otter and H. Olsson. New features in Modelica 2.0. In *Proceedings of the 2nd Interna-*

tional Modelica Conference, Oberpfaffenhofen, Germany, 2002.

- [11] T. Pulecchi and M. Lovera. Object-oriented modelling of the dynamics of a satellite equipped with single gimbal control moment gyros. In *Proceedings of the 4th International Modelica Conference, Hamburg, Germany*, volume 1, pages 35–44, 2005.
- [12] F. Schiavo and M. Lovera. Modelling, simulation and control of spacecraft with flexible appendages. In *Proc. of the 5th International Symposium on Mathematical Modelling, Vienna, Austria, 2006.*
- [13] F. Schiavo, L. Viganò, and G. Ferretti. Modular modelling of flexible beams for multibody systems. *Multibody Systems Dynamics*, 12(1):73–88, 2006.
- [14] A. Turner. An open-source, extensible spacecraft simulation and modeling environment framework. Master's thesis, Virginia Polytechnic Institute and State University, 2003.
- [15] D. Vallado *Fundamentals of astrodynamics and applications*. Microcosm Press/Kluwer Academic Press, 2001.
- [16] J. Wertz. *Spacecraft attitude determination and control*. D. Reidel Publishing Company, 1978.

Session 2a

Thermodynamic Systems for Power Plant Applications 2

Simulation of Components of a Thermal Power Plant

René Schimon Dragan Simic Anton Haumer Christian Kral Markus Plainer
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 paper different models for simulating components of thermal power plants and other thermal or thermodynamic processes are presented. The different simulation results were performed with *Dymola* which is based on *Modelica*. The models were realized with time domain differential equations and algebraic equations. For all components the fluid was modeled by using the *Modelica.Media* library which is part of the *Modelica* standard library.

The heat transfer for the heat exchanger component was modeled by calculating the heat transfer coefficient in dependency on the flow velocity of the medium in the pipes.

1 Introduction

Arsenal Research currently works on the development of a simulation library for thermal and thermodynamic processes. Elementary problems like heat transfer and fluid dynamics will be processed. To a large extent the *Modelica_Fluid* library already covers a lot of important models, which are used to develop parts of thermal power plants. The new library will complete the *Modelica_Fluid* library. Some of the presented models are based on components of the *Modelica.Fluid* library. For some special technical problems the *Modelica_Fluid* library was modified and extended.

2 Extended models to be developed

Arsenal Research is working on specific problems related with heat transfer and fluid dynamics. The development of the following components is currently initiated:

- Centrifugal pump

- An ideal pump based on physical parameters
- A pump with losses based on physical parameter

- Simple pipe comprising pressure losses
- Heat exchanger including pressure losses, thermal convection, thermal conduction
- Turbine modeled with a characteristic curve

3 Components

3.1 Fluid flow machines

In a thermodynamic processes or thermal power plant fluids and different gases have to be transported through pipes. Pipes cause pressure losses. To generate a constant mass flow, pumps are needed. Yet, in some cases stationary density differences give rise to a constant mass flow without having a pump, like in a natural circulation boiler. For assistance or to start a fluid flow, pumps are the most important components in thermal processes. Pumps are fluid flow machines. Mechanical shaft energy is transformed into kinetic and potential flow energy. The impeller which is directly attached on the shaft, accelerates the fluid elements. Because of the diffuser effect in the shovel channels of the impeller wheel, the operating fluid leaves the impeller with increased pressure. Pumps represent a link between pressure increase and flow velocity or the mass flow. This context leads to the characteristic of a pump. Each pump typ has its typical characteristic in dependency on its geometry and rotation speed.

3.1.1 Ideal centrifugal fluid flow machines

An ideal pump is a fluid flow machines with an infinite number of shovels. The converted energy in the

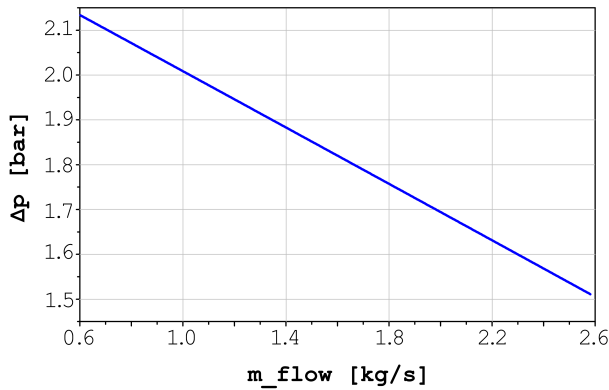


Figure 1: Characteristic of an ideal water pump

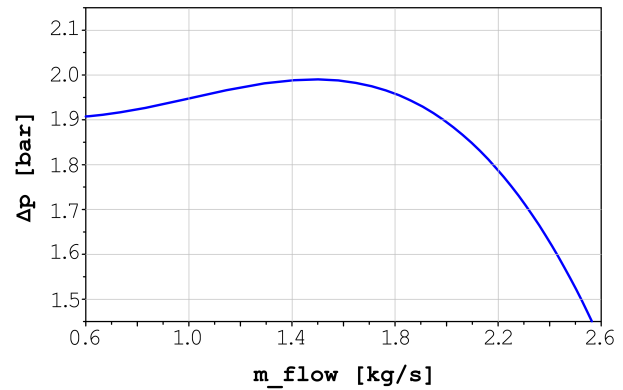


Figure 2: Characteristic of the centrifugal water pump with losses

impeller wheel of the water pump is the kinetic and potential energy of the operating fluid. Therefore the main equation of the water pump represents the proportion between mechanical energy of the input shaft with respect to the impeller wheel of the water pump and the specific energy of the operating fluid [1].

$$\dot{m} \cdot \omega = 2 \cdot \pi \cdot b_2 \cdot \rho \cdot \tan(\beta_2) \cdot (r_2^2 \cdot \omega^2 - Y_\infty) \quad (1)$$

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 shovel of the impeller wheel. Furthermore, \dot{m} represents the mass flow of the operating fluid in the water pump and Y_∞ represents the specific energy of the impeller wheel for an infinite number of impeller shovels. The ideal pump model was implemented *Modelica* using *Dymola* as simulation tool. Figure 1 shows the linear characteristic of the ideal water pump. Pumps as a special fluid flow machines is detailed processed in [2].

3.1.2 Centrifugal Pump

In realistic pumps, friction, fluid impacts and other fluid dynamic effects cause losses in the water pump. These losses can be split into:

- Decrease of the specific energy
- Hydraulic losses in shovel channels
- Impact losses
- Friction losses of the impeller wheel

A perviously developed model of an ideal pump [3] was used to design a water pump with losses. The new model now uses the *Modelica.Media* library to model the fluid. Figure 2 shows the characteristic of this model.

3.2 Initialization

In real thermal power plants, some components, like the turbine, have to be brought up to operational conditions by a well defined starting procedure. Reasons for this starting procedure can be load limits of certain components, cavitation of operating fluids or restriction due to the process.

In computer simulations, certain start values define the initial state of the simulation. Therefore, a simulation usually can be started from any state. From the particular starting point, the system should then reach a steady state condition. The initialization of a complex model becomes more difficult when using the *Modelica.Media* library.

The *Modelica.Media* library has certain operational limits. Outside these limits an error occurs. For example, the start-up procedure of a real pump is very complex. At the suction side of the pump the pressure decreases during the start-up procedure. Without controlling the process of starting up a simulation, like in a real system, an error may occur. This, however, may lead to a complex controlling due to the huge number of physical quantities to be controlled.

3.3 Pipe as an important part of Heat Exchangers

The pipe is one of the most elementary components. A lot of thermal power plant components are based on simple pipe models. One of these important components is the heat exchanger.

In a real pipe, the pressure drop of a fluid flowing through the pipe, decreases due to the wall roughness of the surfaces. The pressure drop also depends on the flow velocity and on the roughness of the inner surface of the pipe. The cross-section of the pipe also influences the pressure losses. For circular pipes the pressure drop can be calculated according to:

$$\Delta p = \lambda \cdot \frac{L}{D} \cdot \frac{\rho \cdot v^2}{2} \quad (2)$$

The pressure drop Δp is a function of a coefficient of friction, λ , the length of the pipe, L , the characteristic length, D , the density, ρ , and the velocity, v . The coefficient of friction, λ , depends on the *Reynolds* (Re) number. The Re number is defined by (3). The Re number defines the type of flow. The flow can be turbulent, laminar or in the transient area.

$$Re = \frac{v \cdot D}{\nu} \quad (3)$$

The pressure drop substantially depends on the flow type. For the laminar area applies:

$$\lambda = \frac{64}{Re} \quad (4)$$

For the turbulent area basically applies:

$$\frac{1}{\sqrt{\lambda}} = -2 \cdot \log\left(\frac{2.51}{Re \cdot \sqrt{\lambda}} + \frac{k}{3.71 \cdot D}\right) \quad (5)$$

In this equation k is the roughness of the inner pipe surface. Figure 3 represents λ versus the Re number for different flow types.

The *Modelica_Fluid* library contains a lot of different pipe models with different levels of abstractions. Arsenal Research develops extended models based on the components of the *Modelica_Fluid* library. Simultaneously, Arsenal Research developed simplified models for easy handling and modeling of test cases, incorporating less parameters than the comprehensive models. *Modelica_Fluid* and the extended and the simplified models are using the same basic equations, with respect to e.g. friction models, Reynolds equation, etc.

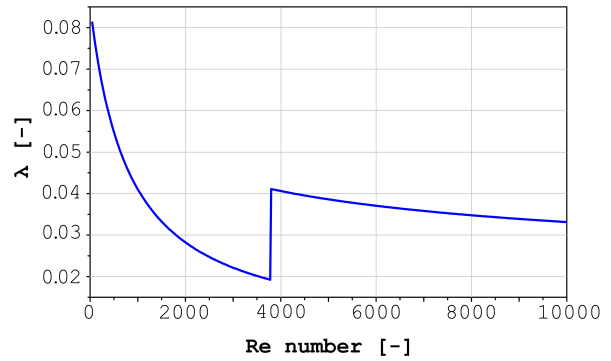


Figure 3: Coefficient of friction versus different flow types

Figure 4 shows the pressure drop versus Re number. The discontinuity indicates the transition from laminar (at low Re numbers) to turbulent fluid flow (at high Re numbers).

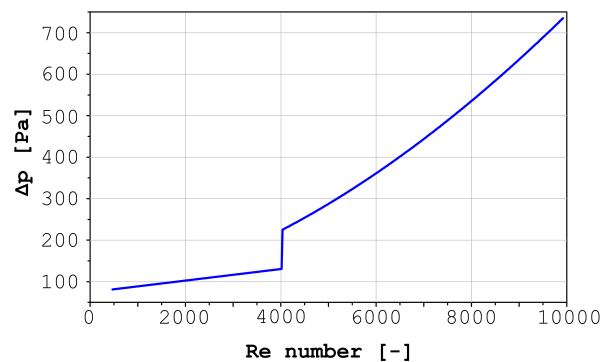


Figure 4: Pressure drop in pipes versus flow velocity

3.4 Heat Exchanger

Heat Exchangers are very important for nearly every thermal process. In heat exchangers usually two interacting fluid circuits are involved. The temperatures of both fluids are approaching. Yet a certain temperature difference is required to maintain the energy flow from the higher temperature fluid to the lower temperature fluid. Figure 7 shows a scheme of a pipe heat exchanger.

3.4.1 Heat propagation in Fluids

For modeling a heat exchanger it is necessary to understand how heat propagation in fluids works. Heat

propagation is described with partial differential equations in one dimension [4]. In standing fluids heat is transported by diffusion. The diffusivity a depends on the operating fluid. The thermal conduction equation for one dimension is shown in (6). In this equation q is an additional heat source, T is the temperature and x is the location. The diffusivity a is a fluid property, which is a function of the heat conduction λ , the density ρ and the specific heat capacity c , according to (7)

$$\frac{\partial T}{\partial t} = a \cdot \frac{\partial^2 T}{\partial x^2} + q \quad (6)$$

$$a = \frac{\lambda}{c \cdot \rho} \quad (7)$$

In flowing fluids the heat is also transported with the flowing medium, which depends on the fluid velocity. This leads to a remodeling of the heat propagation (6). Properties in a hydrodynamic flow depend on time and location. The temperature is the important property in this case. So the temperature depends on the time and the place in the pipe. The total differential of the temperature with respect to the time and the place has to be calculated according to (8). To substitute this context into (6) it is necessary to have an expression [5] for:

$$\frac{\partial T}{\partial t}$$

So function (8) has to be divided by dt .

$$dT = \frac{\partial T}{\partial t} \cdot dt + \frac{\partial T}{\partial x} \cdot dx \quad (8)$$

$$\frac{dT}{dt} = \frac{\partial T}{\partial t} + v \cdot \frac{\partial T}{\partial x} \quad (9)$$

After summarizing equation (9) and (6) we receive an equation for the heat transport in axial direction:

$$\frac{\partial T}{\partial t} + v \cdot \frac{\partial T}{\partial x} - a \cdot \frac{\partial^2 T}{\partial x^2} = q \quad (10)$$

3.4.2 Heat Transfer caused by Convection

To design a heat exchanger model, different physical effects have to be considered. It is necessary to know the quantity of heat flow which is transported from a

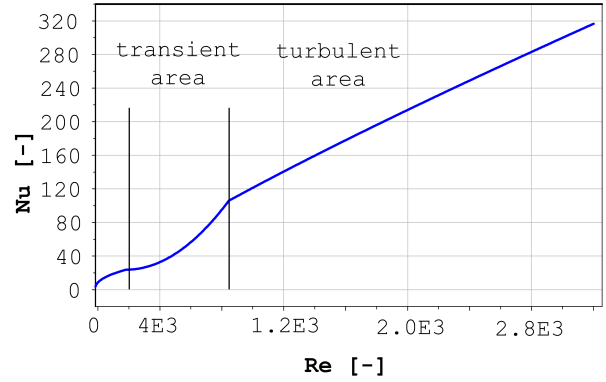


Figure 5: Nu number versus increasing flow velocity

fluid in the pipe to the outside surface. The heat transfer between the fluid and the pipe wall has to be modeled. This heat transfer happens by means of convection. The heat transfer coefficient describes the heat transfer between fluid and pipe wall, and is defined by the *Nusselt (Nu)* number according to (11). The *Nu* number is, however, a function of the *Re* number and of the medium properties. The *Re* number describes the type of flow (laminar, turbulent) and therefore, also the *Nu* Number depends on the type of flow.

$$Nu = \frac{\alpha \cdot L}{\lambda} \quad (11)$$

To determine out the quantity of heat, flowing between the fluid and the pipe wall, the Nusselt-Number has to be calculated. To transfer high quantities of heat flow high *Re* numbers are necessary. Figure 5 shows the *Nu* number versus the *Re* number.

The heat transfer coefficient α increases with increasing *Re* number. A salient turbulent flow is important to have a high quantity of heat transfer. In turbulent areas the heat transfer increases linear with the *Re* number [6].

3.4.3 Heat Transfer caused by Diffusion

As in shown in (6), the diffusivity is responsible for the heat propagation in fluids. The *Peclet (Pe)* number is the proportion between convective heat transport and heat transport through conduction:

$$Pe = \frac{d \cdot v}{a} = Re \cdot Pr \quad (12)$$

The Peclet-Number determines whether the diffusion is substantial or can be neglected.

3.4.4 Model of a Pipe

To design a detailed model of a heat exchanger, the model should consider all the above described physical behaviors. Different types of heat exchangers lead to different results. Every different types of heat exchangers needs different equations to describe the physical coherence.

- Parallel flow heat exchangers
- Counter flow heat exchangers
- Cross flow heat exchangers
- Plate heat exchangers

Each type of heat exchanger has a different efficiency because of their different heat transfer behavior. In *Modelica* continuous heat transfer is modeled by heat transfer between n infinite small pipes. Where n is the number of elements. To get a model of a heat exchanger, an infinite small pipe segment could be the main model. A simply unlagged pipe always has a heat transfer with the environment. Figure 6 shows a model of an infinite small pipe segment. The component `pipe`, models the pressure drop in this infinite small segment. The pressure drop is caused by the friction between the fluid and the inner wall of the pipe. The additional connector is a real vector of the size 5. This connector transmits the pressure p , specific enthalpy h , massflow m_flow , segment length L and the characteristic length d to the component `heat transfer`. The characteristic length is for a circular cross-section the diameter. The component `heat transfer` needs the parameter to calculate the Nu number, which defines the heat transfer coefficient α for an infinite small area. To get the whole heat transfer through a pipe surface, α has to be multiplied with the pipe surface A according to (14). The model `heat transfer` transmits G_x to the component `convection`. Now the energy flow Q_x between fluid and the pipe wall in radial direction caused by convection can be calculated by:

$$Q_x = G_x \cdot \Delta T \quad (13)$$

$$G_x = \alpha \cdot A \quad (14)$$

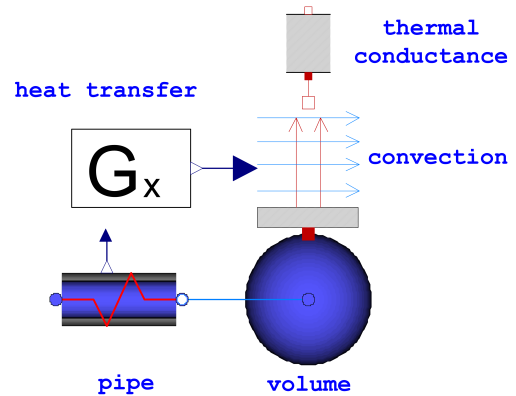


Figure 6: Model of an infinite small pipe segment

The component `thermal conductance` simulates the heat conduction through the wall of the pipe. The red line which is drawn in in the component `pipe` in Figure 6, indicates the heat transfer in axial direction caused by diffusion. The influence on heat propagation caused by diffusion is negligible small. High Pe numbers show that for usual technical applications the diffusion can be neglected. To minimize the *CPU-Time*, diffusion was not implemented in the heat exchanger model. At the outer port of `thermal conductance` an additional heat source could be simulated, considering e.g. solar radiation or a cooling environment.

3.4.5 Model of a Heat Exchanger

In a heat exchanger, a second fluid constitutes an additional heat source. Figure 7 shows the scheme of a parallel heat exchanger. The second fluid A, circumflows the pipe which contains the fluid B. Heat flow interchanges between both fluids. The heat flows from the fluid with higher temperature to the fluid with lower temperature. In a parallel heat exchanger the fluids flow in the same direction. At the inlet the temperature gradient is on its highest level and decreases towards the flow outlet. The temperatures of both fluids are approaching. Figure 7 shows that heat flow is transported from the hot fluid A to the second fluid B. The pipe has a mass which also has to be heated up, and therefore, the heat capacity of the pipe has to be considered. The efficiency of a heat exchanger is at its maximum, if the heat flow between both fluids is as high as possible. For this reason it is important that the pipe is a good heat conductor. The

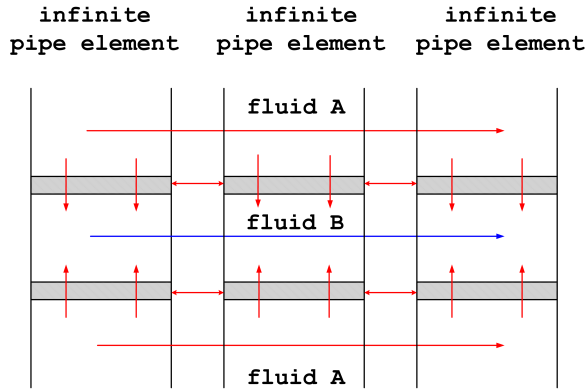


Figure 7: Scheme of a parallel tube heat exchanger

wall of the pipe conducts heat also in axial direction. This has to be respected in the heat exchanger model, because for some types of heat exchangers, this effect is very important. Figure 7 shows a scheme of a parallel pipe heat exchanger and Figure 8 shows the implementation in a *Modelica* model.

Figure 8 shows the model of an infinite small element of a finitely long parallel tube bundle heat exchanger. The model *volume* is directly taken out of the *Modelica_Fluid* library. It was equipped with an additional connector. This connector transmits some geometrical parameter of the pipe to the *volume*, so that the volume of the pipe segment can be calculated. The additional components, *conduction* and *capacity*, model the pipe with heat conduction in axial and radial direction. For a parallel heat exchanger these components do not have a significant effect. The second *heat transfer* component simulates the heat flow, which is transmitted from the outer fluid to the pipe. In the heat exchanger model the number of elements and the length of the whole pipe bundle has to be specified.

The model of one segment is connected *n* times one after another to model a whole heat exchanger. Figure 9 shows the temperature versus the length of the heat exchanger. This is an result taken out of a simulation with only 10 element. The model contains over 2000 algebraic and differential equation.

4 Conclusion

Simulations of components of thermal power plants or of other thermal processes are very complex. Especially the consideration of the complex media of the

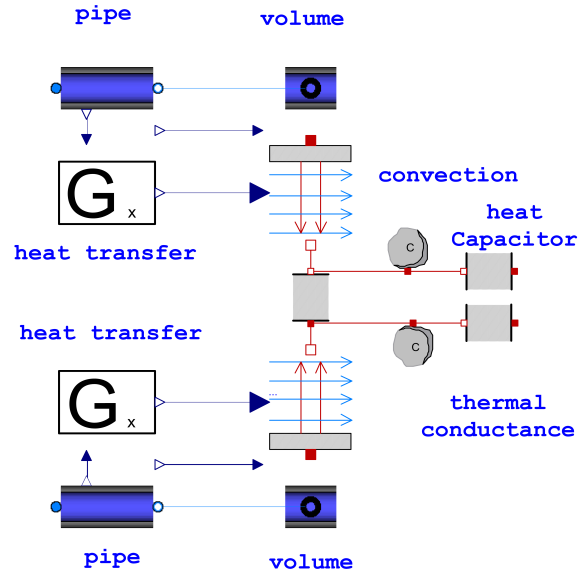


Figure 8: Model of a segment of a parallel heat exchanger

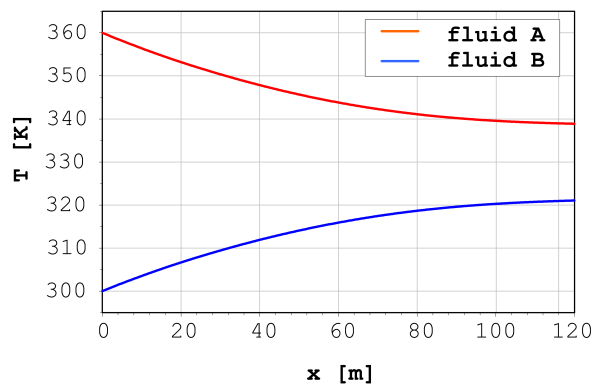


Figure 9: Temperature distribution inside the heat exchanger

Modelica.Media library leads to very detailed models and precise results. Nevertheless, the *Modelica.Media* library allows for a high precision of results. For some applications, however, it makes sense to model some components with a more simplified level of abstraction. Therefore it is necessary to carefully decide on the detail of abstraction for each model.

Abbreviations

CPU central processing unit

References

- [1] J. F. Gülich, *Kreiselpumpen*. Heidelberger Platz 3, 14197 Berlin: Springer Verlag Berlin Heidelberg New York 2004, 2004.
- [2] D. Simic, C. Kral, and H. Lacher, "Optimization of a cooling circuit with a parameterized water pump model," *5th International Modelica Conference 2006*, 2006.
- [3] 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.
- [4] H. Baehr and K. Stephan, *Waerme- und Stoffuebertragung*, vol. 5. Auflage. Springer, 2006.
- [5] G. Merker and C. Eiglmeier, *Fluid- und Waermetransport – Waermeuebertragung*. Stuttgart, Leipzig: B.G. Teubner, 1999.
- [6] H. O. jr., *Prandtl - Fuehrer durch die Stroemuungslehre Grundlagen und Phaenomene*, vol. 11. Auflage. Vieweg, 2002.

Pressurized Water Reactor Modelling with Modelica

Annick Souyri Daniel Bouskela
EDF/R&D
6 quai Watier, F-78401 Chatou Cedex, France
annick.souyri@edf.fr daniel.bouskela@edf.fr

Bruno Pentori Nordine Kerkar
EDF/SEPTEN
12-14 avenue Dutriévoz, 69628 Villeurbanne, France
bruno.pentori@edf.fr nordine.kerkar@edf.fr

Abstract

In order to optimize and validate the design and the operation of its nuclear power plants facilities, EDF (Electricité de France) uses a proprietary tool called LEDA to perform static and dynamic simulation at the system level. EDF wishes to replace LEDA by state-of-the-art off-the-shelf tools, mainly to reduce maintenance cost while keeping up with the latest trend in modeling and simulation technology.

To validate the feasibility of replacing LEDA by Modelica based tools, several benchmarks models have been chosen, that represent the variety of engineering studies made at EDF. The objective of this work is to show that these tools are fit to dynamic modeling and simulation of a PWR plant. To that end, a reference LEDA model of such plant has been successfully translated into Modelica and simulated using Dymola. The results of the Dymola simulation experiments are compared to those obtained with LEDA.

This paper describes the structure of the Modelica model, and the modeling and the numerical difficulties encountered during the translation and simulation process.

1 Introduction

For more than 20 years, EDF has been using modeling and simulation at the system level for the sizing, design verification and validation, and the operation of its nuclear and conventional thermal power plants.

To that end, EDF has developed and maintained since the early 80's a modular code called LEDA. LEDA is used for static (plant sizing) and dynamic

studies (modeling and simulation of the normal or incidental plant transients). It is an efficient tool, that has a complete model library and can solve direct and inverse problems. But, because of its now ageing architecture, it cannot keep up with the latest trend in modeling and simulation technology.

So, to improve the efficiency of its simulation tools while reducing their cost, EDF is studying the feasibility of using state-of-the-art readily available tools instead of LEDA code.

The replacement tools should at least have the same capabilities as LEDA, i.e. have an open component library, be able to perform static and dynamic studies, compute steady states and solve inverse problems. They also should not induce an excessive dependency upon the tool providers.

Modelica based tools offer such characteristics. That is why they are considered as good candidates to replace LEDA.

In order to evaluate the feasibility to replace LEDA by Modelica based tools, benchmark cases have been selected, which cover the variety of studies made at EDF. The first case to be studied was the quasi-2D modeling of a steam generator [1]. The next industrial case in the nuclear field to be studied, and objective of this work, is the dynamic modeling of a 1300 MW PWR power plant (P4).

The P4 LEDA model is a reference model used to study the behavior of the plant wrt. the power grid solicitations. In particular, it is useful for verifying the design of the control system against important transients, such as the house load operation.

This paper shows how the P4 LEDA model was translated into Modelica and tested with Dymola.

2 Description of the P4 model

The EDF nuclear plants belong to the PWR (Pressurized Water Reactor) type. For such type of nuclear plants, water acts both as the neutron moderator for the nuclear reaction, and as the heat transport fluid. Nuclear reaction occurs in fuel rods which are inserted into the vessel that contains water coming from the primary loop. The primary water is heated in the reactor core by the energy created from the nuclear reaction, while being maintained at high pressure in order to stay in the liquid phase. Hot water leaving the reactor vessel exchanges its heat with water coming from another circuit called the secondary loop. This heat exchange occurs through the steam generator, where primary water (in liquid state) and secondary water (in boiling state) are separated. The primary water is thus cooled and goes back to the nuclear reactor, while the secondary water heats up and becomes steam flowing to the turbine that drives the alternator to produce electricity. Steam leaving the turbine passes then through a condenser to go back to the steam generator as liquid feedwater.

The following circuits and components are included in the model: the primary loop and the pressurizer, point neutronic kinetics, the steam generator, the vapour line from the steam generator to the turbine admission valve, the steam generator feed water line. The main control systems are also taken into account in the model: the mean primary temperature control, nuclear power control, pressurizer pressure and level control, steam generator water level control, secondary pressure control, secondary power control, turbine power control.

A Modelica library of 0D and 1D thermal hydraulics components has been developed, based on the original equations of the equivalent Fortran LEDA model components. These equations are the basic mass, momentum and energy balance equations, completed with closure equations derived from empirical correlations valid for the operating domain under consideration. Steam generator and vapour lines are described by 1D models. Empirical correlations (heat transfer, pressure losses), adapted to the physical range of operation of PWR, have also been translated into Modelica. The Modelica components have even been improved when necessary: more stable numerical scheme for 1D thermal hydraulics, inertial terms added, more adequate correlations for heat transfer, ...

2.1 EDF Thermofluid Library

A thermofluid Modelica library is being developed at EDF. The objective is to provide the physical and technological model components needed for steady-state and dynamic simulation of nuclear and thermal power plants under normal and incidental operating conditions.

The library components must be able to describe single and two-phase flow, with heat transfer when needed, deal with zero and reverse flow, compressible and incompressible flow for water/steam, and smoke networks for thermal power plants.

The library uses a finite volume approach, based on the staggered grid scheme for space discretization, and the upwind scheme for the handling of flow reversal [2]. Both schemes are well suited for convection, which is the predominant energy transport law within the network. Discretization is performed along the main flow direction only (1D modelling).

The basic model components are divided into two groups: nodes and edges. Nodes represent mixing volumes such as tanks, boilers, splitters and mergers, etc. They implement the mass and energy balance equations. Edges represent flow resistant elements such as valves, simple pressure loss pipes, etc. They implement the momentum balance equations. The network is built by connecting edges to nodes in order to obtain a complete set of mass, energy and momentum equations with their closure equations, and automatically fulfil the numerical scheme requirements. Complex library components such as heat exchangers, evaporator pipes or steam generator are also built by assembling edge and node elements. A more complete description of the modeling approach chosen by EDF for the thermofluid library is presented in ref. [1].

It is also important to note that EDF has chosen not to use the Modelica inheritance mechanism, in order to keep the readability of the model: the complete set of equations can be found directly in the component model itself, instead of being scattered throughout the library when they are partially derived from super-classes.

2.2 Components of the P4 model

In order to build the P4 model within the physical limits described earlier, the following components have been developed:

- fluid flow in pipes (primary and secondary water loop),
- pressurizer (to maintain primary water as liquid),

- turbine,
- valve,
- pump and motor (primary loop),
- steam generator (secondary water loop), represented by a riser connected to an upper part (the dome),
- point neutronic kinetics (modelling nuclear reaction and resulting temperature).

As one of the main purposes of the model is to validate the response of the plant against the grid solicitations, the main control sub-systems of the plant have been also modelled. They have been tested separately in open loop, then connected to the P4 process model as shown in Figure 1.

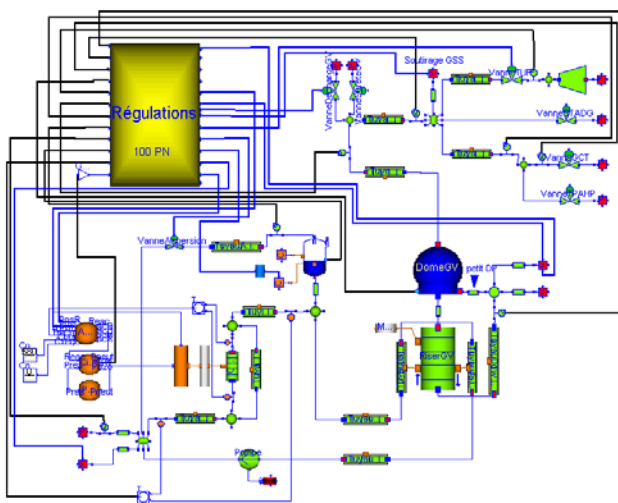


Figure 1: EDF 1300 MW Pressurised Water Reactor Modelica Model (P4)

2.3 Physics of the model

Fluid flow in pipes

A generic component has been developed. It can be used for different parts of the plant, either for the primary or the secondary loop. This model describes the behavior of a single phase fluid flow in one or several parallel conduits, where heat exchange can occur between the fluid and the internal metallic pipe wall, and between the external pipe wall and the outside environment. It is well adapted for the representation of connection circuits between the different equipments of the plant. Two-phase flow is only valid in the case of low vapor fraction, for no two-phase flow correlation has been implemented yet.

The model is based on mass, momentum and energy conservation equations, as 1-dimensionnal, partial differential equations. Discretization is performed along the main flow direction. The momentum conservation equation includes compressibility terms,

while fluid inertia and acceleration terms can be neglected as an option. Radial heat transfer in the pipe metallic wall is not discretized (single radial cell), and longitudinal conduction of heat is neglected in the fluid flow and in the wall. As mentioned earlier, like other components of the EDF Thermofluid library, a finite volume approach is used, based on the staggered grid scheme for space discretization, to ensure a better stability of the numerical scheme. Two types of cells are defined: “edges”, which solve momentum equations, and “nodes”, which solve mass and energy equations for the fluid, with heat transfer conduction equations in the metallic pipe wall.

The Dittus-Boelter correlation is used for the heat transfer coefficient between the fluid and the wall.

The pressurizer

The role of the pressurizer is to maintain the primary water pressure at a fixed level, in order to avoid vaporization within the primary loop. This is done by ensuring that the liquid and vapour states are always present in the pressurizer, so the pressure inside the pressurizer (and hence inside the primary loop) can be controlled by acting on the water temperature in the pressurizer. To do so, the equilibrium between water and steam is maintained in the pressurizer at the saturation temperature corresponding to the pressure setpoint by heating or cooling the water in the pressurizer. Heating is achieved by the electric heaters immersed in water, and cooling is achieved by condensing the steam by aspersion of water extracted from the primary loop.

So, there are two separate regions in the pressurizer: a liquid region and a vapor region. The physical model is based on a non-equilibrium formulation of the fluid balance equations for each region. The mathematical model is based on the mass and energy balance equations for the liquid and the vapor, plus a heat balance equation at the pressurizer wall, taking into account the heat exchange between the wall and the liquid, and between the wall and the vapor. Two closure equations are used for the evaporation and the condensation flow rates at the interface between the liquid and the vapor regions. The evaporation flow rate is related to the connected enthalpies (liquid and liquid and vapor at saturation conditions). The condensation flow rate is related to the connected enthalpies (vapor and liquid and vapor at saturation), to the heat exchanges wall/vapor and wall/liquid, and to the conditions of aspersion (flow rate and enthalpy). These two closure equations also use empirical coefficients, related to the bubble rising time in the liquid and the droplet falling time in

the vapor. Water and steam properties are computed from the IAPWS'97 formulations.

The turbine

The model is based on the Stodola law, which relates the flow rate through the turbine to the vapor conditions at the turbine inlet and outlet.

The valves

The model can represent the different types of valves to be found in the PWR plant. It calculates the flow rate through the valve as a function of the upstream and downstream pressure, the upstream enthalpy and the geometrical characteristics of the valve itself. It can represent a single phase (liquid or vapor) or two-phase flow. Special attention was given to the verification of the continuity between the different flow regimes. To that end, a non dimensional parameter analysis was used.

Pump and motor

The centrifugal pump model is based on the characteristic curves of the primary loop pumps in PWR type plants. No mass accumulation inside the pump is taken into account. Metal heat capacity and heat exchange with the outside are neglected. The mathematical model is based on the equation of variation of internal energy of the system, taking into account the mechanical power dissipation that heats the water flowing through the pump. Algebraic equations are also used. The first one is the relation between the rotational kinetics energy of the pump and the shaft speed, and the second is the energy balance of the fluid between the inlet and the outlet of the pump. The pump model is powered by an electric motor model.

The steam generator (SG)

The steam generator is a key component for the operation and the safety of the plant, because it is responsible for the cooling of the reactor.

The primary water flows into U-tubes and yields its heat to the secondary water. The secondary water, circulating outside the U-tubes, is liquid at the inlet of the SG, then flows down the outer part of the SG and starts to boil when reaching the bottom centre part of the SG, until the top of the boiling section. There, the ratio between the total flow rate and steam flow rate (circulation rate) reaches a value of 4 to 5 at nominal power. This part of the SG is called the riser, where the flow is mainly two-phase (a mixture of water and steam). Moreover, due to the non ho-

mogeneity of heat exchange inside the riser, two regions must be considered. When secondary water is flowing outside the first half part of U-tubes with hot primary water flowing in, the region is called "hot leg". When flowing outside the other half part of U-tubes with cooler primary water flowing in, the regions is called "cold leg".

The water and steam mixture passes then through separators where the two phases are separated in the upper part of the SG. The liquid part goes back to the SG feedwater, and the vapour part goes to the turbine. This part of the SG is called the dome.

The SG model has two different parts: the riser and the dome. Dedicated components have been developed because of the complexity of the flow, and the specificity of the geometrical characteristics of this type of heat exchanger.

The SG riser

The basic equations for this model are the same as the ones in the fluid flow model in pipes component described earlier. Options have been added to meet the specific needs of the riser:

- possibility to take into account two types of heat exchange for one cell of flow (one for the "hot leg" region, one for the "cold leg region"),
- implementation of a heat transfer correlation adapted to two-phase boiling flow for heat transfer coefficient (Thom correlation [3]).

The SG dome

As there are two separate regions in the SG dome (water and vapor), the basic equations for this component are the same as the ones modelling the pressurizer described earlier. The basic equations are the same for each phase. Evaporation and condensation flow rates are used as closure laws. Two equations have been added to calculate the flow rate entering the dome (pressure drop due to the separators between the riser and the dome), and the pressure at the entrance of the SG (the altitude of the entrance is lower than the dome, so there is a pressure drop due to gravity).

Neutronic kinetics

The model calculates the neutronic power generated in the fuel, as a function of the total reactivity of the core due to the coolant density effect, the fuel Doppler effect and the effect of boron concentration. It is a point reactor kinetics balance equation that describes the evolution of a neutron population, including the effect of precursors concentration leading to delayed neutron sources.

3 Simulation

3.1 Validation of the model

The reference benchmark test for this model is a house load operation. It is a high amplitude transient, that occurs when the plant is suddenly disconnected from the normal energy discharge network. This transient is used to check both the global operation of the reactor with the control system in operation, and the physics taken into account in the model.

The transient starts with neutronic and thermal hydraulic parameters set at values corresponding to the full power operating conditions of the plant. This means that before starting the transient, a stable regime must be reached at nominal conditions. To do so, the initial state for the model is calculated by (1) setting all the time derivatives to zero (simulation must start from steady state) (2) performing inverse calculations in order to adjust the parameter values to start the simulation from the nominal conditions.

The transient scenario is the following:

- At $t = 0$, the plant is disconnected from the grid by opening manually the electric circuit breaker. The turbine control system then closes the Turbine Admission Valve in about 1 minute. This leads to a vapor pressure rise, and consequently the opening of the bypass turbine condenser group valves.
- During the first minute, the nuclear power decreases rapidly, because of the insertion of the control rods in the core. During this phase, the pressure, water level and temperature of the primary loop show sudden rises due to the momentary deficit of the secondary load, then the thermal power balance between the primary and secondary loops is restored through the steam generator.
- After about 1 minute, the temperature control system leads to a stabilization, then a partial extraction of the temperature control rods. This slows down the decreasing rate of nuclear power, which stabilizes at about 30% of nominal power at the end of the transient. This value is reached in about 10 minutes, which is the time needed for the control rods to hit their setpoint.

3.2 Results of dynamic simulation

In order to cover the whole transient, the simulation time has been set at 2000 seconds.

The model has 7000 unknowns and 320 states.

The computing time is 600 seconds with a fixed time step solver, about 3 times faster than real time. Calculations were performed on a Pentium 4, 2.4 Ghz, with 512 Mo of CPU memory.

In order to reach this computing time, adjustments of phenomenological time parameters describing the actuators dynamics have been necessary. Also, numerical difficulties have been encountered due to the computing of the control rods insertion. Rods are inserted step by step. It is a discontinuous process controlled with hysteresis that trigger frequent threshold crossings. The number of resulting event detections to be computed turned out to be very large, leading to unacceptable computing time with the variable time step solver DASSL. This is why implicit fixed time step solver had to be used, in order to decrease the computing time of state event detection to acceptable level.

A previous validation of the model was made with the LEDA code against on-site experiments and transient recordings. Since the physical modelling is very close to the one implemented in the LEDA model, validation of the equivalent Modelica model was performed on the basis of the results obtained with LEDA. The following figures show the evolution of the main variables of the model versus time (in seconds); dotted lines are for LEDA simulation results and continuous lines for Modelica simulation results.

Figures 2 and 3 show the position of the control rods in the core. The two groups are inserted first, then the temperature control group is extracted again when the nuclear power has sufficiently decreased, trying to compensate for the primary loop mean temperature decrease (see also Figure 4 and 6).

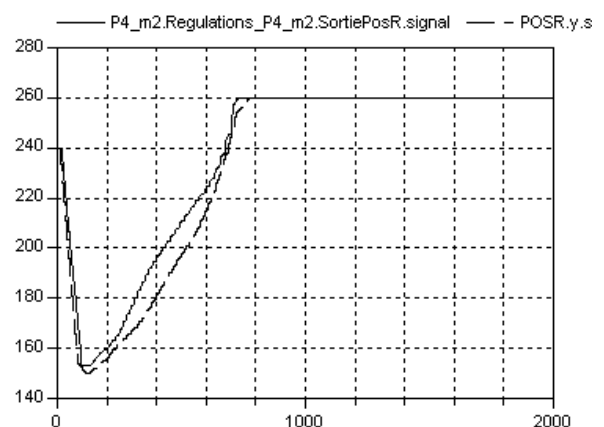


Figure 2: Position of Temperature control rods

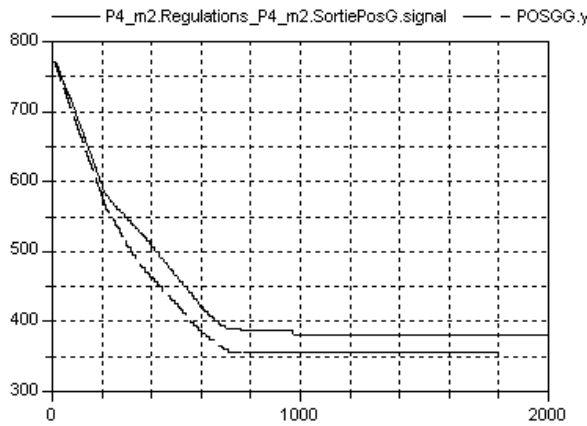


Figure 3: Position of Power control rods

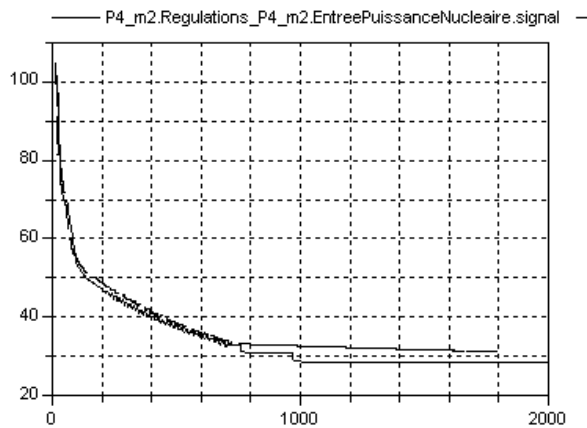


Figure 4: Nuclear Power

Figure 5 shows the evolution of the position of the bypass turbine condenser group valves, that open at the beginning of the transient, just after the closing of the turbine valve.

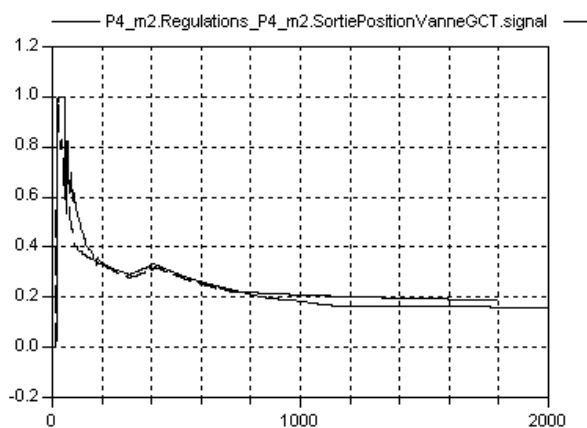


Figure 5: Position of by pass turbine condensers group valves

As previously written, the primary loop pressure and temperature exhibit a sharp rise at the start of the transient, due to the deficit of secondary load. Figure 6 shows the primary loop mean temperature

peak. However, if the overall dynamic evolution is correct, there is a noticeable difference between the maximum temperature calculated with LEDA and the one given by the Modelica model.

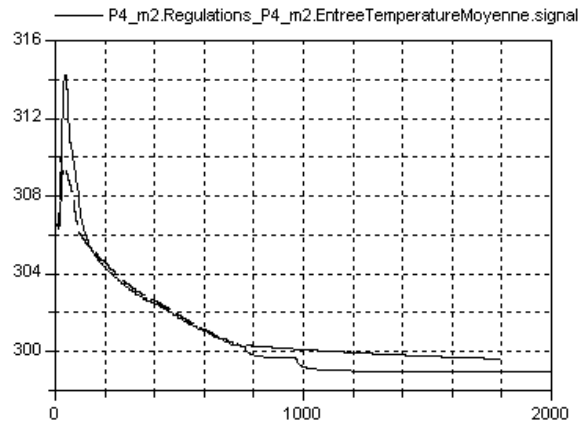


Figure 6: Primary loop mean temperature

The SG level is an important variable of the secondary loop, which is taken into account in the control and safety systems of the plant. The shrink and swell phenomenon is a variation of the water level in a two-phase fluid container, that occurs after a sudden change in vapor pressure or flow rate entering the container. This phenomenon is encountered in the SG, and must therefore be described by the model. As shown in Figure 7, the first shrink of water level in the SG is simulated, followed by a swelling of the water level before stabilization.

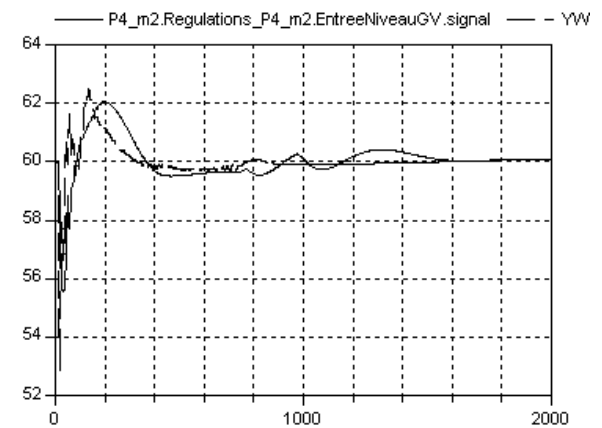


Figure 7: Water level in the Steam Generator

These results show that the dynamic response of the plant process and control system modelled with Modelica is quite satisfactory. However, a more complete validation of the model is needed, with a better (optimised) adjustment of the model parameters.

There are noticeable differences between the Modelica and the LEDA model. The LEDA model has a 1D neutronic model that takes into account the non

homogeneity of nuclear power in the core, whereas the Modelica model has a simple point neutronic model. Figure 2 and 3 also show differences between the control rod positions, which may be due to differences in the control rod calibrations.

These modelling approach differences could explain the discrepancies observed in the simulation results from the two models.

4 Conclusions

A full model of a PWR 1300 MWe plant has been translated from LEDA to Modelica. It is a large dynamic model, that exhibits numerical difficulties due to the large number of states, and the step-by-step discontinuous operation of the control rods that lead to frequent state events.

Computing time has been reduced to an acceptable level by using an implicit fixed time step solver instead of the variable time step solver DASSL.

It would probably be possible to reduce computing time even more by a thorough analysis of the model equations, but due to the large number of equations, this is a difficult task. So further development of Modelica based tools should address the methodology issue of modelling large dynamic systems by e.g. giving to the user the possibility to perform incremental model development and analysis.

However, this study shows that it is possible to perform dynamic simulations of a PWR plant with Modelica. The next step will be to test the coupling of such a Modelica model with an existing non-Modelica code (e.g. neutronics code).

References

- [1] Avenas C. et al, "Quasi-2D Steam Generator Modelling with Modelica", ISC'2004, Malaga, Spain.
- [2] Patankar S.V., "Numerical Heat Transfer and Fluid Flow", Hemisphere Publishing Corporation, 1980.
- [3] Thom J.R.S., Walker W.M., Fallon T.A., Reisting G.F.S., "Boiling in subcooled water during flow up heated tubes or annuli", Proc. IME (London), vol. 180, pp 226-246, 1955-56.

Simulation of the Start-Up Procedure of a Parabolic Trough Collector Field with Direct Solar Steam Generation

Tobias Hirsch Markus Eck
 German Aerospace Center (DLR)
 Pfaffenwaldring 38-40, 70569 Stuttgart, Germany
 tobias.hirsch@dlr.de, markus.eck@dlr.de

Abstract

Solar thermal power plants are one of the most interesting options for renewable electricity production. For a plant based on parabolic trough collectors, the start-up procedure of the solar field in the morning has to be well defined in order to start electricity production as soon as possible. In this paper, the Modelica language is used to describe the thermo-hydraulic components of the collector field. A control system for the plant start-up is developed based on the Modelica *StateGraph* library. With this simulation model a number of studies are performed to estimate the time consumption of the start-up procedure.

Keywords: parabolic trough; solar; control

1 Introduction

Among the wide range of renewable energy technologies for electricity production, solar thermal power plants are one of the economically most interesting options. Direct solar irradiation is concentrated into a focal point or focal line by curved mirrors. Parabolic trough collectors use the high temperatures to heat up a fluid in absorber tubes arranged in the focal line. The use of an organic oil as a heat transfer fluid is state of the art. Benefits are expected by directly evaporating and superheating water in the absorber tubes (direct solar steam generation, DISS) [1]. For the implementation of a first plant, the aspect of start-up and shut-down has to be solved. While the power block itself might operate through the night by using a thermal energy storage or an auxiliary fossil boiler, the solar field cools down at night.

As long as the solar field has not reached its operating point in terms of pressure and temperature, the steam turbine can not be started. Table 1 illustrates the impact of the start-up procedure duration on the levelized costs of electricity. If the procedure can be

shortened by 1 hour a net gain of 7.9 % will be obtained.

This paper presents simulation studies covering the topic of solar field start-up procedures for parabolic trough fields with direct steam generation. Especially the two-phase flow conditions inside the absorber tubes necessitate the analysis with the help of detailed numerical simulations. Having a simulation tool at hand, different start-up strategies can be tested and evaluated to come to an optimal solution. The Modelica language is used for the study since combination of hydraulic, solar and control components can easily be achieved. Some central aspects of the model and the results of the numerical simulation will be presented in the following.

Table 1: Impact of plant start-up time on the costs of electricity generation.

Start-up time	0 min	30 min	60 min
Relative costs	100.0 %	103.5 %	107.9 %

2 General structure of the model

To cover the central aspects of the simulation task, the model is split into two parts,

- the plant layer and
- the control layer.

In the plant layer all hardware components like pipes, absorber tubes, tanks and junctions are represented. The control layer incorporates all elements of the control system, i.e. controllers, parameter definitions, set-point tables and the process sequence control. The control layer is defined as

```
model StartUp_Controller_Var1
  ...
end StartUp_Controller_Var1;
```

An instance of this model named Control is created in the plant layer.

```
model plant_layer_design_01
  StartUp_Controller_Var1 Control;
end plant_layer_design_01;
```

The exchange of data between the two layers is enabled by a data bus which is defined as a connector element.

```
connector Bus
  Real Signal_Real[n_Signals_Real];
  parameter String[:] Names_Real =
    fill("n.a.",n_Signals_Real);
  parameter Integer n_Signals_Real =
    size(Names_Real,1);
end Bus;
```



The array Signal_Real hosts the data, while the parameter Names_Real allows the specification of names for the single data channels. An instance of the connector class is defined as an inner variable in the plant layer and as an outer variable in the control layer. In Dymola, the size of the array and the descriptions can easily be defined via the graphical user interface. For connecting real signals to the bus, a port is defined as:

```
model BusPort_Real_In

  outer DissDyn.Signal.Bus Bus;
  Modelica.Blocks.Interfaces.
    RealInput u;
  parameter Integer SignalNumber
    (min=1, max=Bus.n_Signals_Real);
```

```
equation
  u=Bus.Signal_Real[SignalNumber];
end BusPort_Real_In;
```

The parameter SignalNumber identifies the bus channel that should be connected to the signal plugged to input u. To graphically distinguish between ports, that assign a signal to a bus variable and ports that readout a bus variable, two port definitions

Real_In  and Real_Out  are used. In parallel to the real bus variables, Integer and Boolean variables are included in the bus in the same manner. The graphical annotation of the signal number is helpful for the setting-up and checking of the model.

3 Plant layer

The central element of the plant layer is the hydraulic circuit composed of buffer tank, feedwater header, absorber tubes and live steam header. Figure 1 illustrates the arrangement of the components for the reference configuration of a 5 MW_{el} solar field [3]. From the seven parallel rows of the plant layout, only one is modeled in the simulation together with mass flow multipliers at the inlets and outlets. Between the evaporation and superheating section a phase separator is arranged to allow recirculation operation. The water separated from the steam is transported by means of a drainage line to the buffer tank from where it is pumped back to the inlet of the field. During normal operation, the connection from live steam header outlet to the buffer

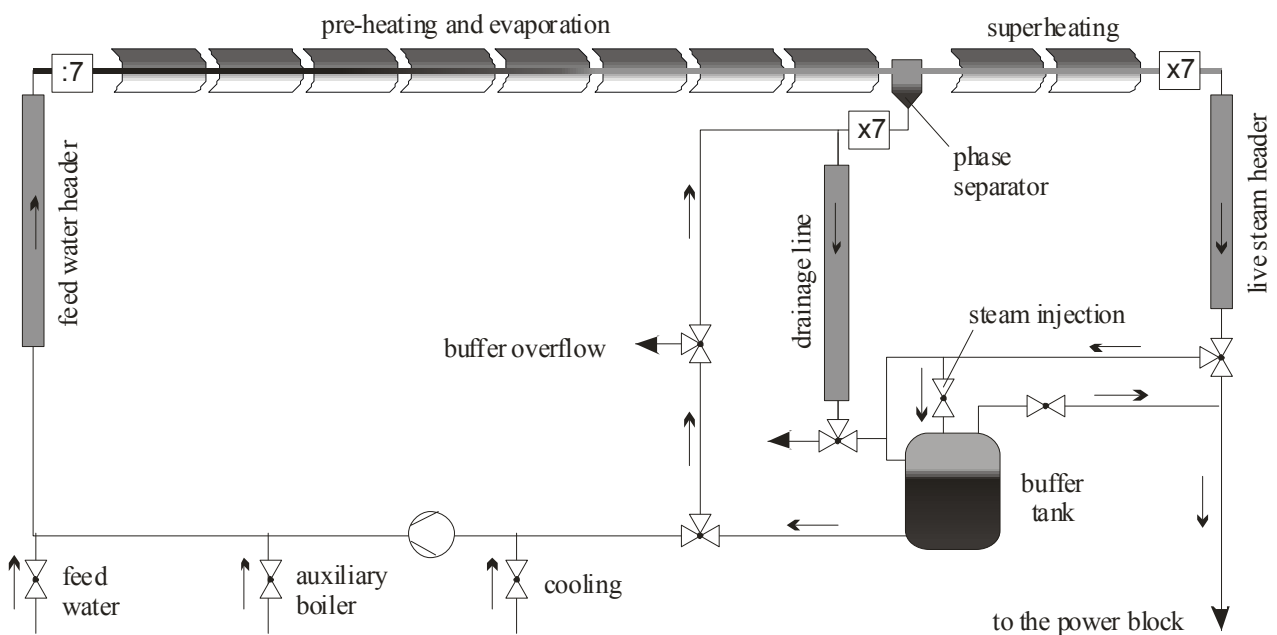


Figure 1: diagram of the plant layer

tank is closed so that the superheated steam directly flows to the power block. During the start-up phase this connecting line is opened and the drainage line closed. This allows recirculation of water over the whole system (global recirculation). The recirculated water is mixed with fresh feed water and, if necessary, with hot water produced by an auxiliary boiler. Another input line for cold water is added to cool the recirculation pump and thus to avoid cavitation in this device. The buffer tank is designed to hold water and steam at saturation conditions. There is one drain for the water at the bottom and one for steam at the top of the tank. Table 2 gives the geometrical parameters for the components used in this study.

The components models are taken from the *DissDyn* library developed at DLR [2]. For all pipe components, a one-dimensional discretization is used with the state variables pressure, specific enthalpy and wall temperature in each control volume. As boundary conditions, mass flow and specific enthalpy have to be provided at the inlet and pressure at the outlet. For the plant configuration, this means that a time dependent pressure boundary condition is required at the outlet of the live steam header. In the first part of the start-up procedure the flow from the field is sent to the power block. In the later stage of global recirculation, the directional control valve is switched to lead the flow into the buffer tank. For the first case, a constant pressure can be used as a boundary condition. For the second case, the boundary condition is directly coupled to the pressure in the buffer tank. A special component is designed to switch the boundary condition from a constant value to the pressure in the tank by means of a ramp. For the recirculation pump it is assumed that the power always fits to the pressure lift over the pump.

From the buffer tank steam can be extracted to the power block. If the buffer pressure falls below a given limit, steam from the field outlet can be in-

Table 2: Geometrical parameters of the plant components, length l , inner tube diameter d_i , outer tube diameter d_a , number of elements for spatial discretization n .

	l [m]	d_i [mm]	d_a [mm]	n [-]
feedwater header	210	60	70	12
evaporator	812	55	70	61
superheater	200	55	70	17
live steam header	210	100	120	12
drainage line	350	100	110	10
buffer tank	10.2	1000	1050	1

jected for stabilization. To keep the buffer liquid level within the allowed limits, water can be taken off into the buffer overflow line which leads to the power block or to the inlet of the drainage line.

All valves shown in the plant diagram are implemented as mass flow definitions. This approach avoids the high effort of using valve characteristics and the corresponding controllers.

4 Control layer

The start-up procedure is treated as a directed sequence of processes which themselves consist of a sequence of sub-processes. To model this structure, the Modelica *StateGraph* library is applied where each state element represents the corresponding process or sub-process. A state can either be active or inactive. A state element and its successor in the chain are linked by a transition. In case the state is active and its successor is inactive, the transition is enabled. This means, that the transition will fire as soon as a Boolean condition becomes true. After this event, the successor is active and the state has fallen back to inactive. Providing the first state with the active attribute this attribute will be passed through the whole sequence of states, and indicates which process in the system is running at each moment. The transitions between the states are linked to Boolean criteria described by system variables and switching conditions. A possible transition might be that a certain temperature or pressure is reached in the system. Figure 2 shows a screenshot of the control layer with the 9 main processes passed through during the simulation from top to bottom. Some of the main processes are divided into sub-processes by means of a parallel element. At this element, the path, and with it the active attribute, diverges into two branches where one of them exists of just a single state and the other is composed of a sequence of states. The transition that follows the parallel element, becomes enabled not before the final states in each branch both become active. This structure is chosen, since some of the processes require a number of minor steps that have to be passed before the process itself is finished. For analysis of the simulation results it is useful to define a subprocess identifier that represents the actual system state in terms of a numerical value

$$S = \sum_i state[i] \cdot active \times weight[i] ,$$

with the *weight* being an array with monotonically increasing values.

While the described structure helps to define the actual state and its chronological sequence, the input signals for the plant are generated in a number of control systems. The following elements are used to react on a change in the system state:

- On/Off switch that is triggered by two input signals “switch to on” and “switch to off”
- Switch that chooses between two real inputs depending on the status of the Boolean input signal
- Set-point table that selects the output value from a set of Boolean expressions and their dedicated real values
- Real expressions incorporating Boolean expressions that checks if one of the assigned states is active.

These components are completed by control elements that do not react on the system state itself but on characteristic variables of the systems:

- Hysteresis and min/max elements that check if real input signals extend given limits
- Proportional-integral controllers
- Algebraic expressions depending on a number of input variables

Without going into detail on the individual control loops table 3 summarizes the main control activities and the type of control applied for this task.

5 Simulation results

The developed start-up strategy will be described together with the simulation results. Nevertheless, some aspects like the boundary conditions should be treated beforehand.

5.1 Boundary conditions

It is assumed that from the shut-down procedure of the last evening warm water is stored in the feed water tank (6 bar, 154°C, 20 m³) and in the buffer tank (60 bar, 275°C, 6.4 m³). This water can be used to pre-heat the solar field. An auxiliary boiler is available that generates water or steam at different pressure levels. In the morning, the temperature in the field has fallen to 30°C. The whole volume in the field, except the buffer tank, is full of water.

5.2 Simulation of the start-up procedure

Figure 3 shows the results of the start-up simulation. At time t=0 the collectors are focused. Up to this point, the field is pre-heated. The collectors are fo

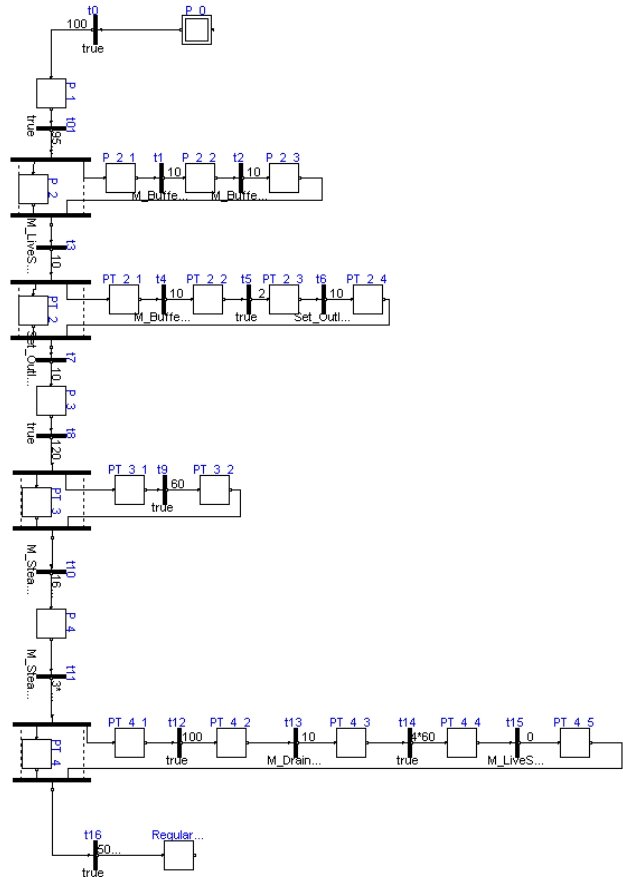


Figure 2: Screenshot of the control layer composed of state, transition and parallel elements from the StateGraph library.

Table 3: Main control tasks and types of control element used for them

feed water mass flow	PI-controller algebraic expression on/off switch set-point table
feed water temperature	algebraic expression set-point table
recirculation mass flow	set-point table
buffer pressure	PI-controller
buffer overflow	hysteresis element
recirculation cooling	algebraic expression
auxiliary power	algebraic expression
collector focusing	on/off switch
separator drainage	on/off switch

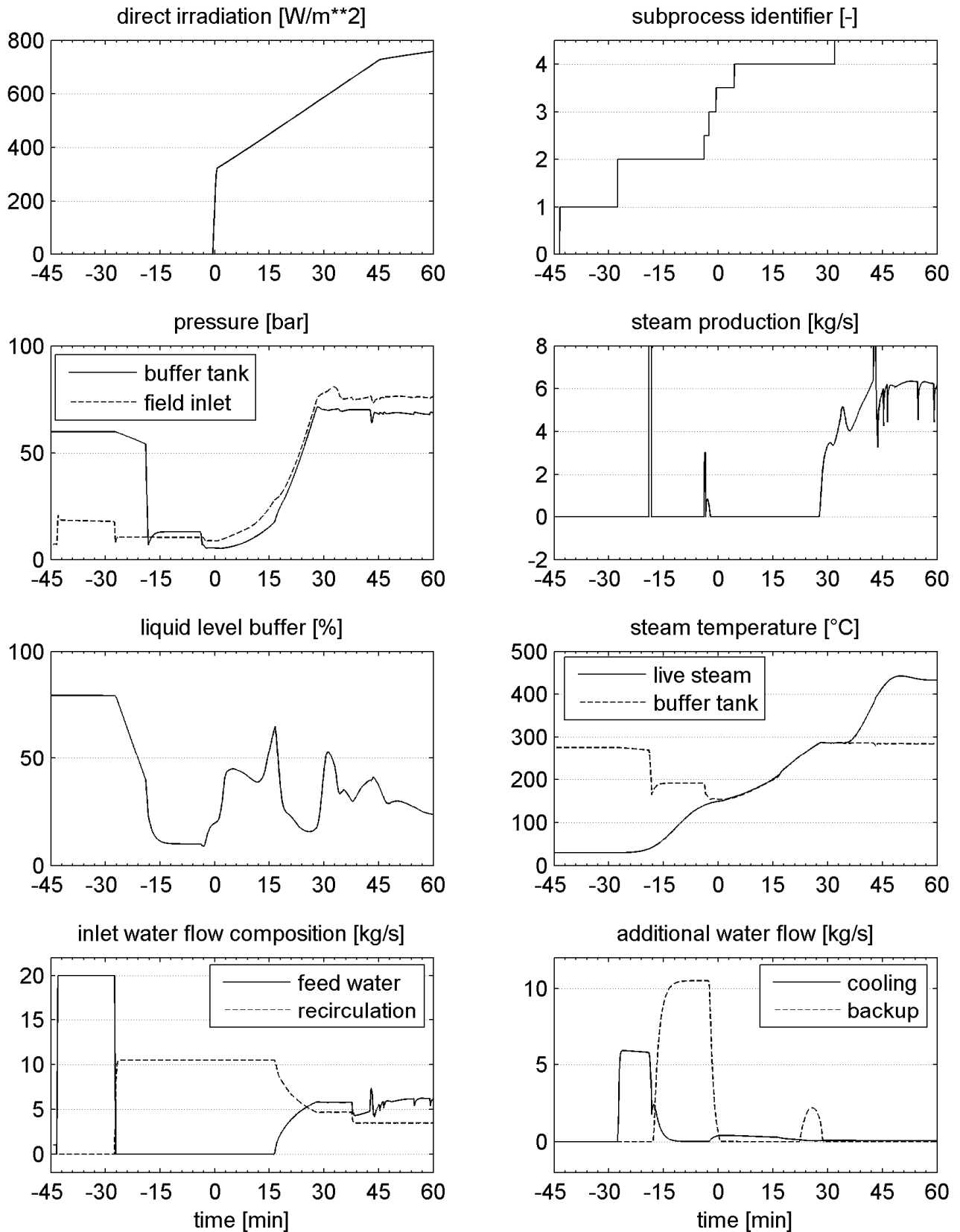


Figure 3: Start-up procedure in terms of important system variables.

cused at a solar altitude of 10° (at time $t=0$ min). Up to an altitude of 20° (at time $t=45$ min) shading between collectors occurs. The direct solar irradiation signal is corrected by these two effects that reduce the effective solar input on the collectors. The impact of the shading is illustrated in figure 4.

Pre-heating the field with warm water from the feed water tank

As a first step, warm water from the feed water tank is sent into the field ($t=-42$ min). From the temperature signal it can be seen that the pre-heating does not reach the end of the field when the feed water tank is completely emptied (at $t=-27$ min). The pressure is remained on a low level since evaporation in the field should start as early as possible. The water pushed out from the field is directed to the power block.

Pre-heating the field with hot water from the buffer tank

The water from the buffer tank is mixed with additional cold water to bring the temperature below the saturation temperature in the field. This is necessary to avoid evaporation in the feed water header which would otherwise prevent a controlled distribution of the water on the parallel channels. The flow into the field is fixed at 10.5 kg/s from which about 4.5 kg/s originate from the buffer tank. At $t=-17$ min the buffer tank falls below its minimum level but the temperature at the field outlet is still low. Redirecting the flow from the field outlet into the buffer tank would lead to a pressure decrease in the tank. For this reason, the pre-heating of the field is continued with water from the auxiliary boiler (named backup boiler in the figure) until the field outlet temperature reaches 150°C . This condition becomes true at $t=-3$ min.

Global recirculation without solar input

The connection between field outlet and buffer tank is opened and recirculation over the buffer tank is started. The pressure signals from buffer tank and field inlet are now linked. Since water of 150°C is mixed with the water of 165°C in the buffer tank, a slight temperature drop and, in the consequence, pressure drop in the buffer occurs.

Global recirculation with solar input

After 3 min of waiting time, the collectors are focused and solar heat input into the system starts. No steam is extracted from the system so the pressure in the field continuously rises. When the first steam is produced in the absorber tubes, large amounts of water are displaced and sent to the buffer tank. Since outflow of the tank is nearly constant, the liquid level rises and finally triggers additional water extraction

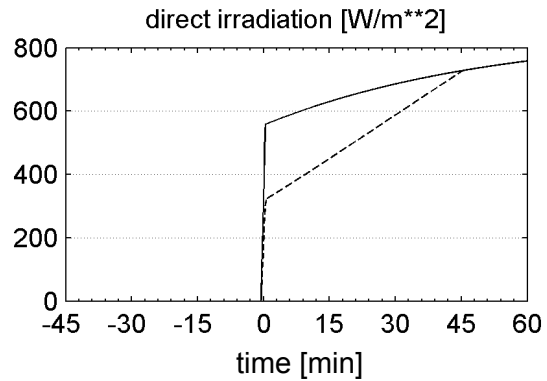


Figure 4: Impact of shading (- -) between two parallel collector rows on the effective direct irradiation compared to a stand-alone collector (---).

from the tank. The maximum water temperature at the field inlet is set to a value of 200°C that corresponds to the nominal operating conditions. From a tank pressure of 16 bar on, the recirculated water exceeds this temperature limit. For maintaining the desired temperature, cold feed water is mixed with the hot water from the buffer tank. The mass flow of recirculated water is reduced to maintain a constant total flow. This effect can be clearly seen from the feed water and recirculation water signals in figure 4.

Steam extraction to the power block

Having reached the nominal operation pressure of 70 bar, saturated steam is extracted from the buffer tank to keep the pressure constant. The automatic feed water control is activated although its output does not yet reach the value provided by the automatic cooling of the inlet water.

Switching from global to local recirculation

As soon as the drainage line is pre-heated to a temperature of 250°C and the steam extraction from the buffer tank reaches 3.5 kg/s, the separator drainage is opened and water from the drainage line is sent to the buffer tank. The inlet of the superheating section falls dry and the field outlet temperature rises. At a steam temperature of 350°C , the connection from the field outlet to the buffer tank is closed and the steam is directly sent to the power block. For the operation of the drainage system, it is now important to maintain the pressure in the buffer tank constant. Parallel to the steam extraction valve, a steam injection is included to prevent the pressure from falling below the limit.

End of the start-up procedure

When the steam temperature reaches 400°C the start-up procedure is considered as completed and plant

control can switch to nominal operation mode. This point is reached 44 min after focusing the collectors.

5.3 Solar-only start-up

If hot water for pre-heating the field is not available in the morning, the field has to be started in solar-only mode. Figure 5 shows the simulation results. In addition, no auxiliary boiler is used in this configuration. The start-up time from focusing the collectors is 55 min that means 11 min longer than with pre-heating. On the first glance, this seems to be not much. From the pressure increase it can be seen that the initial time lag at 7 bar pressure is 20 min which reduces to a value of 13 min during the following process. Since the process in solar only mode takes place later, the solar input is already on a higher level and more heat is provided into the system. Having reached the 70 bar, it takes 45 s in the solar-only mode until the steam production meets 3.5 kg/s and the switching to local recirculation is started. In contrast to that in the pre-heating version, the same procedure takes 252 s, since heat input into the system is significantly less.

5.4 Start-up at low irradiation

A reduced level of solar irradiation can e.g. be caused by haze in the atmosphere. To estimate the impact on the start-up procedure simulations are performed with a modified parameter in the solar irradiation model that represents a turbid instead of a clear sky atmosphere. From figure 6 it can be seen that the start-up time is increased by 12 min. The procedure itself stays similar to the original configuration.

6 Conclusions

The developed Modelica simulation model has proven its capability to simulate complex fluid-dynamic processes like the ones taking place during the start-up of a parabolic trough solar power plant. The interdisciplinary approach of the Modelica language is used to combine the thermo-hydraulic com-

ponents with a control system. Simulation runs show that a start-up time of about 55 min is necessary to reach nominal operation conditions. The long duration is mainly attributed to the low solar input after sunrise which results from the shading between two parallel collector rows. The start-up can be accelerated by 13 min when pre-heating the field with water stored in the feed water tank and in the buffer tank from the last evenings shut-down. The developed control concept is stable even if a reduce level of solar irradiation is assumed. Based on the cost estimate given in the introduction, the reduction by 13 min would lead to 2% lower electricity costs. Although this number is not high further potential is expected by improving the start-up strategy.

Acknowledgements

The authors would like to thank the German Ministry for the Environment, Nature Conservation and Nuclear Safety for the financial support given to the SOLDI project under contract No. 16UM0024.

References

- [1] Eck M., Zarza E., Eickhoff M., Rheinländer J., Valenzuela L. Applied research concerning the direct steam generation in parabolic troughs. *Solar Energy*, Vol. 74, 2003, pp. 341-351
- [2] Hirsch T., Eck M., Steinmann W.-D. Simulation of transient two-phase flow in parabolic trough collectors using Modelica. In: *Proceedings of the 4th Int. Modelica Conference*, Hamburg, Germany, 7.-8. March 2005, pp. 403-412.
- [3] Zarza E., González L., Rojas M.E., Caballero J., Rueda F.: Conceptual design of a 5 MWe Direct Steam Generation System. In: *Proceedings of the 12th Solar Paces International Symposium*, Oaxaca, Mexico, 2004.

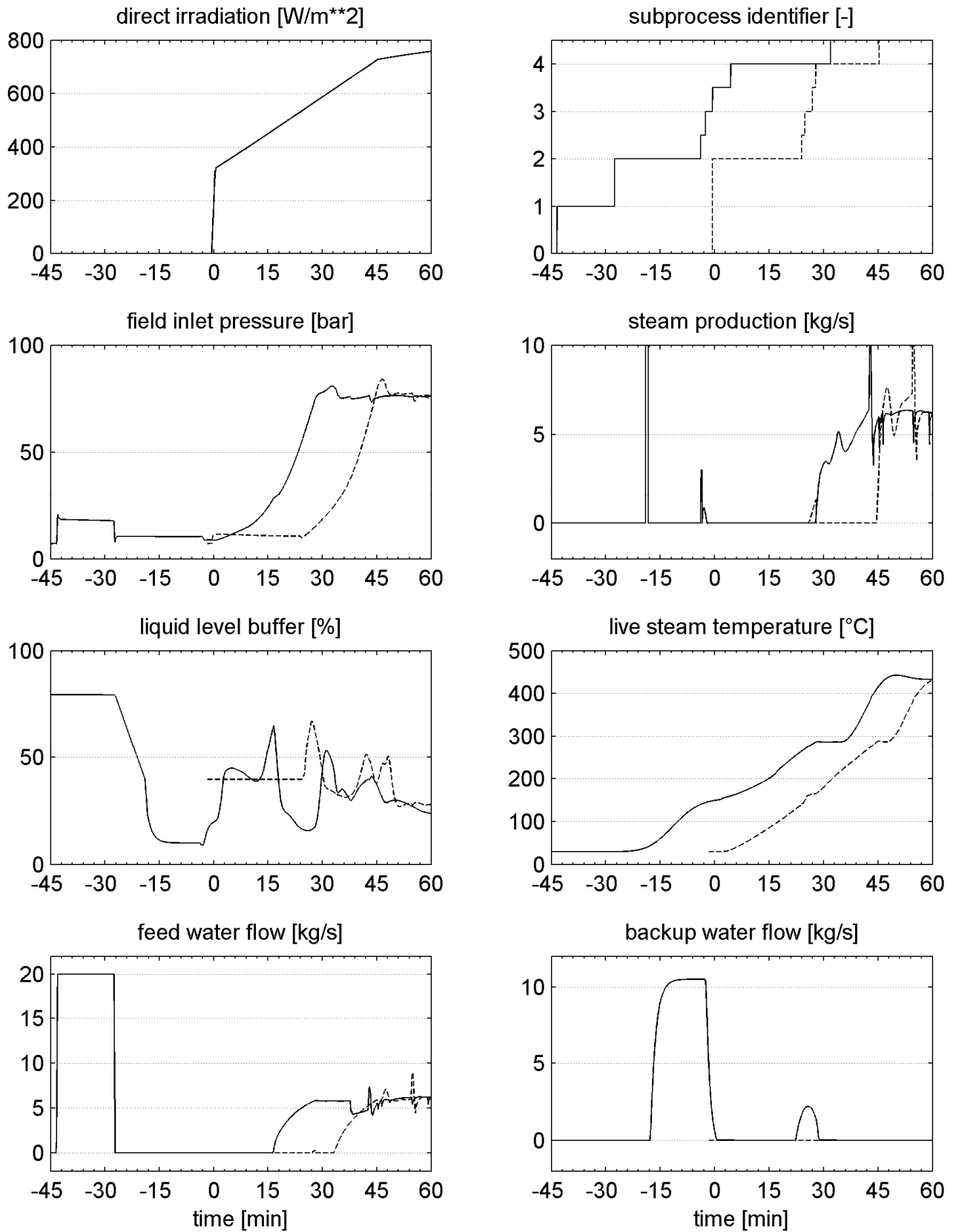


Figure 5: Start-up procedure with pre-heating of the field (---) and solar only start-up procedure (- -)

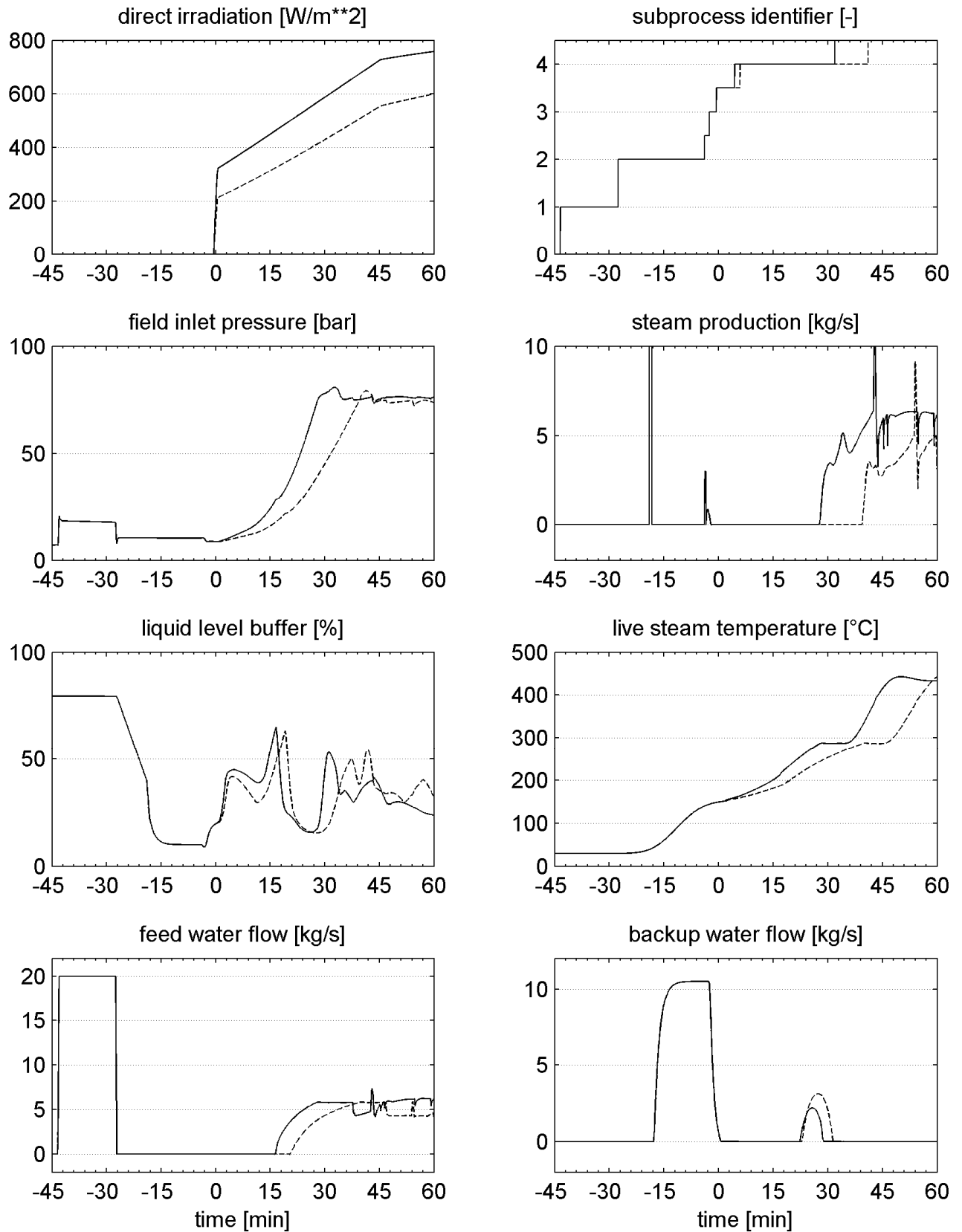


Figure 6: Start-up procedure with clear (---) and turbid (- -) atmosphere.

Session 2b

Automotive Applications 2

Modeling the Dynamics of Vehicle Fuel Systems

John J. Batteh Patrick J. Kenny
 Ford Motor Company, Research and Advanced Engineering
 {jbatteh, pkenny}@ford.com

Abstract

This paper describes the development and application of a multi-domain, physical model in Mod- elica for the simulation of vehicle fuel systems. The fuel system model includes components from the electrical, mechanical, and hydraulic domains to represent the physical components in the vehicle fuel system. A brief overview of the modeling back- ground and formulation is provided. Following a discussion of the model calibration and refinement effort, sample simulations are shown with the full system model for various transient tests. Additional applications and usage scenarios of the fuel system model are briefly discussed.

Keywords: hydraulics; mechanics; powertrain

1 Introduction

Drivability, emissions, and fuel economy, particularly during transient conditions, are the main drivers for the design requirements cascaded to vehicle subsystems. As gasoline prices become more volatile, vehicle fuel economy has become an increasingly important customer attribute. Fuel economy contributes strongly to customer satisfaction and perceived quality relative to the competition. Achieving fuel economy targets while also meeting increasingly-stringent emissions regulations [1] poses a significant engineering challenge to auto manufacturers. While aggregate fuel economy is a function of many factors, such as engine fuel consumption, vehicle weight, and the fuel control strategy, the basic components of the vehicle fuel system are fundamental pieces in the overall fuel economy picture.

In an effort to improve vehicle fuel economy, there is an increased focus on the design and behavior of the vehicle fuel system components. While the overall design of the fuel system must meet certain steady-state requirements for fueling capacity, *etc.*, transient operation is key for acceptable fueling system dynamic performance, emissions, and fuel econ-

omy. In particular, the majority of real world driving is transient as are the drive cycles on which fuel economy and emissions are measured.

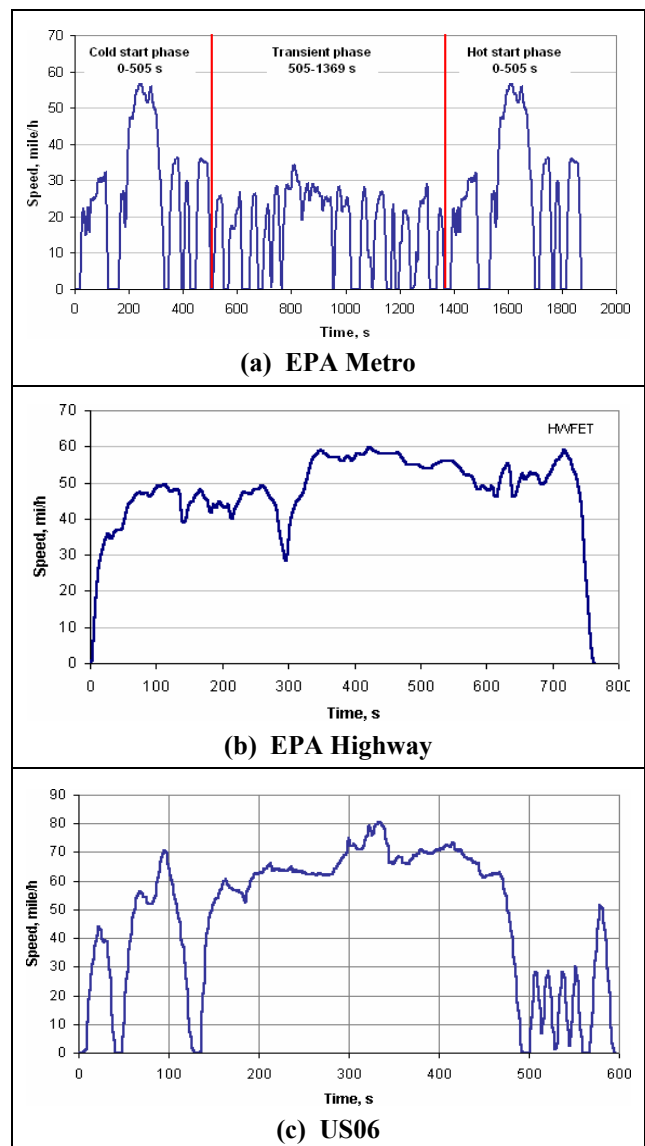


Figure 1. Common US drive cycles

Figure 1 shows the highly transient nature of three common US drive cycles: EPA Metro, EPA Highway, and US06. The EPA tests are used for emissions certification while the US06 is used to

represent a more aggressive driving style with higher speeds. Similar transient characteristics are found in the regulatory New European Drive Cycle (NEDC) and Japanese drive cycles. The speed traces for these and other drive cycles can be obtained from [2].

While customer and regulatory drive cycles point to the importance of transient performance, balancing steady-state and transient design considerations at the fuel system level is a nontrivial task. Cascaded requirements from the system and subsystem level lead to the design of the individual components, such as motors, pumps, and valves, but often the only mechanism to test the transient response of the system is late in the product development cycle when prototype hardware is available. These physical prototypes are expensive and may have long lead times to produce. Information obtained from physical testing may be obtained too late in the design cycle to allow an opportunity for design changes or iterative improvement. Rather than rely on the testing of physical prototypes, it is clearly desirable to develop multi-domain physical models to simulate the transient response of the system upstream in the design process where design changes are most easily accommodated.

This paper describes the development and application of a multi-domain physical model for a vehicle fuel system. Following some background information on this modeling effort, an overview of the physical components from the electrical, mechanical, and hydraulic domains is given. The processes for calibrating the component models from bench data and some model sensitivity results are shown. Furthermore, application and validation of the calibrated model are described including some model improvements to better match the dynamic response of the experimental data. Following the simulation results, some potential usage scenarios of the fuel system model are presented.

2 Fuel System Modeling

Modeling the dynamics of a vehicle fuel system requires physical models that span multiple domains. Figure 2 shows a generic schematic of a vehicle fuel system [3]. The fundamental components of the system are the fuel tank, fuel motor and pump, fuel lines, injector, various orifices such as a fuel filter, and associated regulation valves. A basic, lumped representation of these components requires elements in the mechanical, electrical, and hydraulic domains. This section provides some background

information regarding this modeling effort and gives an overview of the component models that comprise a typical vehicle fuel system. The background information provides additional anecdotal evidence of the benefits of component-based physical modeling.

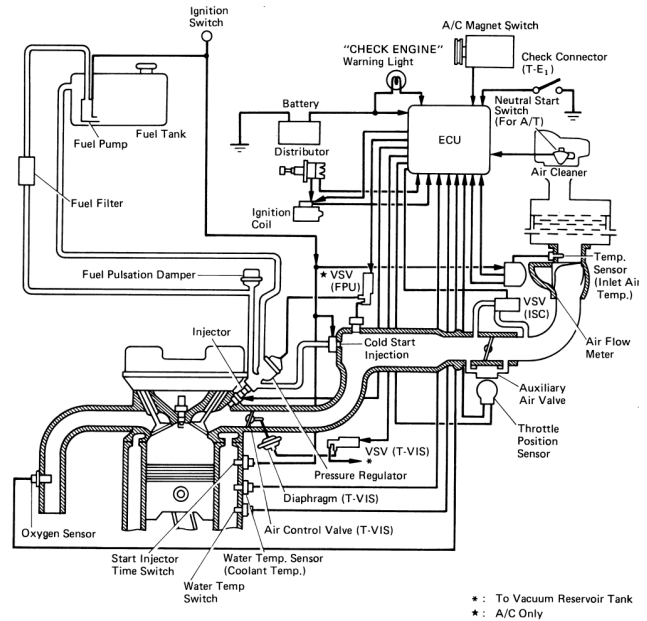


Figure 2. Schematic of a vehicle fuel system [3]

2.1 Background

At its most basic level, modeling involves the mathematical description of the physical behavior. However, great care must be taken in the abstraction of the physical system into its mathematic representation as relevant physical insight can be lost. While Modelica has the advantage of component-based physical models, other approaches that require direct generation of the underlying mathematical equations for use in equation-based solvers can be problematic. To underscore this point, an example relevant to fuel system modeling is presented.

The in-tank fuel pump, shown in the upper left hand side of Figure 2, is a fundamental component of the vehicle fuel system. The fuel pump is typically an integrated motor and pump assembly. The assembly is often characterized by steady-state data consisting of the voltage input to the motor, current in the motor, pressure difference across the fuel pump, pump speed, and pump flow rate. A common approach when using steady-state data to formulate a transient model is to establish a regression equation with unknown coefficients, fit the coefficients with the data, and then establish a semi-empirical transient form of the regression equation using applicable conservation laws. This technique when applied to the motor yields a semi-empirical formulation for

Kirchhoff's current law (with some non-standard units):

$$L \frac{dI}{dt} = V - a_o - a_1 I - a_2 RPM \quad (1)$$

where L is the motor inductance, V is the applied voltage to the motor, I is the current in the motor, and RPM is the motor shaft speed. A semi-empirical conservation of angular momentum for the motor shaft yields the following equation:

$$J \frac{dRPM}{dt} = c_o + c_1 I + c_2 RPM + c_3 RPM^2 + c_4 \frac{VI}{RPM} - c_5 \frac{QdP}{RPM} \quad (2)$$

where J is the shaft inertia, Q is the volumetric flow through the pump, and dP is the pressure difference across the pump. Those familiar with the modeling of DC motors will recognize some of the terms in the two equations above: the c_1 term represents the torque input from the motor, the c_5 term is the work done on the fluid, and the c_2 , c_3 , and c_4 terms are meant to account for losses in the system. While it is certainly possible to generate values for the a and c coefficients from the pump steady-state data, this approach has severe flaws and can result in unstable behavior. For example, this approach can result in inconsistent formulations since the power supplied by the electrical system is related to the torque input to the shaft (*i.e.* a_2 and c_1 are not independent). Furthermore, losses in the motor represented by the c_4 term should be related to the voltage drop across the motor (*i.e.* the a_2 term) not the voltage input to the system. Using the steady-state regression coefficients based on the formulation above, the transient equations implemented in SIMULINK [4] proved unstable for obvious reasons. Unfortunately, these types of modeling inconsistencies are extremely easy to overlook when formulating the underlying system conservation equations directly rather than modeling the physical behavior of the individual components.

Given the issues with generating a consistent, stable model for the entire system, a more fundamental, component-based approach was started using Modelica. This approach, while still using regressions to describe some physical behavior, ultimately proved more robust by sharply focusing the modeling on the physical behavior of the individual components. An overview of the Modelica models is given in the sections that follow.

2.2 Fuel Pump Assembly

The model for the fuel pump assembly is shown in Figure 3. It consists of a model of the electric mo-

tor with components from Modelica Standard Electrical library and the fuel pump connected by a shaft. The dynamics of this model are similar to those given by Eqs. (1-2) but with a consistent formulation. The regression coefficients generated for Eq. (1) above were used to specify the parameters for the electrical system.

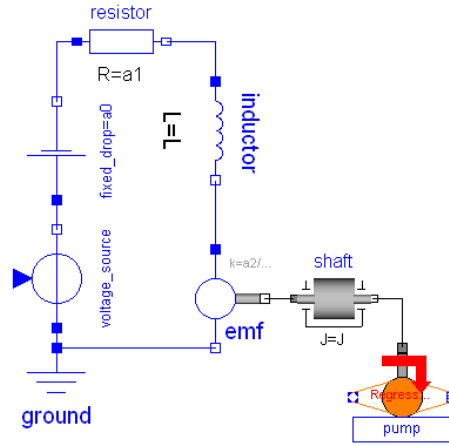


Figure 3. Modelica model of fuel pump assembly

A simple, efficiency-based formulation is used for modeling the pump to account for the various losses such as friction, leakage, *etc.*:

$$\eta_{pump} = \frac{\tau_{fluid}}{\tau_{shaft}} \quad (3)$$

where τ_{shaft} is the pump shaft torque and τ_{fluid} is the torque imparted to the fluid. At steady-state, the shaft torque is equal to the motor torque. Substituting the definitions for the shaft torque and the torque imparted on the fluid yields the following equation for the steady-state efficiency:

$$\eta_{pump} = \frac{QdP}{V_m I} \quad (4)$$

where V_m is the steady-state voltage drop across the emf device and can be calculated from the electrical system model by applying the voltage and shaft speed from the steady-state data. In keeping with the component-based approach, the pump efficiency is regressed from the steady-state data based on the pump operating conditions. A sample functional form for the efficiency equation is as follows:

$$\eta_{pump} = d_o + d_1 dP^2 + d_2 Q + d_3 Q^2 + d_4 RPM^2 \quad (5)$$

Note that this equation is semi-empirical and could be posed with different or additional terms based on the pump operating conditions and available data.

The results from a regression to the pump data are shown in Figure 4 and agree quite well with the experimental data. The R^2 value of the regression

can be increased semi-arbitrarily by including additional terms in the regression equation. The flowrate through the pump was also regressed from the steady-state experimental data using the following functional form:

$$Q = b_o + b_1RPM + b_2dP \tag{6}$$

The sample pump efficiency map in Figure 5 shows that the efficiency is stable (*i.e.* between 0 and 100%) along the pump operating line shown in blue. Note that while negative efficiencies from Eq. (5) are shown in Figure 5, these regions do not intersect the pump operating line; thus, the pump would never encounter negative efficiencies during simulations. The stability of this formulation was confirmed over the entire pump operating range (*i.e.* RPM and dP).

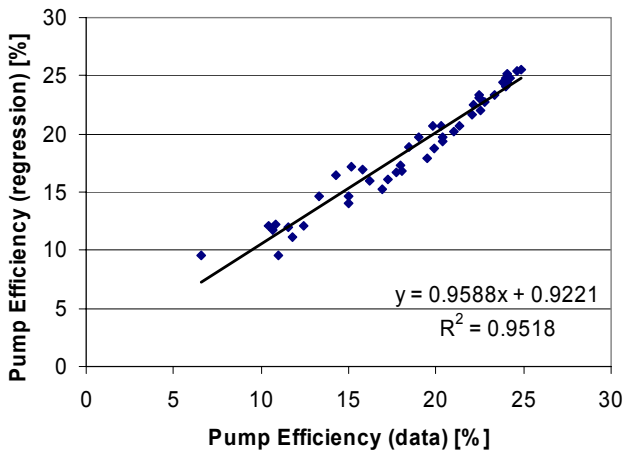


Figure 4. Sample regression for pump efficiency

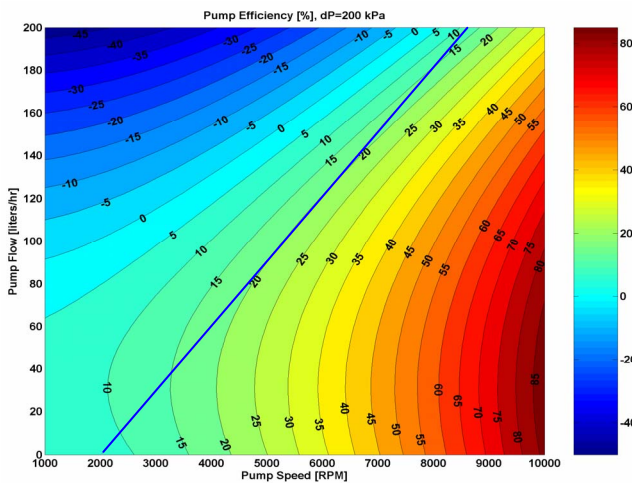


Figure 5. Sample pump efficiency map

2.3 Hydraulics

A BasicHydraulics library was developed consisting of accumulators, orifices, pumps, reservoirs, and various valves and flow devices for use in

hydraulics modeling. This library is targeted for casual Modelica users and thus intentionally did not employ advanced Modelica concepts such as the MediumModel. It employs a standard hydraulic formulation similar to the Modelica HyLib library [5] but is targeted at novice users. Given previous work in hydraulics modeling [6], it should be possible to develop a library of this scope from the forthcoming Modelica Fluid library [7]. A few of the more interesting components in the vehicle fuel system will be highlighted.

Figure 6 shows an excerpt of the code from the accumulator model including the effects of aeration in the liquid. In lieu of the MediumModel a parameter record is used for the fluid properties. The model includes the standard conservation equation:

$$\frac{V}{\beta} \frac{dP}{dt} = \sum Q \tag{7}$$

where the effective β is calculated from the liquid compressibility and the mole fraction of gas in the liquid, y_{gas} . Adjustments to these parameters can also be used to account for flexibility in the lines and affect the overall system stiffness as will be shown in subsequent sections.

```

parameter Modelica.SIunits.Volume volume=1e-6 "Volume";
parameter Modelica.SIunits.Pressure initP=101325 "initial pressure";
Modelica.SIunits.Pressure P(stateSelect=StateSelect.prefer)
"Pressure of fluid in accumulator";
parameter Interfaces.FluidProperties fluid_props "fluid property record";
protected
parameter Modelica.SIunits.BulkModulus beta_liq=fluid_props.beta
"fluid bulk modulus";
parameter Modelica.SIunits.VolumeFraction y_gas=fluid_props.y_gas
"volume fraction of gaseous component";
Modelica.SIunits.BulkModulus beta "effective fluid bulk modulus";
initial equation
P = initP;
equation
P = a.p;
l = beta/beta_liq + beta*y_gas/(1 - y_gas)*l/P;
volume/beta*der(P) = a.Q;

```

Figure 6. Code excerpt from the accumulator model

The vehicle fuel system commonly contains valves used for pressure regulation. Figure 7 shows the model used for the valves. This model combines elements from the hydraulic and mechanical domains and consists of a pintle mass between two stops. The pintle experiences a force from the hydraulic pressure and opposing preload and spring forces. The dynamic pintle position is used for the variable flow area calculation in the orifice. The mechanical components used in the valve are from the Modelica Translational library.

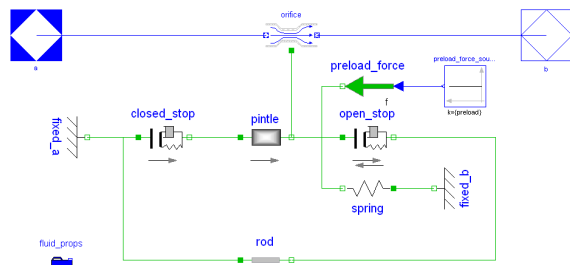


Figure 7. Valve model

A simple model is used for the vehicle fuel injector. The model, shown in Figure 8, contains a table similar to that in the vehicle control system. Given the commanded pulse width and pressure difference across the injector, the fuel injection mass is calculated and converted into a volumetric flowrate based on the engine speed. Consequently, the dynamic pressure upstream of the injector is extremely important as it has a direct impact on the quantity of fuel injected.

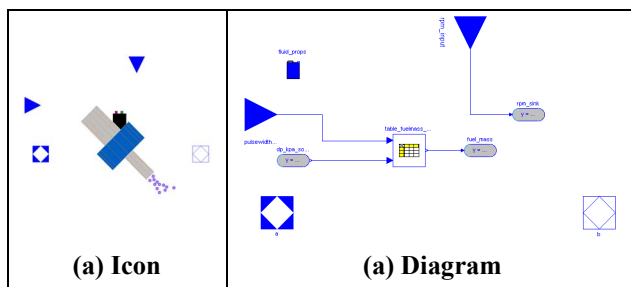


Figure 8. Injector model

2.4 Vehicle Fuel System

Figure 9 shows the vehicle fuel system model. The system model contains the fuel pump assembly shown in Figure 3 in conjunction with hydraulic components for the various lumped volumes in the system, fuel tank, fuel filter, injector, and valves. The single injector model accounts for the cycle-averaged fueling to the entire engine rather than individual pulses for the injector in each cylinder. This formulation is consistent with the overall model formulation and number of lumped volumes considered; the pulsed flow formulation might be required in other applications requiring higher fidelity representations of pressure pulsations in the system (*i.e.* fuel rail dynamics).

The primary inputs to the model are those provided by the control system, namely the voltage input to the electrical system, the engine speed, and the desired amount of fuel to be injected as determined by the fuel pulse width from the engine controller.

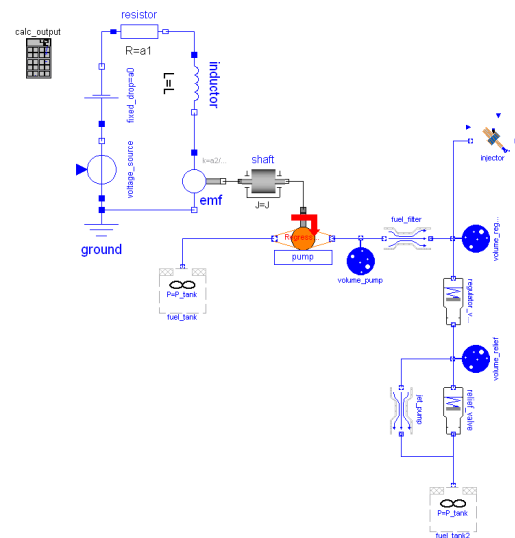


Figure 9. Vehicle fuel system model

3 Results

Prior to the simulation of the fuel system model in Figure 9, the relevant physical system parameters must be provided. These parameters include volumes of the various accumulators; flow areas and discharge coefficients of the various orifices; pintle mass, maximum travel, preload force, and spring constant in the various valves; and bulk properties of the fuel including fuel density and an estimate of the mole fraction of air dissolved in the fuel. While many of these parameters can be obtained from detailed system specifications, some of the parameters require calibration from experimental data. The following section gives an overview of the calibration process for a few selected component models. The simulation results in the following sections were obtained using Dymola [8].

3.1 Model Calibration

Another area where component-based modeling has a distinct advantage over equation-based system models is in model calibration and validation. Efforts to calibrate model behavior at the system level typically prove frustrating and often lead to unphysical calibrations due to multiple calibration knobs and dynamic interactions between components. Component-based physical modeling is ideally suited for calibration on the component or sub-model since unique, isolated test models can be easily constructed to replicate experimental bench tests.

Due to the timing of the vehicle fuel system model development effort, the only data that was available for model calibration resulted from a bench test of the entire fuel system. Careful extraction of

the relevant experimental data and construction of unique test models allowed both the steady-state and transient calibration of vital fuel system components.

While the regression provided in Eq. (6) provides good overall agreement with the pump data provided, there can be small differences that arise in steady-state values due to point-by-point regression accuracy and prototype hardware variation. To dial in the steady-state pump flow calculation at a given operating condition, the test model in Figure 10 was constructed. This model prescribes the pump speed and pressure difference from experimental data and allows for precise calibration of a flow multiplier to match the steady-state experimental data. A small adjustment in the flowrate gives the results shown Figure 11 where good steady-state agreement is achieved on each side of the transient test. It should be emphasized that this test was used to calibrate steady-state behavior, and thus the transient differences between the model and the data should be ignored.

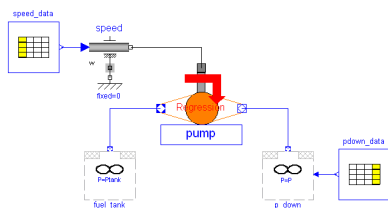


Figure 10. Pump flow calibration model

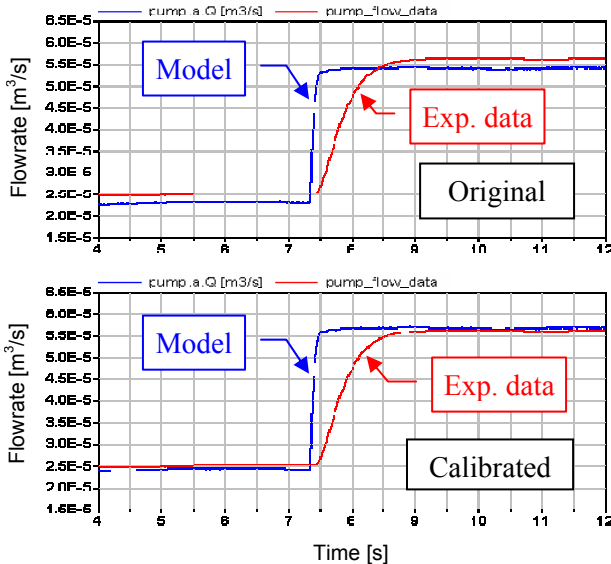


Figure 11. Pump flow calibration (steady-state) results

Figure 12 shows a model used for steady-state calibration of the flow in a valve as shown in Figure 7. In this case, the valve flowrate is not directly measured so the flow characteristics of the valve are calibrated such that the steady-state pressure up-

stream of the valve matches the experimental data given the flowrate upstream of the volume and the pressure downstream of the valve. Again, slight tweaks to the valve parameters, namely the pre-load force and the flow area, yield significant improvements in the steady-state behavior shown in Figure 13.

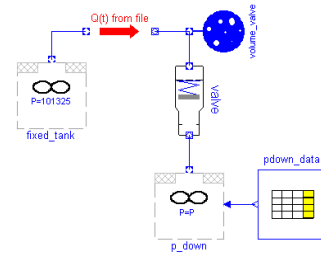


Figure 12. Valve calibration model

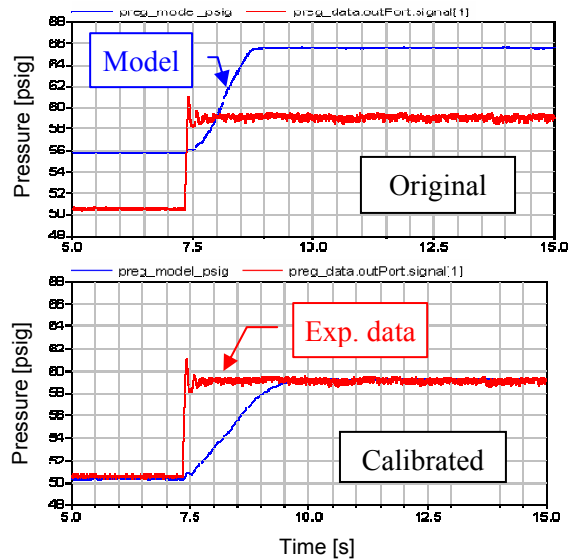


Figure 13. Valve flow calibration (steady-state) results

3.2 Model Sensitivity

Given the initial calibration of the components, the system in Figure 9 was simulated to understand the fuel system model sensitivities to estimated parameters. One key parameter in hydraulics modeling is the effective compressibility β in Eq. (7). This parameter accounts for the compressibility of the fluid, lines, and gas dissolved in the liquid. Figure 14 shows the effect of changing the system compressibility via the amount of air dissolved in the fuel. Decreasing the amount of air dissolved in the fuel results in a faster pressure response in the volume. It is not surprising that this value, which is difficult to estimate and often used as a model calibration factor, has a profound impact on the system stiffness as shown in the rate of pressure rise upstream of the injector.

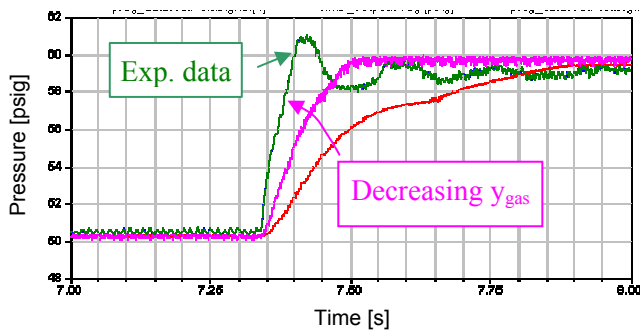


Figure 14. Sensitivity to effective compressibility

Figure 15 shows the effect of changing the size of the lumped volume upstream of the injector. As expected, reducing the size of the volume also increases the rate of pressure rise but, in this simulation, does not affect the response as significantly as changing the amount of dissolved gas in the liquid. It should be noted that system volumes are not typically considered adjustable parameters as the values are available from system/component design specifications. However, the sensitivity was explored here as there was considerable uncertainty in the differences between the experimental setup and the vehicle configuration from which the system data was obtained.

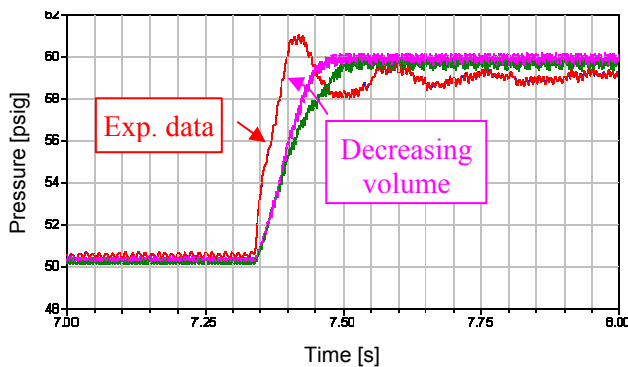


Figure 15. Sensitivity to volume size

3.3 Revised Model and Simulations

Following the sub-model calibration adjustments, the calibrated model was used in transient simulations. Figure 16 shows an overall comparison between the experimental data and the simulation for the pressure upstream of the injector. While the model captures the steady-state behavior reasonably well, there are certainly some differences in the transient details, namely the rate of pressure rise and the oscillatory response following the transient.

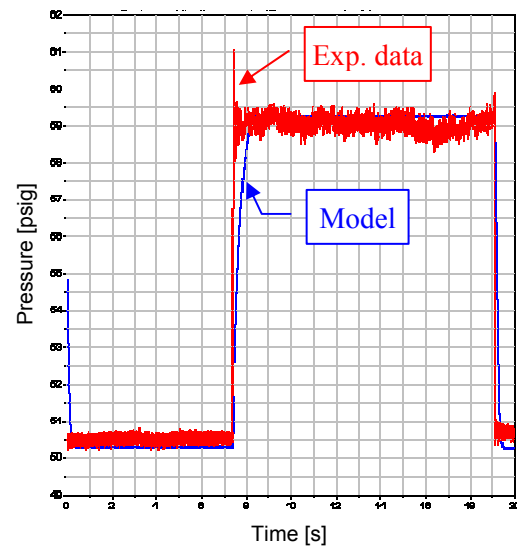


Figure 16. Transient simulations

Given the results from the model sensitivity study and these initial transient tests, the vehicle fuel system model was modified slightly to more accurately represent the dynamic response exhibited in the experimental data. Figure 17 shows the modified model as compared to the original in Figure 9. Note the region highlighted in the box. The initial model formulation considered a single, lumped volume in front of the injector. Since this volume represents the portion of the fuel system from the pump to the injector, it was rather large. Thus, the simulated response represented a filtered value of the actual dynamic response. The revised model shown in Figure 17 replaces the single large volume with two smaller volumes and models the fluid inertia between the volumes to account for the underbody fuel lines. Figure 18 shows the improved model results. Though further improvement is possible, the revised model clearly does a better job of capturing the detailed dynamic response of the fuel system.

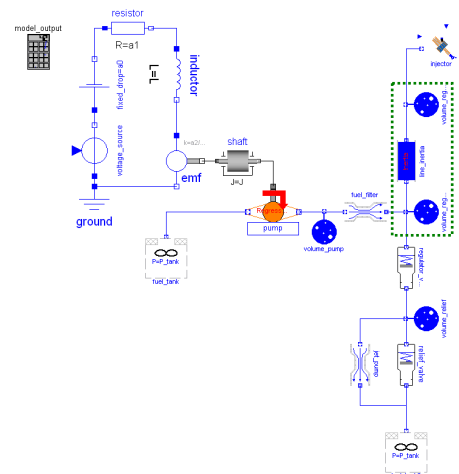


Figure 17. Fuel system model with inertia

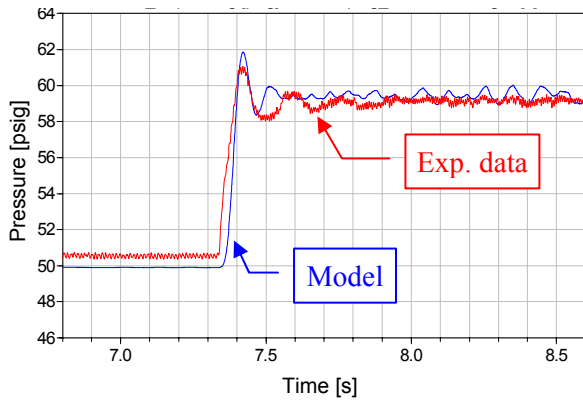


Figure 18. Improved model results

3.4 Drive Cycle Simulations

Having established that the model does a reasonable job of simulating the dynamic response of the vehicle fuel system subject to simple transient inputs, the model in Figure 17 was used to simulate drive cycle dynamics over a common US emissions cycle. Figure 19 shows the vehicle speed during the drive cycle (the first 1400s of the cycle shown in Figure 1a). The dynamic inputs to the fuel system model were provided by VPACS, a Ford-proprietary tool that models the vehicle and control system ([9],[10]). The simulations ran faster than real time though real time simulation was not a stated requirement for the system model.

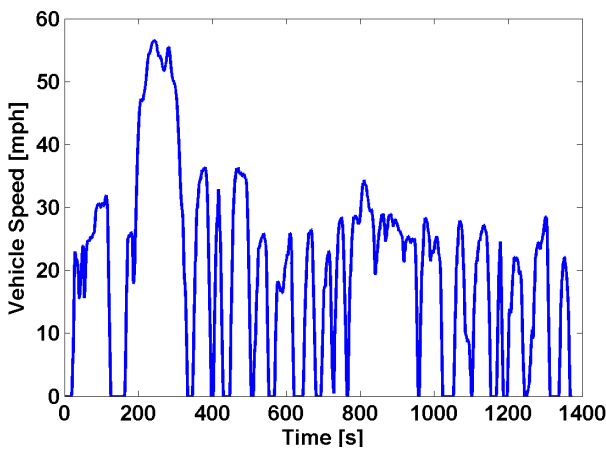


Figure 19. Vehicle speed during drive cycle

Figure 20 shows the pressure upstream of the injector and the engine fuel flowrate during the simulation. The dynamics are a result of the transient operating conditions (*i.e.* vehicle speed, engine operating conditions, desired fueling from the engine controller, *etc.*) combined with the dynamics of the fuel system. In these sample simulations, the pressure upstream of the injector is nominally maintained at a constant value, but some excursions can be seen. Depending on the implementation of the fuel control

system, these excursions can lead to differences between the desired and actual fueling to the engine and potentially degraded fuel economy, drivability, and emissions.

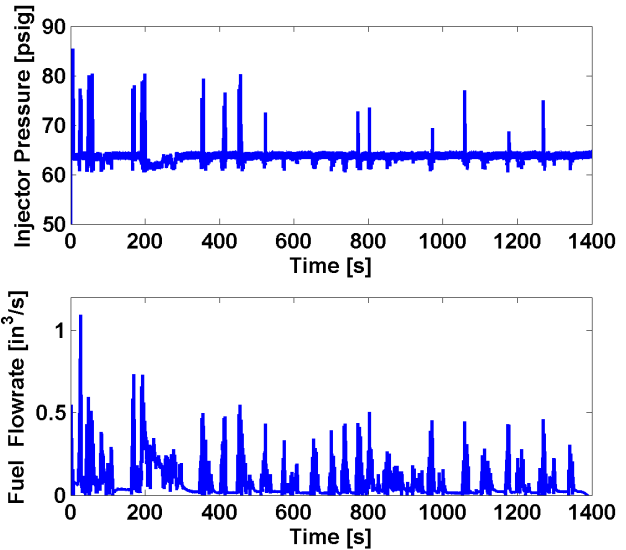


Figure 20. Drive cycle simulation results

3.5 Model Usage Scenarios

While this paper describes the initial model development for a multi-domain physical model of the vehicle fuel system, this model capability can be used in various applications. As an extension of the model sensitivity analysis described previously, the model can be used in conjunction with the NestedAnalysisToolkit [11] or the "Design Optimization" feature in Dymola [8] for structured system robustness analyses. In addition, the model can be combined with detailed models of the liquid fuel behavior ([12],[13]) and cycle simulation dynamics ([14]-[16]) in the engine to understand the impact of the injection dynamics on engine operation, drivability, and emissions. While the complex dynamics considered in such comprehensive engine models may not be suitable for simulation of entire drive cycles, the combined fuel system and engine model can be useful for in detailed analysis of selected transient events. In particular, the engine cranking and cold start behaviors which are extremely important for emissions compliance can be simulated while including the dynamics of the fuel system.

4 Conclusions

This paper details the formulation, development, calibration, and simulation of a multi-domain physical model for a vehicle fuel system. Simulation

of the calibrated model shows good agreement with experimental data. A sample simulation of the model over a vehicle drive cycle is included to illustrate possible application usage. Several potential applications of this model are discussed including integration with other detailed models of the power-train system. In addition to the modeling details, this work provides additional anecdotal evidence of the benefits of component-based physical modeling as compared to equation-based system modeling approaches.

Acknowledgements

The authors would like to thank Larry Buch for his upfront formulation work, system parameter identification, and ongoing support of this work, and Michael Tiller for his valuable discussions regarding the proper modeling of the fuel pump assembly.

References

- [1] US light-duty vehicle emissions regulations, <http://www.dieselnet.com/standards/us/light.html>
- [2] Emissions test cycle description/speed traces, <http://www.dieselnet.com/standards/cycles/>
- [3] <http://www.autozone.com>
- [4] Simulink. The Mathworks, Natick, MA, USA.
- [5] Beater, P., 2000, "Modeling and Digital Simulation of Hydraulic Systems in Design and Engineering Education using Modelica and HyLib", *Modelica Workshop 2000 Proceedings*, pp. 33-40, <http://www.modelica.org/events/workshop2000/proceedings/Beater.pdf>
- [6] Tiller, M., 2005, "Development of a Simplified Transmission Hydraulics Library based on Modelica.Fluid", *Proceedings of the 4th International Modelica Conference*, pp. 237-273, http://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3b4.pdf
- [7] 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
- [8] Dymola. Dynasim AB, Lund, Sweden, <http://www.dynasim.com>.
- [9] DeLosh, R.G., Brewer, K.J., Buch, L.H., Ferguson, T.F.W., and Tobler, W.E., 1981, "Dynamic Computer Simulation of a Vehicle with Electronic Engine Control", SAE-810447, Society of Automotive Engineers.
- [10] Strayer, B.A. and Trinker, F.H., 2005, "A Predictive Model for Feedgas Hydrocarbon Emissions: An Extension to Warm Engine Maps", SAE-2005-01-3862, Society of Automotive Engineers.
- [11] Batteh, J.J., Tiller, M. and Goodman, A., 2005, "Monte Carlo Simulations for Evaluating Engine NVH Robustness", *Proceedings of the 4th International Modelica Conference*, p. 385-392, http://www.modelica.org/events/Conference2005/online_proceedings/Session5/Session5a1.pdf
- [12] Puchalsky, C., Megli, T., Tiller, M., Trask, N., Wang, Y., and Curtis, E., 2002, "Modelica Applications for Camless Engine Model Development", *2nd International Modelica Conference Proceedings*, p. 77-86, http://www.modelica.org/events/Conference2002/papers/p11_Puchalsky.pdf
- [13] Batteh, J.J. and Curtis, E.W., 2003, "Modeling Transient Fuel Effects with Variable Cam Timing", SAE 2003-01-3126, Society of Automotive Engineers.
- [14] 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
- [15] 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
- [16] Bowles, P. and Batteh, J., 2003, "A Transient, Multi-Cylinder Engine Model Using Modelica", SAE-2003-01-3127, Society of Automotive Engineers.

Motorcycle Dynamics Library in Modelica

Filippo Donida, Gianni Ferretti, Sergio M. Savaresi, Francesco Schiavo, Mara Tanelli
 Politecnico di Milano, Italy
 Piazza Leonardo da Vinci 32, 20133 Milano

Abstract

This paper presents a Modelica library developed for the dynamic simulation of a motorcycle, developed within the Dymola environment (see [1], [2], [3]) and tailored to test and validation of active control systems for motorcycle dynamics. As a matter of fact, as a complete analytical model for two-wheeled vehicles is not directly available due to the complexity of their dynamic behavior, a reliable model should be based on multibody modeling tools endowed with automated symbolic manipulation capabilities. In this work we illustrate the modular approach to motorcycle modeling and discuss the tire-road interaction model, which is the crucial part of the simulator. Moreover, we propose a virtual driver model which allows to perform all possible maneuvers.

Keywords: Automotive Systems, Motorcycle Dynamics, Multibody Modeling, Dynamic Simulation.

1 Introduction and Motivation

In this work we present a library for the dynamic simulation of a two-wheeled vehicle developed in Modelica, within the Dymola environment. This library is tailored to be employed for test and validation of active control systems for motorcycle dynamics.

The design of a control oriented simulator for two-wheeled vehicles is a very challenging task, as a complete analytical model is not directly available due to its complexity and its high sensitivity to parameters' variations. Accordingly, a reliable model should be based on multibody modeling tools endowed with automated symbolic manipulation capabilities (see *e.g.*, [4], [5], [6]).

As a matter of fact, in two-wheeled vehicle one has to handle strong dynamic coupling between the rigid bodies (front and rear frames, front and rear wheels) and the elastic joints (fork and front and rear suspensions), which make it difficult to devise appropri-

ate reduced model for control purposes (see *e.g.*, [7], [8], [9]). The effort of analyzing well-defined driving conditions on a complete dynamic simulation model seems to be the key for a comprehensive control design for motorcycles. Such approach is well confirmed in the available literature (see *e.g.*, [7], [8], [9]).

In this work, we illustrate the modular approach to motorcycle modeling and present the library architecture discussing all the different packages. In particular, the core of the library lies in the tire-road interaction model, which manages the interaction between tires and road and has a major impact on both ride and handling properties of motorcycles (see *e.g.*, [10]).

Moreover, we propose a virtual driver model which allows to track a predefined trajectory and keep a target speed during different maneuvers.

The paper is organized as follows. Section 2 gives an overview of the overall library architecture, while Section 3 and Section 4 describe in detail the wheel-road interaction model and the chassis and suspension models, respectively. Section 5 discusses the employed aerodynamic model, whereas in Section 6 the road surface model is discussed. Section 7 introduces the implemented *Virtual Driver* model and its functionalities. Finally, in Section 8 some simulation results are presented to assess the validity of the proposed simulation model.

2 Motorcycle Dynamics Library Overview

The development of this library is based on the Modelica Multibody library. The `MotorcycleDynamics` is a library package for Dymola, developed in Modelica 2.2, which offers all the capabilities needed to perform virtual prototyping for a two-wheeled vehicle. The package shares basically the same structure of the `VehicleDynamics` library, see [11], and it is structured in a tree-based fashion. The root contains the standard motorcycle model and nine sub-packages

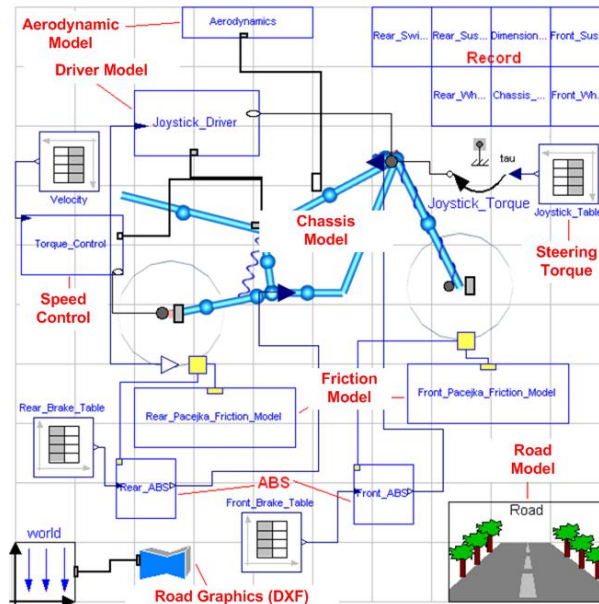


Figure 1: The complete simulation model block diagram.

- **Motorcycle:** is the eleven degrees of freedom motorcycle model with all the default components (e.g., wheels, suspensions, and so on). Figure 1 shows a the complete block diagram of the motorcycle;
- the package **Chassis:** it contains the chassis standard structure and the corresponding geometrical data. It is possible to define new Chassis models and to customize the existing ones;
- the package **Suspension:** it contains the basic spring-damper model, `Base_Suspension`, the front suspension (which also include the front fork and the handlebars) and the rear suspension;
- the package **Rear_Swinging_Arm:** it defines the rear swinging arm, which, again, is fully parametrized and customizable;
- the package **Wheel:** it contains all the wheel geometrical data, the Wheel Road Interaction model, the friction model, the tire relaxation model and the interface model between the wheel and the road surface. Specifically, two different friction models are available to the user. The first is based on a linear approximation of the friction forces and it is reliable at low slip, while the second relies on the Pacejka formula and it's highly nonlinear;
- the package **Braking_Systems:** it includes the braking system model and an Anti-lock Braking System ABS control;
- the package **Driver:** it offers different drivers models and capabilities which allow to perform all the different maneuvers;
- the package **Environments:** it models the road surface and handles also its animation rendering;
- the package **Aerodynamics:** it models the lift and drag aerodynamics forces;
- the package **Example:** it contains some examples which allow the user to explore the library capabilities.

In the following Sections we will describe in detail the structure of each package so as to highlight its peculiarities.

3 Wheel-Road Interaction

The core of the Motorcycle Dynamics library lies in tire modeling and in defining the interaction between tires and road, which has a major impact on both ride and handling properties of motorcycles (see e.g., [10]). In fact, the tire allows contact between the rigid part of the wheel (the hub) and the road surface, and it is the means for ensuring adherence to the road and for transferring to the ground longitudinal (i.e., traction and braking) and lateral friction forces, which guarantee steerability.

From the conceptual point of view, the `Wheel_road_Interaction` model (which is

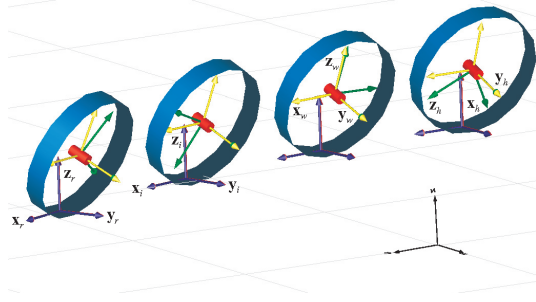


Figure 2: Wheel reference frames: hub (green) , wheel (yellow) , P (red) and POC (blue) .

part of the package `Wheel`) computes the forces arising at the contact between tire and road and the connection with the wheel holds *explicitly* through a mechanical connector. In order to compute these forces, it is fundamental to define the correct wheel reference frames. Namely, as it is shown in Figure 2, four reference frames are needed:

- `hub` : is the frame integral to the rigid part of the wheel, that is the hub reference frame;
- `wheel`: is the frame to describe the forward motion of the wheel;
- `P`: is the frame of the ideal contact point between wheel and ground, when no motion is applied to the wheel;
- `POC`: is the frame of the real contact point between wheel and ground, when the wheel is moving.

The said forces are in turn computed according to a description of the road, defined in the road model, which will be presented in Section 6. The connection between `Wheel_road_Interaction` model and `Road` model (which is part of the package `Environments`) is therefore performed *implicitly*, through the `inner/outer` statement, generally used to describe force fields. In this way, the description of the road can be made available to all wheels and vehicles in the scene.

The *hub* frame is attached to the wheel hub, it rotates with the wheel around the axis \mathbf{y}_h and it is the frame associated to the connector trough which the wheel is attached to its rotational joint. The *wheel* frame is attached to the wheel center, with unit vector \mathbf{y}_w coinciding with \mathbf{y}_h , while \mathbf{x}_w is the unit vector associated with the wheel velocity direction - obtained as the intersection between the plane orthogonal to \mathbf{y}_w and the

road tangent plane at the ideal contact point. The unit vector \mathbf{z}_w completes the frame

$$\mathbf{y}_w = \mathbf{y}_h \quad \mathbf{x}_w = \frac{\mathbf{y}_w \times \mathbf{n}}{|\mathbf{y}_w \times \mathbf{n}|} \quad \mathbf{z}_w = \mathbf{x}_w \times \mathbf{y}_w .$$

The *ideal* contact point frame has the origin laying on the road plane, along the direction identified by \mathbf{z}_w , the unit vector \mathbf{z}_i is the road normal unit vector \mathbf{n} , \mathbf{x}_i coincides with \mathbf{x}_w and \mathbf{y}_i simply completes the frame

$$\mathbf{z}_i = \mathbf{n} \quad \mathbf{x}_i = \mathbf{x}_w \quad \mathbf{y}_i = \mathbf{z}_i \times \mathbf{x}_i .$$

The location \mathbf{r}_i of the origin of the ideal contact frame is computed as:

$$\mathbf{r}_i = \mathbf{r}_h - \Delta_z \mathbf{z}_w ,$$

with \mathbf{r}_h being the location of the origin of the hub frame and with Δ_z being the distance between the hub origin and the road plane in the \mathbf{z}_w direction. The *ideal* and *real* contact frames are aligned with each other and differ only in the origin position; the location \mathbf{r}_r of the real contact frame is displaced from \mathbf{r}_i (see *e.g.*, [6], [12]) by the so-called tire trail. Hence

$$\mathbf{r}_r = \mathbf{r}_i + \mathbf{R}_r \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \end{bmatrix} ,$$

where \mathbf{R}_r is the rotation matrix from the real contact frame to the absolute frame. In turn, the displacements δ_x , δ_y and δ_z have been computed¹ as in [6]:

$$\begin{aligned} \delta_x &= f_w R \operatorname{sgn}(v_x), \\ \delta_y &= (\Delta_z - R) \sin(\varphi), \\ \delta_z &= (\Delta_z - R) \cos(\varphi), \end{aligned}$$

where R is the wheel radius, v_x is the forward wheel ground contact point velocity and f_w is the rolling friction coefficient [6], which can be computed as:

$$f_w = A + \frac{B}{p} + \frac{C}{p} v_x^2 ,$$

where p is the tire pressure and v_x the wheel forward velocity, *i.e.*,

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \mathbf{R}_r^T \mathbf{v}_r = \mathbf{R}_r^T \frac{d\mathbf{r}_r}{dt} ,$$

¹The $\operatorname{sgn}(x)$ function has been smoothed by replacement with the function $\operatorname{sgn}(x) \approx \frac{2}{\pi} \arctan(kx)$, with $k \approx 10^{10}$.

and A, B, C are suitable non-negative parameters.

The wheel roll angle φ is given by

$$\varphi = -\arctan \frac{\mathbf{z}_w^T \mathbf{y}_r}{\mathbf{z}_w^T \mathbf{z}_r}.$$

The forces arising from the tire-road interaction can be decomposed into a vertical force \mathbf{F}_z , a longitudinal force \mathbf{F}_x and a lateral force \mathbf{F}_y . The normal force \mathbf{F}_z can be computed as:

$$\mathbf{F}_z = - \left(K_{el} \delta_z + D_{el} \frac{d\delta_z}{dt} \right) \mathbf{z}_r$$

with K_{el} and D_{el} being the tire elastic and damping constants which describe the tire elasticity properties.

As for the longitudinal and lateral forces descrip-

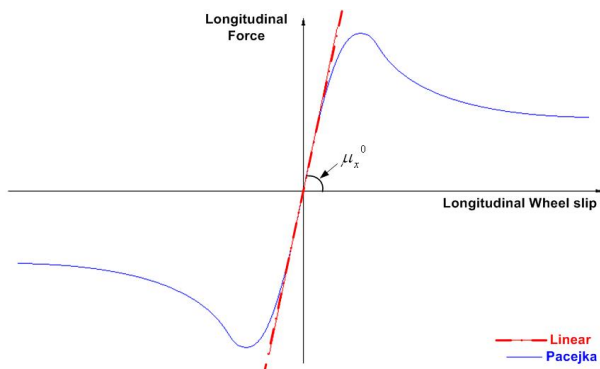


Figure 3: Plot of the longitudinal force \mathbf{F}_x as a function of the wheel slip computed with the nonlinear Pacejka formula (solid line) and its linear approximation (dashed-dotted line).

tion, two different models can be employed, according to the specific needs, namely a linear approximation model or a nonlinear model based on the Pacejka formula presented in [10].

The user can select either the linear or the Pacejka model, according to the current maneuver and to the analysis purpose of the simulator (see Figure 3 for a comparison between the longitudinal force \mathbf{F}_x as a function of the wheel slip computed with the nonlinear Pacejka formula and its linear approximation). The longitudinal force \mathbf{F}_x , is either a traction force \mathbf{F}_t - during acceleration - or a braking force \mathbf{F}_b . In the linear model, both traction and braking forces are computed as functions of the longitudinal wheel slip λ (see Figure 3), as

$$\mathbf{F}_x = \mu_x^0 \lambda \mathbf{x}_r, \quad (1)$$

where

$$\mu_x^0 = \left. \frac{\partial |\mathbf{F}_x|}{\partial \lambda} \right|_{\lambda=0}$$

and

$$\lambda = \left(R \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^T \mathbf{R}_r^T \boldsymbol{\omega}_r - v_x \right) / v_x.$$

Finally, the lateral force \mathbf{F}_y is computed as a linear function of the tire side-slip angle (see *e.g.*, [12]), *i.e.*, the angle between the rotational equivalent wheel velocity $\boldsymbol{\omega}R$ and the wheel-ground contact point velocity v_x and of the roll angle. Namely

$$\mathbf{F}_y = (k_\alpha \alpha - k_\varphi \varphi) \mathbf{F}_z = (K_\alpha \alpha - K_\varphi \varphi) \mathbf{y}_r, \quad (2)$$

where α is the tire side-slip angle and φ is the roll angle. In Equation (2),

$$K_\alpha := \left. \frac{\partial |\mathbf{F}_y|}{\partial \alpha} \right|_{\alpha=0, \varphi=0} \quad K_\varphi := \left. \frac{\partial |\mathbf{F}_y|}{\partial \varphi} \right|_{\alpha=0, \varphi=0},$$

are the cornering and camber stiffness, respectively.

In the Pacejka model, instead, both longitudinal and lateral forces are highly nonlinear functions, namely

$$\mathbf{F}_x = \mathbf{F}_x(\mathbf{F}_z, \lambda, \alpha, \varphi), \quad (3)$$

$$\mathbf{F}_y = \mathbf{F}_y(\mathbf{F}_z, \lambda, \alpha, \varphi). \quad (4)$$

The Pacejka formulas for \mathbf{F}_x and \mathbf{F}_y are based on a semi-empirical model, and the analytical expressions of the forces depend on a huge number of parameters estimated from data. This model has been implemented according to the results proposed in [10], where extensive tests on different tires have been carried out so as to estimate the needed parameters via numerical optimization.

Moreover, the wheel-road interaction model has to take into account the tire relaxation dynamics, which describes the tire deformation due to elasticity properties of the tire material. This phenomenon is modeled computing the real longitudinal and lateral forces as

$$\dot{\mathbf{F}}_{xreal} = \frac{1}{\tau(t)} (\mathbf{F}_x - \mathbf{F}_{xreal}) \quad (5)$$

$$\dot{\mathbf{F}}_{yreal} = \frac{1}{\tau(t)} (\mathbf{F}_y - \mathbf{F}_{yreal}), \quad (6)$$

where \mathbf{F}_x and \mathbf{F}_y are as in (1) and (2) if the linear force model is selected, or as in (3) if the Pacejka model is employed. The filter time-varying time constant is $\tau(t) = s_{0l}/v_x$, where s_{0l} is the tire-relaxation length, usually set equal to half of the tire circumference, and v_x is the wheel-ground contact point velocity.

Finally, these interaction forces have to be balanced by the forces \mathbf{F}_h (and torques $\boldsymbol{\tau}_h$) at the hub frame, thus

$$\mathbf{0} = \mathbf{F}_h + \mathbf{F}_{xreal} + \mathbf{F}_z + \mathbf{F}_{yreal}$$

$$\mathbf{0} = \boldsymbol{\tau}_h + (\mathbf{r}_r - \mathbf{r}_h) \times (\mathbf{F}_{xreal} + \mathbf{F}_z + \mathbf{F}_{yreal}).$$

4 Chassis and Suspensions Modeling

In order to develop a reliable simulator, the geometry of the motorbike has to be carefully defined. As a matter of fact, very small changes in the center of gravity position may substantially alter the dynamic vehicle behavior. Moreover, we designed the library to be highly reusable and customizable.

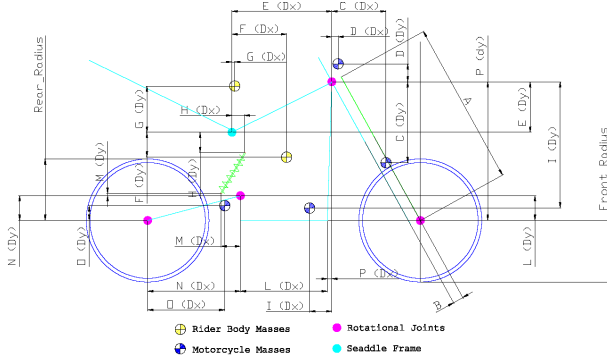


Figure 4: Motorcycle and driver geometry and center of gravity position.

To this end, each subpackage contains a record data structure (see also Figure 1) to store and define all geometric data, *e.g.*, masses, inertias and physical parameters for each component. For example, Figure 4 shows the chassis and driver geometric data which are stored in the `Base_Dimension_MassPosition_Data` record. The `Suspension` package contains the basic damper model `Base_Suspension`, the front suspension `Front_Suspension` (which comprises the front fork, the handlebars and their hard stops `Steer_Stopper`) and the rear suspension `Rear_Suspension`.

Both front and rear suspension employ the same mono-directional double spring-damper model

$$\begin{aligned} F_s &= k_s(x - x_0) \\ F_d &= c_d \dot{x} \\ F_{susp} &= -F_s - F_c - PreLoad, \end{aligned}$$

where

- k_s is the spring elastic constant and $(x - x_0)$ is the spring compression/deflection;
- F_s is the spring elastic force and it is a function of the spring compression/deflection;
- c_d is the damping coefficient;
- F_d is the damper force and it is a function of the compression/deflection velocity \dot{x} ;

- *PreLoad* is a constant parameter which depends on the spring tuning and allows to change the static load distribution of the motorbike.

5 Aerodynamics

Two are the main aerodynamic forces which need to be taken into account in vehicle modeling, and are shown in Figure 5

- the *Drag Force* which is directed along the longitudinal axis;
- the *Lift Force* which is directed along the vertical axis.

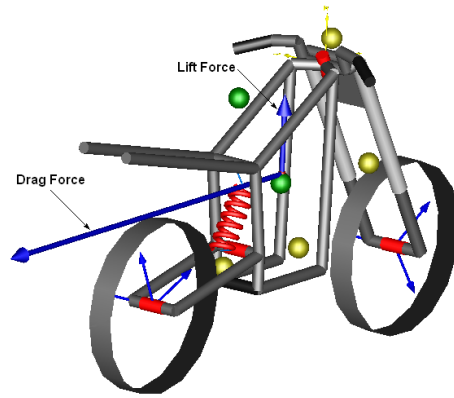


Figure 5: Lift and Drag Aerodynamic forces.

These forces are applied in a specified point called *pressure center* which is generally shifted forward with respect to the chassis center of gravity (see [6]). The drag force affects the maximum speed and acceleration values and it is proportional to the square of the forward speed. It is computed as

$$\mathbf{F}_d = \frac{1}{2} \rho C_x A v^2, \quad (7)$$

where ρ is the air density, C_x is the drag coefficient, A is the section of the motorbike area in the forward direction and v is the vehicle forward velocity.

The lift force (also proportional to the square of the forward speed) reduces the vertical load on the front and rear tire. It is computed as

$$\mathbf{F}_l = \frac{1}{2} \rho C_l A v^2, \quad (8)$$

where C_l is the lift coefficient and all the other parameters have the same meaning as in (7).

6 Road Surface Model and Rendering

The road model is part of the `Environments` package and it has been developed on the basis of the road model given in the `VehicleDynamics` library [11]. However, we extended that model by adding the z co-

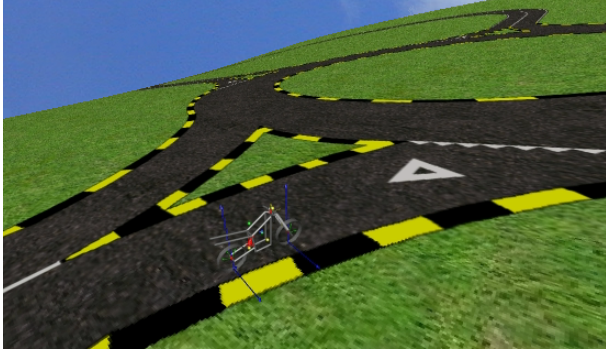


Figure 6: Screenshot of the motorbike and of a road model.

ordinate. Specifically, the z coordinate may vary as a function of the (x,y) position, unlike in the model available in the `Dymola` library, where the z coordinate can be set by the user but has to remain fixed for the whole simulation.

The road characteristics, accessible through the `Road` model, are the following:

- the vector \mathbf{n} , perpendicular to the surface tangent to the road at the tire-road contact point;
- the quote of the contact point z ;
- the friction coefficient μ , which allows to model different road conditions (*e.g.*, dry and wet asphalt, icy road and so on).

Hence, the road model is essentially defined by a surface of equation $f(x,y,z) = 0$, from which the relevant normal unit vector can be computed as $\mathbf{n} = \nabla f / |\nabla f|$, and by the road surface characteristics (described via the friction coefficient μ). The computation of the surface gradient has been automatically performed through the statement `partialderivative()`, described in [13].

As for the rendering, it is possible to visualize the road in the `Dymola` 3D animator via a `dxg` file created with an *ad hoc* Matlab routine and the open source software `QuikGrid`. Figure 6 and Figure 7 shows examples of road surfaces created with this procedure.

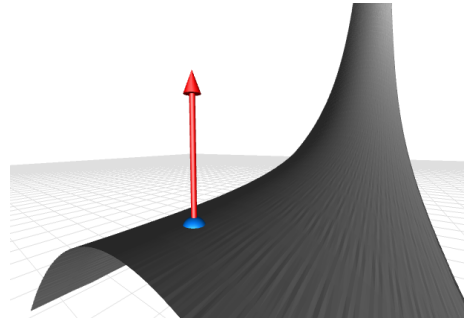


Figure 7: Example of a saddle-like road surface with the normal unit vector and height.

7 Virtual Driver

For the effective simulation of two-wheeled vehicles it is necessary to develop a virtual driver model. To this end, we complemented our library with a `Driver` package, which encloses all the control systems needed to perform a desired maneuver.

To this end, the motorcycle model is complemented with two control loops which take care, on one hand, of keeping a constant target speed throughout a specified maneuver and, on the other, of stabilizing the motorbike and following a predefined trajectory.

The closed loop speed control has been implemented as a Proportional-Integral control of the form

$$R(s) = \frac{k_{PS} + k_I}{s},$$

where the proportional and integral gain can be tuned so to model different drivers' behaviors. This controller takes as input the current forward vehicle velocity and outputs the appropriate traction or braking torque needed to follow the target speed set-point value.

The first task in order to model a virtual driver is to be able of following a desired roll angle profile, which allows to follow a road with turns. Accordingly, the proposed controller computes the steering torque to be applied as a function of the roll angle error, defined as

$$e_\varphi(t) = \varphi(t) - \varphi^o(t),$$

where the value of the set-point roll angle φ^o is computed, at any given time instant, as

$$\varphi^o(t) = \frac{v(t)^2 C(t)}{g},$$

where $v(t)$ is the vehicle forward velocity, $C(t)$ is the instantaneous curvature of the trajectory and g is the gravitational acceleration. Hence, the set-point

roll profile is obtained as the concatenation of the instantaneous roll equilibrium values, computed assuming steady-turning conditions [6]. The roll-angle controller has the form

$$\tau_s(t) = k_0\ddot{\varphi}(t) + k_1\dot{\varphi}(t) + k_2e_\varphi(t), \quad (9)$$

where τ_s is the steering torque to be applied in order to track the predefined roll. The presence of the first and second derivatives of the roll angle come from the observation that a real driver tunes his response not only based on current roll angle, but also according to the rolling velocity and acceleration.

If we consider, coherently with the steady turning assumption mentioned above, the set-point $\varphi^o(t)$ to be constant, then we can rewrite (9) as

$$\tau_s(t) = k_0\ddot{e}(t) + k_1\dot{e}(t) + k_2e(t),$$

so that the transfer function of the roll controller can be expressed in the more familiar form

$$R_\varphi(s) = \frac{k_0 + k_1s + k_2s^2}{(1 + sT_1)(1 + sT_2)}, \quad (10)$$

where, again, the time constants T_1 and T_2 can be tuned to obtain faster or slower tracking performance. Such roll controller is then complemented with two more terms, accounting for the direction and position errors, which have to be taken into account in order to track not only a roll profile, but also a desired trajectory, *e.g.*, to drive the bike on a race track. Accordingly, the final expression for the tracking controller (see also [14]) has the form

$$\tau_s(t) = k_0\ddot{\varphi}(t) + k_1\dot{\varphi}(t) + k_2e_\varphi(t) + k_3\Delta P(t) + k_4\Delta\Psi(t),$$

where $\Delta P(t)$ represents the position error, while $\Delta\Psi(t)$ the direction error. These are computed discretizing the desired trajectory and evaluating the discrepancy between the current position and the closest trajectory point. Such point, though, is chosen according to a *look-ahead* rationale, *i.e.*, as the closest point computed over an error prevision time interval. It is worth noting that this trick is actually performed by human riders, which tend to adjust the vehicle direction looking forward on the road and not based on the current vehicle position. In our controller, we inserted the trajectory tracking performance requirement by computing the overall desired roll profile considering also the position and direction errors $\Delta P(t)$ and $\Delta\Psi(t)$. Such new set-point $\varphi^o(t)$ is then fed as input to the controller (10).

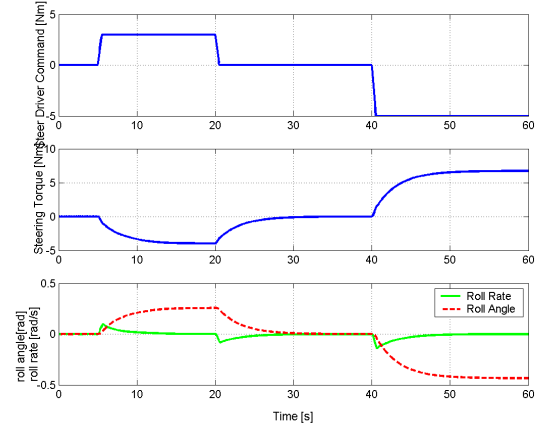


Figure 8: Steering torque input from the driver (top), steering torque applied by the stabilizer (middle), roll rate (bottom, solid) and roll angle (bottom, dashed).

The second possible driver model allows the simulator user to input the desired trajectory as through a joystick, that is to provide as input to the simulator a steering torque profile to be followed. According to the given profile, then, a control loop has been designed which stabilizes the motorbike to a steady state tilt angle with zero roll velocity. As far as the stabilizer design is concerned, in fact, the user input steering torque is treated as a measurable disturbance, and the stabilizer provides as output the steering torque needed to achieve a steady state behavior of the roll angle. The performance of such controller are shown in Figure 8, where the user input is a sequence of three steering torque steps. As it can be seen, the stabilizer provides as output the correct steering torque which allows the motorbike to follow the profile entered by the user. As a consequence, the roll angle and the roll rate experience a transient after each step input before stabilizing to a steady state value.

8 Simulation Results

We now show some simulation result which assess the validity of the `MotorcycleDynamics` library. First of all, to validate the trajectory tracking performance a simulation experiment is presented where the motorcycle is commanded to run a circular trajectory, with a radius of 12.5 m at a target speed of 8 m/s. The results obtained with the `Dymola 3D-Animator` are shown in Figure 9(a), where the solid line depicts the trajectory of the rear hub frame origin, while the simulated trajectory of front wheel, rear wheel and chassis are shown in Figure 9(b). Note in particular the large in-

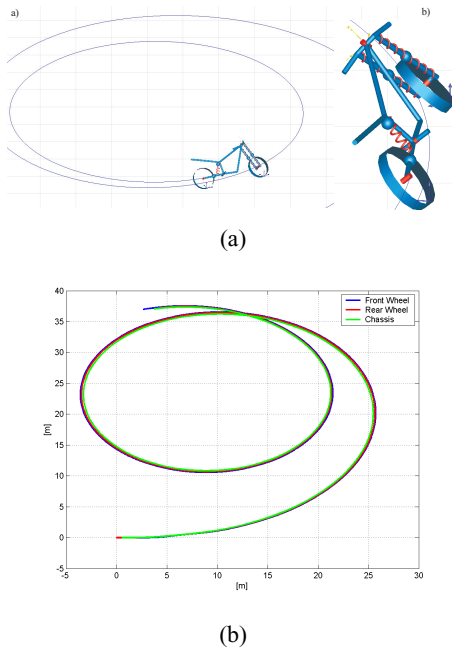


Figure 9: Simulated trajectory visualized via the Dymola 3D-Animator (top) and plot of the simulated trajectory of Front Wheel, Rear Wheel and Chassis (bottom).

clination of the motorcycle (Figure 9(a)b), needed to run a trajectory with such a high curvature. To better understand the virtual driver behavior, Figure 10(a) shows, via the 3D animator, the four phases during a curve. Namely, the curve is entered with a countersteering, which makes the vehicle tilt correctly, then there is a steering phase which takes to the steady state cornering. Finally, the curve is exited and the vehicle is back to zero tilt angle. Figure 10(b) shows, in the same maneuver a plot of the steering angle, steering torque and roll angle. We also show in Figure 11(a) a screenshot of the 3D animator when a dangerous braking maneuver (panic brake) on a curve is performed and the subsequent fall of the motorbike. Figure 11(b) shows the correct behavior of the simulated roll angle and front wheel sideslip angle which correctly diverge when the bike falls. Finally, we command the motorbike to follow an eight trajectory on a hilly road, modeled via a hyperbolic paraboloid of the form

$$z = \left(\frac{x}{20}\right)^2 + \left(\frac{y}{30}\right)^2 + 0.55. \quad (11)$$

Figure 12 shows the various phases of the maneuver and the simulated trajectory.

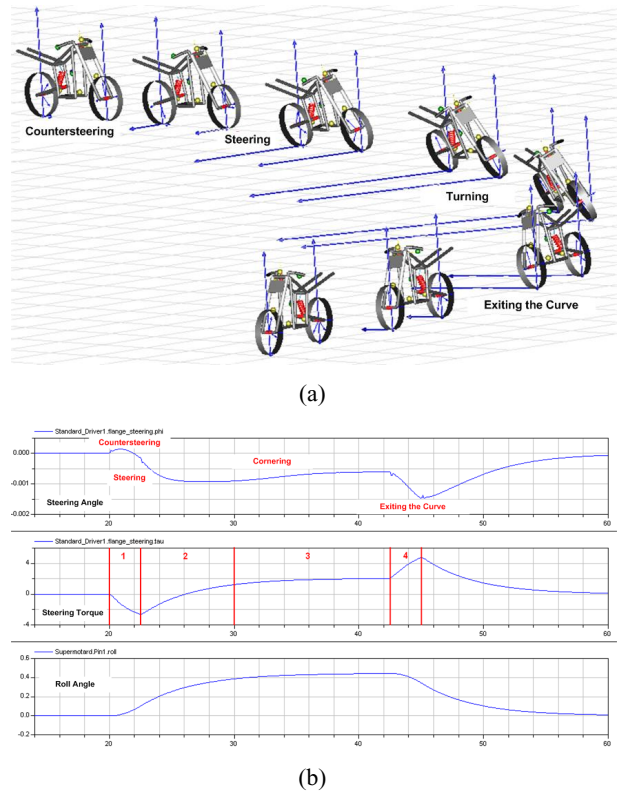


Figure 10: Screenshot of a turning maneuver (top) and plot of the steering angle, steering torque and roll angle in the same maneuver (bottom).

9 Concluding Remarks

This work presented the development of the `MotorcycleDynamics` library for Dymola, developed in Modelica 2.2, which offers all the capabilities needed to perform virtual prototyping for a two-wheeled vehicle. The overall architecture of the library has been thoroughly discussed, and its functionalities have been highlighted via simulation experiments.

A controller for target speed and trajectory tracking, based on a virtual driver model has been presented and simulation results assessed its validity.

Future work will be devoted to validate the motorcycle model with experimental data and to exploit the library capabilities for active control system prototyping.

Moreover, we plan to extend the driver model so to insert the driver lean angle as an additional degree of freedom in the motorcycle model, in order to model the driver as *active*, capturing his real behavior and its effects on the overall vehicle dynamics.

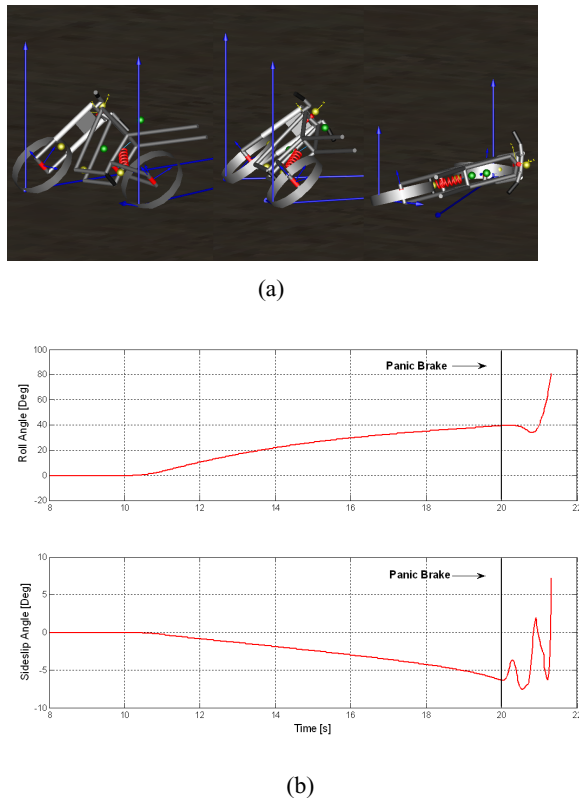


Figure 11: Animation of a motorcycle fall during a braking maneuver on a curve (top) and corresponding roll angle and front wheel sideslip angle (bottom).

References

- [1] M. Tiller, *Introduction to Physical Modeling with Modelica*. Kluwer, 2001.
- [2] M. Otter, H. Elmqvist, and S. Mattsson, “The new modelica multibody library,” in *Proc. 3rd International Modelica Conference*, Linköping, Sweden, November 3-4, 2003, pp. 311–330.
- [3] S. Mattsson, H. Elmqvist, and M. Otter, “Physical System Modeling with Modelica,” *Control Engineering Practice*, vol. 6, pp. 501–510, 1998.
- [4] R. Sharp, “The stability and control of motorcycles,” *Journal of Mechanical Engineering Science*, vol. 13, pp. 316–329, 1971.
- [5] V. Cossalter and R. Lot, “A motorcycle multibody model for real time simulations based on the natural coordinates approach,” *Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility*, vol. 37, pp. 423–447, 2002.
- [6] V. Cossalter, *Motorcycle Dynamics*. Milwaukee, USA: Race Dynamics, 2002.
- [7] D. J. N. Limebeer, R. S. Sharp, and S. Evangelou, “The stability of motorcycles under acceleration and braking,” *Proc. I. Mech. E., Part C, Journal of Mechanical Engineering Science*, vol. 215, pp. 1095–1109, 2001.
- [8] V. Cossalter, R. Lot, and F. Maggio, “On the Stability of Motorcycle during Braking,” in *SAE Small Engine Technology Conference & Exhibition*, Graz, Austria, September 2004, 2004, sAE Paper number: 2004-32-0018 / 20044305.
- [9] G. Ferretti, S. Savaresi, F. Schiavo, and M. Tanelli, “Modelling and simulation of motorcycle dynamics for Active Control Systems Prototyping,” in *Proceedings of the 5th MATHMOD Conference*, Vienna, Austria, 2006.
- [10] R. S. Sharp, S. Evangelou, and D. J. N. Limebeer, “Advances in the modelling of motorcycle dynamics,” *Multibody System Dynamics*, vol. 12, pp. 251–283, 2004.
- [11] J. Andreasson, “Vehicle dynamics library,” in *Proceedings of the 3rd International Modelica Conference*, Linköping, Sweden, 2003.
- [12] H. Pacejka, *Tyre and Vehicle Dynamics*. Oxford: Butterworth Heinemann, 2002.
- [13] H. Olsson, H. Tummescheit, and H. Elmqvist, “Using automatic differentiation for partial derivatives of functions in modelica,” in *Proc. 4th International Modelica Conference*, Hamburg-Harburg, Germany, March 7-84, 2005, pp. 105–112.
- [14] L. Marescotti, “Modellazione del sistema pilota-veicolo a due ruote in ambiente integrato Matlab-Adams,” Master’s thesis, Università di Pisa, 2003.

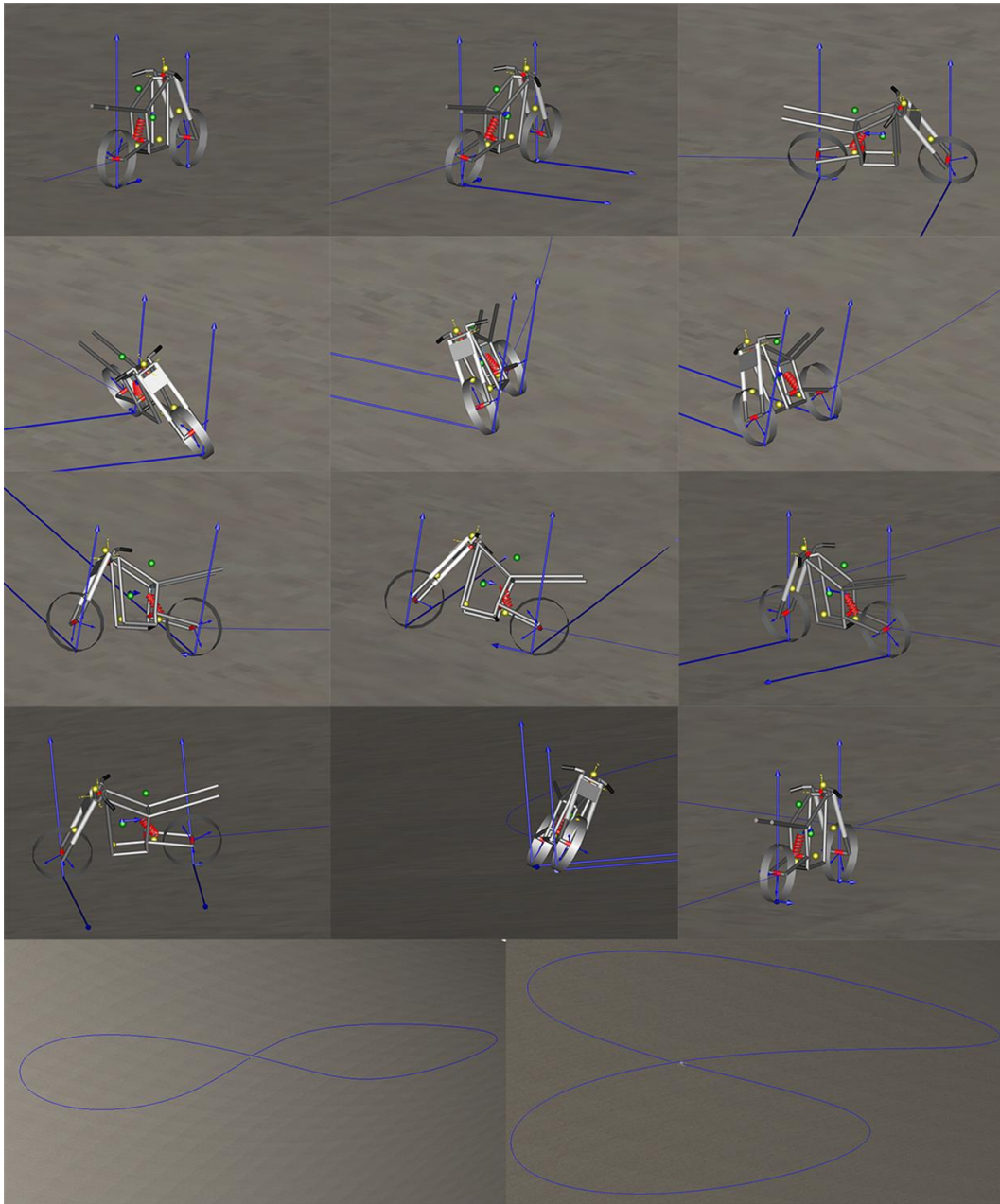


Figure 12: Screenshot of an eight maneuver and of the simulated trajectory.

Development and Verification of a Series Car Modelica/Dymola Multi-Body Model to Investigate Vehicle Dynamics Systems

Christian Knobel* Gabriel Janin‡ Andrew Woodruff§

*BMW Group Research and Technology, Munich, Germany, christian.knobel@bmw.de

‡École Nationale Supérieure de Techniques Avancées, Paris, France, gabriel.janin@ensta.org

§Modelon AB, Lund, Sweden, andrew.woodruff@modelon.se

Abstract

The development and the verification of a Multi-body model of a series production vehicle in Modelica/Dymola is presented. The model is used to investigate and to compare any possible configuration of actuators to control vehicle dynamics with a general control approach based on model inversion and a non-linear online optimization.

Keywords: Multi-body Vehicle Model, Vehicle Dynamics, Model Verification, Model Validation, Active Vehicle Dynamics Systems, Tire Model, Suspension Kinematics, Suspension Compliance

1 Introduction

Systems for control of Vehicle Dynamics went to series production for the first time in 1978 with the limitation of brake pressure to avoid locking the wheels to ensure cornering under all braking conditions. Braking individual wheels (independently from the drivers commands) to stabilize the vehicle at the driving limit went to series production in 1995. In the last few years, control systems for vehicle dynamics with additional actuators to control steering, drive torque distribution and wheel load distribution have entered the market.

All of these systems acting on the force allocation from the center of gravity (CG) to the four tire contact patches (TCP) and on the force transfer at the TCPs. This strong interdependence between these systems¹ is the reason why independent operation of more than one of them is only possible with a loss of potential to prevent critical interferences. In [2] (cf. also [3], [4], [5], [6] and [7]) a general approach was introduced to investigate the ref-

¹cf. [1] for an overview and a detailed classification of systems for vehicle motion control.

erence behavior (best possible allocation and transfer of forces acting on the vehicle) for any configuration of actuators controlling vehicle dynamics including all steering angles $\delta = [\delta_1 \ \delta_2 \ \delta_3 \ \delta_4]^T$, brake/drive torques $M = [M_1 \ M_2 \ M_3 \ M_4]^T$, wheel loads $F_z = [F_{z1} \ F_{z2} \ F_{z3} \ F_{z4}]^T$ and even camber angles $\gamma = [\gamma_1 \ \gamma_2 \ \gamma_3 \ \gamma_4]^T$. The comparison of the reference operation of different configurations may support decisions in the future development of vehicle dynamics. The investigation of the reference behavior also supports controller development and the dynamic specification of the actuators, or makes it possible to investigate the potential loss if the applied actuators have less dynamics compared to the ideal required dynamics. The reconfigurable behavior of the general approach allows further investigation of the impact of actuator failures on vehicle dynamics for reliability investigations.

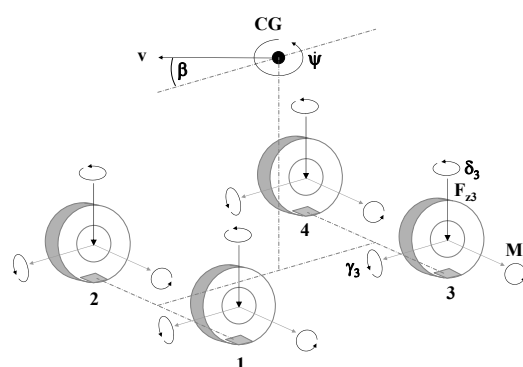


Figure 1: Planar vehicle model with influencing variables and vehicle motion y

Because the approach presented in [2] is based on model inversion of a plane vehicle model with the plane motion $y = [\psi \ \beta \ v]^T$ described by the yaw rate ψ , the body slip angle β and the velocity v (cf. Figure 1), a verified multi-body model is needed as ve-

hicle model to ensure that the control approach works also with all effects neglected during the controller design. Using the verified Modelica/Dymola Multi-body vehicle model presented in this paper as a vehicle model for the control approach allows investigation and comparison of all possible configurations of available actuators quickly and easily.

The possibility to model multi-body suspension assemblies, controllers, hydraulic and mechatronic actuators in one and the same environment was the reason to choose Dymola as the modeling and simulation tool. Development and verification of the model was done bottom-up. Multi-body front and rear suspension, tires, steering system, power train and the body were modeled and then verified separately with test rig results as shown in Section 2. Creating the multi-body vehicle model by connecting those subsystems together is presented in Section 3 as well as the verification process of the full vehicle through objective test maneuvers with a series car equipped with additional measurement technology. Finally, an application example is presented in Section 4.

2 Development and Verification of Subsystems

Using the Multi-Body Library [8] and the Vehicle Dynamics Library [9], tire, suspension, body and environment for the multi-body model were constructed. Simple models for the steering and the power train system are developed only for the verification of the multi-body vehicle model with a conventional series production vehicle.

2.1 Tire

Pacejka's Magic Formula [10] is used to model the plane transfer behavior of the tire, which calculates first the pure forces

$$\begin{aligned} F'_{xoi} &= D \sin(C \arctan(B\kappa - E(B\kappa - \arctan B\kappa))) \\ F'_{yoi} &= D \sin(C \arctan(B\alpha - E(B\alpha - \arctan B\alpha))) \end{aligned} \quad (1)$$

$i \in \{1 \dots 4\}$ designated with ' to indicate the representation with the wheel coordinate system out of the inputs longitudinal slip κ , tire side slip angle α , wheel load F_z and camber angle γ . Secondly,

$$\begin{aligned} F'_{xi} &= F'_{xoi} \cos(C \arctan(B\alpha - E(B\alpha - \arctan B\alpha))) \\ F'_{yi} &= F'_{yoi} \cos(C \arctan(B\kappa - E(B\kappa - \arctan B\kappa))) \end{aligned} \quad (2)$$

the interdependence between the longitudinal and lateral tire forces is considered where the peak parameter $D(F_z, \gamma)$, the shape parameter C , the stiffness parameter $B(F_z, \gamma)$ and the curvature parameter $E(F_z, \gamma)$ are different for (1), (2) and for longitudinal and lateral directions, respectively. For the identification of these parameters, an error minimization is used to fit the model result to the test rig results of the tire used (cf. Figure 2 and Figure 3). Because braking (negative longitudinal tire forces) is more important for vehicle dynamics control than accelerating (positive longitudinal tire forces), different weighting factors are used to get a better correlation of the negative longitudinal tire forces.

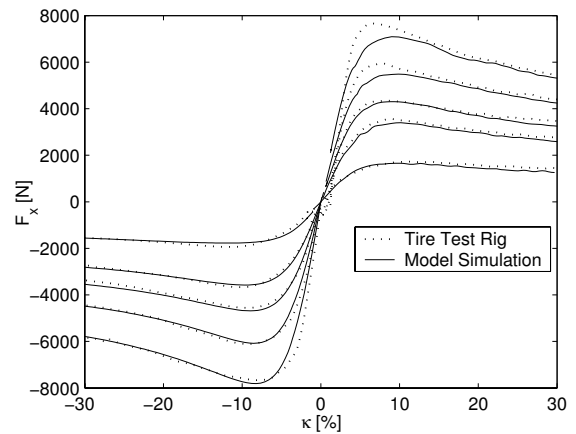


Figure 2: Longitudinal force F_x over longitudinal slip κ for different wheel loads F_z

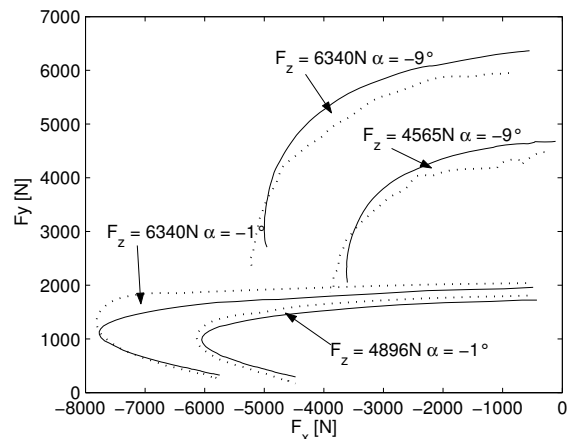


Figure 3: Lateral force F_y over longitudinal force F_x (longitudinal slip κ sweeps at different tire side slip angles α and wheel loads F_z) known as a Krempel Graph

2.2 Suspension

For the front and rear suspension, ADAMS models could be used as source to model the McPherson front suspension (cf. Figure 4) and the semi-trailing

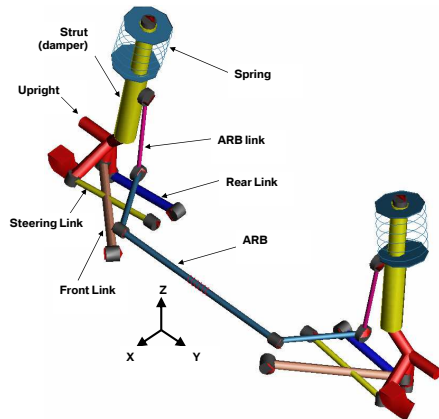


Figure 4: McPherson front suspension design

arm rear suspension design (cf. Figure 5) in Modelica/Dymola.

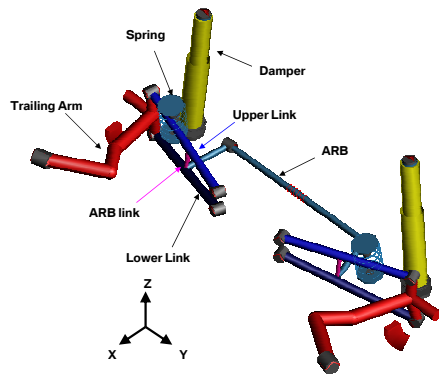


Figure 5: Semi-trailing arm rear suspension design

The default models for those suspension designs in the Vehicle Dynamics Library [9] could not be used without customizing and modifying the design, the joint location and the kinematic relationships to match the behavior of the source models in ADAMS.

The McPherson front suspension used independent lower rods instead of the conventional control arm in [9]. The rear suspension used a trailing arm design with two guiding links, and the body spring and anti-roll subsystems were attached to the upper guiding link (cf. [11]). Non-linear bump stops were added on the damper's tube of the front and rear suspension.

The hard points for all joint locations of the model need to be adjusted to agree with the ADAMS source model.

The kinematics of the suspension models are verified using a vertical travel sweep test rig which needs to be modeled in Modelica/Dymola as well. Camber and toe changes are plotted to verify the kinematic behavior of the suspension models with the source model and the real suspension of the used series vehicle (cf. Figure 6).

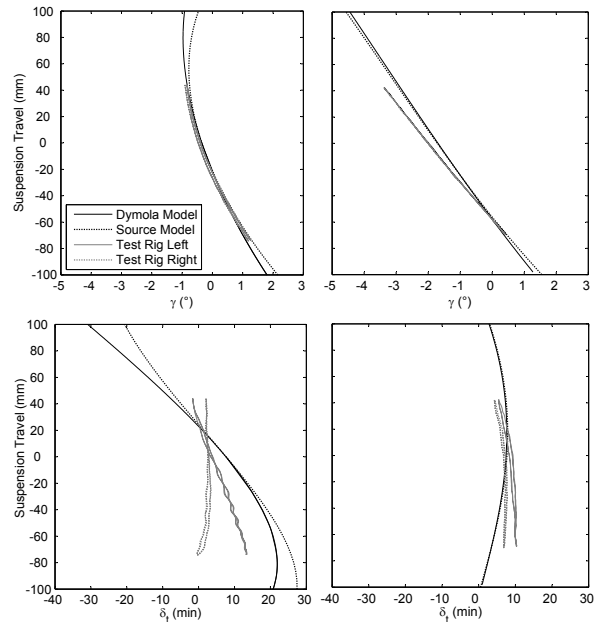


Figure 6: Kinematic analysis: camber in degrees and toe in angular minutes for front (left) and rear suspension (right)

Simulation of the rigid suspension model (without any bushings) was impossible and caused singularity errors. After the implementation of bushings, simulation of the multi-body front and rear suspensions was possible. However, a pure investigation of the rigid kinematics was only possible using very stiff bushings, which is the reason for the differences between the source model and the Modelica/Dymola model in Figure 6.

The kinematics of the real suspension could only be verified with bushings, which is again the main reason for the differences between the model results and the real test rig results shown in Figure 6.

2.3 Further Subsystems of the Vehicle

After modeling the tires and suspensions, adding body, power train and steering system models, as well as the vehicle's environment, is necessary to the complete multi-body vehicle model.

The vehicle's body is considered to be rigid and its mass is distributed as follows: one summarized sprung

mass, including the driver, one passenger and fuel is in the body whereas the unsprung mass of the wheels including brake caliper and rotor and their links is distributed to the four wheels. The vehicle's inertias at the CG are identified on a pendulum test rig.

The complexity and accuracy of the power train and steering models are rather low because they are only used to get reasonable connections between the driver's inputs and the brake/drive torques M and the wheel steer angles δ . The former uses a speed controller and a differential gear to distribute the torques to the four wheels similar to the series production vehicle. The latter consists a rack-and-pinion steering system including a rotational spring in the steering shaft.

The study of a full-vehicle model requires the modeling of its environment. The equations of motion can only be solved by having a complete description of physical system. Therefore, the interaction between the vehicle and the world must be taken into account. It consists the interactions between vehicle and driver, vehicle and air, and tire and road. The road is modeled by a flat surface with with a road friction coefficient μ . The aerodynamic drag force $F_x^{aero} = -\frac{1}{2}\rho c_x v^2$ applied at the center of gravity simplifies the interaction between air and vehicle. These environments are from the Vehicle Dynamics Library [9]. Only driver models need to be built up to be able to simulate the objective test maneuvers for the verification in Section 3.

The active control actuators are modeled by ideal revolute joints inserted at the rigid connection between the suspension and wheel subsystems. This meant the wheels could be manipulated directly and without altering the suspension geometry. Passive systems are represented by constant values as inputs for the ideal actuators.

3 Development and Verification of the Multi-body Vehicle Model

Connecting the subsystems from Section 2 creates the multi-body vehicle model.

Important steps are the definition, design and implementation of the model. A more important step is to check if the model matches real vehicle behavior. Therefore, the vehicle and the model behavior are compared with objective test maneuvers such as steady state cornering (steady state behavior), steering steps (dynamic behavior in the time domain) and sine-sweeps (dynamic behavior in the frequency domain). Body side slip angle β and velocity v are measured using an optical Correvit sensor, roll ϕ and pitch θ

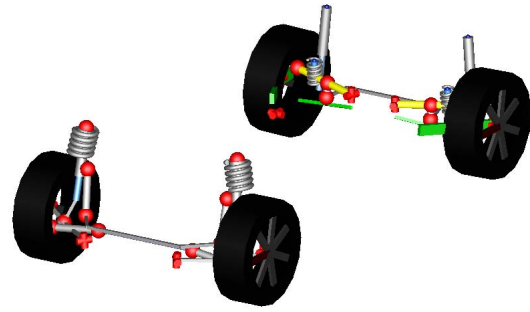


Figure 7: Chassis representation of the multi-body vehicle model

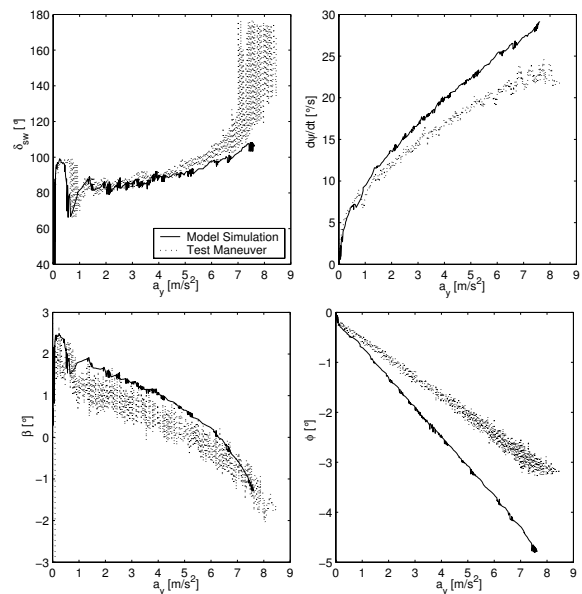
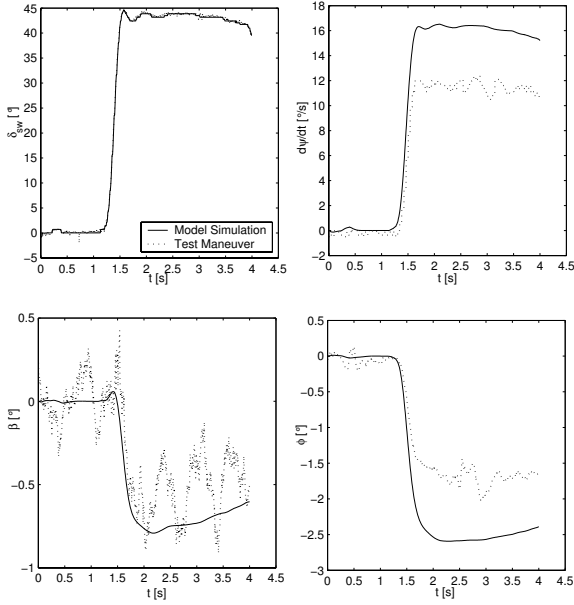


Figure 8: Steady state cornering, $r=40m$, dry road $\mu=1$

are measured indirectly by suspension travel sensors at all four wheels, and all translational accelerations a_x, a_y, a_z as well as the rotational rates $\dot{\psi}, \dot{\phi}, \dot{\theta}$ are measured by a sensor cluster located at the CG. The steering wheel angle δ_{sw} and the steering wheel torque M_{sw} are measured using an instrumented steering wheel.

The results of the verification without any fitting of uncertain parameters like stiffness of the bushings, or friction coefficient μ of the road are presented in Figure 8 and Figure 9.

The main reason for the higher yaw rate generated by the model in both maneuvers is the uncertain road friction coefficient μ . The higher roll in both maneuvers is caused by different stiffnesses of the body springs used in the model and the real vehicle.


 Figure 9: Step steer, $v=70$ km/h, dry road $\mu=1$

4 Application of the Model

Exchanging the conventional steering and power train system of the verified vehicle model and using the ideal actuators as described in 2.3 leads to a generic configuration for vehicle dynamics control. All steering angles δ , drive/brake torques M , wheel loads F_z and camber angles γ (cf. Figure 1) could be used passively or actively controlled by the general allocation approach presented in [2]. A non-linear online optimization calculates the arbitrary parameters for the under determined inverses of the over-actuated² plane motion vehicle model (cf. 1). The number of the arbitrary parameters depends on the available actuators for the influencing variables. These arbitrary parameters are always used by the non-linear online optimization to minimize the maximum adhesion potential utilization

$$\eta_i^2 = \left(\frac{F_{xi}}{F_{xi\max}} \right)^2 + \left(\frac{F_{yi}}{F_{yi\max}} \right)^2 \quad (3)$$

($0 \leq \eta_i \leq 1$) of the four tires $i \in \{1 \dots 4\}$ is approximated by an elliptic relation. The forces $F_{xi\max}$ and $F_{yi\max}$ depend on the wheel load F_{zi} and the camber angles γ_i . The control commands out of these optimization are used as inputs for the multi-body vehicle (cf. Figure 10). Inputs for the optimization are the torque and forces $u = [M_{zCG} \ F_{xCG} \ F_{yCG}]^T$ acting on the center of gravity. The allocation of these forces to the TCPs and the force transfer in the TCPs are optimized

²cf. [12] for definition and examples

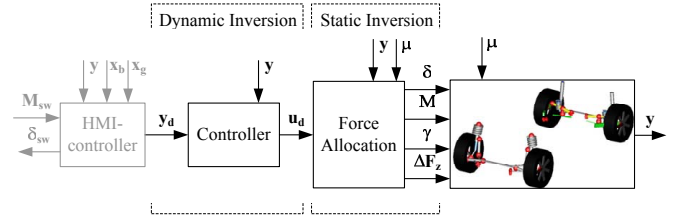


Figure 10: Control loop with the multi-body vehicle model

with the optimization objective

$$\min \max \eta_i \quad (4)$$

This setup facilitates investigation into the reference behavior (best possible allocation and transfer of forces acting on the vehicle) of the vehicle dynamics for every configuration of available actuators influencing vehicle dynamics for the verified multi-body vehicle model.

Changing the number of available actuation during a driving maneuver allows investigation into the impact of actuator failures on vehicle dynamics, which may support reliability investigations of active vehicle dynamics systems.

Such an investigation is presented as an exemplary application of the presented multi-body vehicle model (cf. Figure 11, Figure 12 and Figure 13). The front right steering actuator of a vehicle (equipped with four steering actuators, four drive torque actuators and four actuators to control the wheel load distribution) fails. The failure occurs after one second of driving an open loop single lane change at a constant speed $v=70$ km/h (cf. Figure 13). All actuators and actuator dynamics are limited for the optimization (4) to values of actual available actuators. The actuator fails in the worst case situation at maximum steering angle of $\delta = 1.8$ degrees and is assumed to be self-locked after the failure occurred. To compensate this fixed steering error, the vehicle exhibits the same amount of body slip angle as can be seen in Figure 13. Steering back to straight driving again the failure wheel becomes the outer wheel (with respect to the center of the corner) with more wheel load. Therefore, the optimization reduces the wheel load at this wheel as much as possible. However, the steering angles and drive torques at the other wheels are much higher compared to the usual driving condition without failure (right side of Figures 11, 12 and 13). The maneuver presented is close to the physical driving limit since two tires have already reached their maximum possible adhesion potential utilization (cf. Figure 12). The lateral acceler-

ation a_y is reaching 4 m/s^2 at the minimum and maximum of the yaw rate ψ . This performance meets the actual specification of flat run-flat tires.

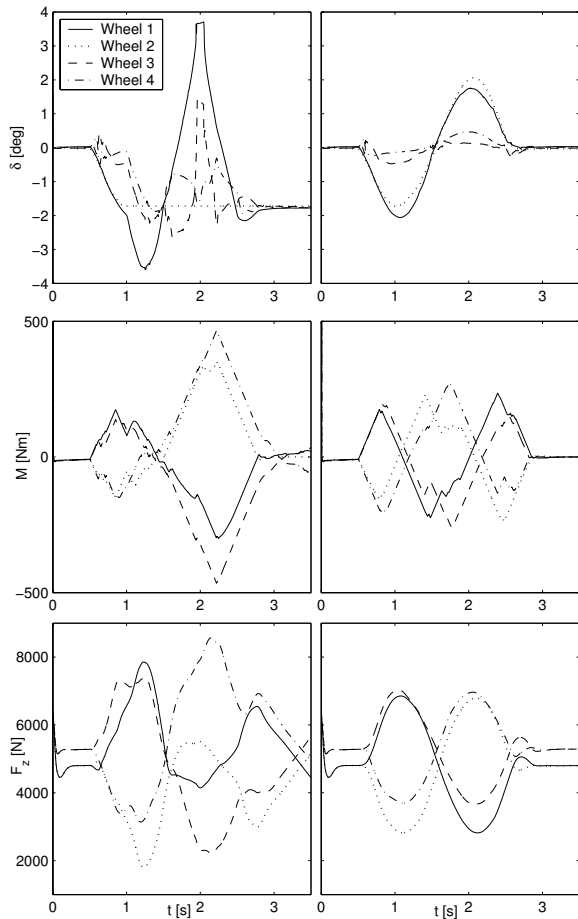


Figure 11: Influencing variables of vehicle with steering actuator failure (left) and usual working vehicle (right)

5 Conclusion and Outlook

The development and the verification of a multi-body model of a series production vehicle is presented. This vehicle model was used to investigate and compare any possible configuration of actuators to control vehicle dynamics. In this context, Modelica/Dymola has proven to be a practical environment for future development of vehicle models including mechatronic and hydraulic actuators, multi-body suspensions and controllers. To develop, verify and use Modelica/Dymola models in an efficient way, however, interfaces to CAD systems to import CAD model data and interfaces to real time environments are desirable. Especially for this project, a library for non-linear optimization was missed, which was the reason why

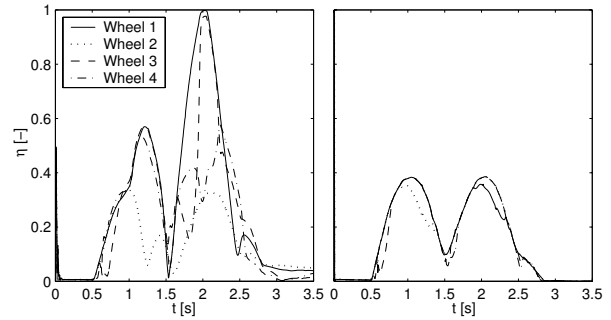


Figure 12: Adhesion potential utilization of failure (left) and usual vehicle (right)

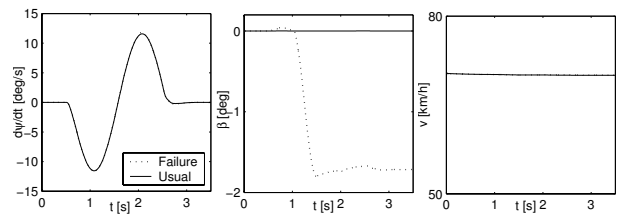


Figure 13: Plane vehicle motion of the vehicle with steering actuator failure (-) and usual working vehicle (:)

this part of the presented control approach was realized in MATLAB/Simulink. The presented Modelica/Dymola multi-body model was implemented into MATLAB/Simulink using the Simulink Interface of Dymola.

The outlook of the presented project is to improve the matching of the model by using an error minimization. Parameters for the error minimization are probably the uncertain stiffness of the bushings and the road friction coefficient μ .

The presented example of steering actuator failure could be improved by adding a strategy of actively controlled camber (assuming the availability of such a system) for steering actuator failures which counteracts the failure force generated by the tire side slip angle α_i of the failure wheel.

References

- [1] J. Andreasson, C. Knobel, and T. Bünte. On Road Vehicle Motion Control - striving towards synergy. In *Proc. of 8th International Symposium on Advanced Vehicle Control AVEC*, 2006.
- [2] C. Knobel, A. Pruckner, and T. Bünte. Optimized Control Allocation - A General Approach to Control and to Investigate the Motion of Over-

- actuated Vehicles. Submitted paper for 4th IFAC Symposium on Mechatronic Systems, 09 2006.
- [3] Y. Hattori, K. Koibuchi, and T. Yokoyama. Force and Moment Control with Nonlinear Optimum Distribution for Vehicle Dynamics. In *Proc. of 6th International Symposium on Advanced Vehicle Control AVEC*, 2002.
- [4] E. Ono, Y. Hattori, Y. Muragishi, and K. Koibuchi. Vehicle Dynamics Control Based on Tire Grip Margin. In *Proc. of 7th International Symposium on Advanced Vehicle Control AVEC*, 2004.
- [5] R. Orend. Steuerung der Fahrzeugbewegung mit minimaler Kraftschlussausnutzung an allen vier Rädern. In *Proc. of Steuerung und Regelung von Fahrzeugen und Motoren - Autoreg. VDI*, 2004.
- [6] J. Andreasson and T. Bunte. Global Chassis Control Based on Inverse Vehicle Dynamics Models. Presented at XIX IAVSD World Congress, 10 2005.
- [7] T. Bunte and J. Andreasson. Integrierte Fahrwerkregelung mit minimierter Kraftschlussausnutzung auf der Basis dynamischer Inversion. In *Proc. of Steuerung und Regelung von Fahrzeugen und Motoren - Autoreg, Wiesloch*, 2006.
- [8] M. Otter, H. Elmqvist, and S. E. Mattsson. The New Modelica MultiBody Library. In *Proc. of the 3. International Modelica Conference*, pages 311–330, 2003.
- [9] J. Andreasson. Vehicle dynamics library. In *Proc. of the 3rd International Modelica Conference*. Modelica Association, 2003.
- [10] H. B. Pacejka. *Tyre and Vehicle Dynamics*. Butterworth-Heinemann, Oxford, 2002.
- [11] A. Woodruff. Camber Prevention Methods using a Modelica/Dymola Multi-body Vehicle Model. Master’s thesis, Mechanical and Materials Engineering Department, Queen’s University, Kingston, ON, Canada, 2006.
- [12] M. Valášek. Design and Control of Under-Actuated and Over-Actuated Mechanical Systems - Challenges of Mechanics and Mechatronics. *Vehicle System Dynamics*, 40:37–50, 2003.

Session 2c

Language, Tools and Algorithms 2

Modeling and simulation of differential equations in Scicos

Masoud Najafi Ramine Nikoukhah
INRIA-Rocquencourt, Domaine de Voluceau,
78153, Le Chesnay Cedex France

Abstract

Block diagram method is an old approach for the modeling and simulation of differential equations. Modeling and simulation of some kind of differential equations such as differential-algebraic equations (DAE) is cumbersome, difficult, or even impossible with this approach. Scicos which is a modeling and simulation software based on Block diagram approach has recently been developed to simulate Modelica programs. In this paper, it will be explained the way different classes of DAE can easily be specified in Modelica and simulated in Scicos.

Keywords: hybrid differential equations; numerical solver; simulation; Modelica; Scicos

1 Introduction

Scilab¹ is a free, and open-source software for scientific calculation. Scicos² is a toolbox of Scilab and provides an environment for modeling and simulation of dynamical systems [6, 4]. For many applications, the Scilab/Scicos environment provides an open-source alternative to Matlab/Simulink and MatrixX [14, 15]. Scicos includes a graphical editor for constructing models by interconnecting blocks, representing predefined or user defined functions, a compiler, a simulator, and code generation facilities. A Scicos block diagram is composed of blocks and connection links. A block corresponds to an operation and by interconnecting blocks through links, we can construct a model, or an algorithm.

The Scicos blocks represent elementary systems that can be used as model building blocks. They can have several inputs and outputs, continuous-time states, discrete-time states, zero-crossing functions, etc. Scicos allows customization with regard to incorporating user C, Fortran, or Scilab codes. Scicos translates the block diagram model into a system of Ordinary Dif-

ferential Equations (ODE) or Differential Algebraic Equation (DAE) and applies an ODE or a DAE solver in order to perform a simulation. A block diagram system representation can only be used to model ODEs and a special class of DAEs, while the solvers used in Scicos support a larger class of DAEs [9]. In this paper we will explain the way Scicos environment has been developed to write and simulate a large class of differential equations, called hybrid differential equations. To get an idea what a Scicos model looks like, a model of a simple control system implemented in Scicos has been shown in Figure 1. In Figure 1 the `Clock` block

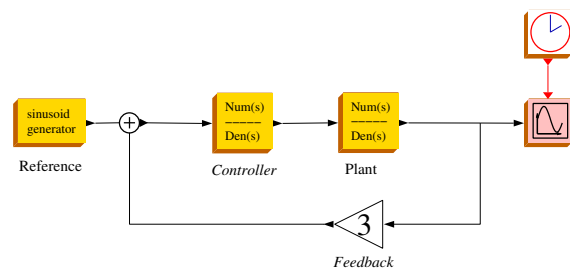


Figure 1: A Scicos model of a control system

generates a periodic activation signal (event) that activates the `Scope` block. At event times the scope reads its input signal and displays them.

2 Mathematical Background

A differential equation expressed either by an *Ordinary Differential Equations* (ODE), *i.e.*,

$$\dot{x} = f(x, u, t)$$

where \dot{x} denotes the derivative of x , the state variables, with respect to the time variable t , and u is the input vector variable, or by *Differential Algebraic Equations* (DAE) [2, 3, 5], *i.e.*,

$$\begin{cases} \dot{x} = f(x, y, u, t) \\ 0 = g(x, y, u, t) \end{cases} \quad (1)$$

¹www.scilab.org

²www.scicos.org

where (1-a) is the differential part and (1-b) is the algebraic part of the DAE. The equation set (1) is a *semi-explicit* DAE, where the differential and the algebraic parts are decomposed. If we cast (1) in the form of

$$0 = F(\dot{z}, z, t), \quad z = \begin{pmatrix} x \\ y \end{pmatrix} \quad (2)$$

This system is called a *fully implicit* DAE. Note that if $\frac{\partial F}{\partial \dot{z}}$ is non-singular, then it is possible to formally solve \dot{z} as a function of z in order to obtain an ODE. However, if it is singular, this is no longer possible and the solution z has to satisfy certain algebraic constraints.

DAEs are characterized by their index. The *index* of a DAE, e.g., (2), is the smallest number of differentiation of (2) to obtain an ODE by algebraic manipulations [7]. In general, the higher the index, the greater the numerical difficulty one encounters, when trying to integrate the DAE numerically. In a semi-explicit index-1 DAE (1), variables whose derivatives appear in DAE are called *differential variables* and the other ones are called *algebraic*, i.e., x in (1-a) is differential and y in (1-b) is algebraic [16, 17].

2.1 Numerical solvers of Scicos

In order to integrate differential equations or simulate any model, Scicos uses two numerical solvers; `Lsodar` [8, 16] and `DASKR` [16, 2]. `Lsodar` is an ODE solver which is used when the Scicos diagram represents an ODE. If a diagram represents a DAE, `DASKR` is used. `DASKR` is a variable step, variable order index-1 DAE solver. The solver properties discussed here are those of `DASKR` and `LSODAR`; however these properties are common to most modern solvers. The most important of these properties will be explained in the following subsections [11].

Consistent initial condition: Most of the problems in using standard solvers are common to both ODE and DAE solvers. However, there is an additional difficulty with the DAE case: the problem of re-initialization and finding consistent initial conditions. Simulation of an ODE can be started from any initial state, but simulation of a DAE should be started from a consistent initial state. This is an additional difficulty in simulation of DAEs.

Continuity criteria for numerical solver: `DASKR` and `LSODAR` use a variable order variable step-size BDF (backward differentiation formula) method to integrate. The BDF methods normally need continuity in variables and their derivatives [1]. Consequently,

`DASKR` and `LSODAR` require that the system be sufficiently smooth over an integration period. This means that simulator must make sure to stop and reinitialize the solver at each potential point of non-smoothness (discontinuity, discontinuity in the derivative, etc.) of the ODE/DAE. These ODE/DAEs require some additional solver features, such as event detection or root finding.

Event detection: The ability to detect the time when a discontinuity occurs, or more precisely, the time when a function crosses some given value (by default considered zero) is of capital importance in simulation of DAEs. For this purpose, the discontinuity function is given to the solver as a zero-crossing function. When a zero-crossing occurs, the solver stops the integration and returns the exact crossing time to the main program. So, it is important to halt the solver and restart at the discontinuity point. The numerical solver can also provide the direction in which a function has crossed the zero.

2.2 Discontinuity handling in the simulator of Scicos

DAEs may have discontinuities or may be variable structure or may change at certain points in time. Such types of DAEs are called hybrid DAEs. A hybrid DAE is a way of describing non-smooth multi-model systems in terms of a finite number of smooth systems. The idea is to divide the state space of the system into different regions. It is assumed that the system is described in terms of a single smooth DAE within each region. A simple example of a multi-model DAE is:

$$\text{If } (g(x) > 0) \quad \text{then } f_1(\dot{x}, x, u, t) = 0 \\ \quad \quad \quad \text{else } f_2(\dot{x}, x, u, t) = 0 \quad (3)$$

where the DAE has two models: the first one is on when $(g(x) > 0)$ and the second is when the condition is not true. This switching may cause a discontinuity in the signals, so they cannot be integrated by the numerical solvers. In order to cope with this problem, the discontinuity should be detected and the solver be reinitialized after the discontinuity point. To detect and localize the discontinuity time, solvers use zero-crossing functions that cross zero over the discontinuity point. For example, for (3), we use $g(x)$ as the zero-crossing function [11].

For localizing the discontinuity point, the simulator associates `Mode` variables with each zero-crossing functions in Scicos. `Mode` variables are used to assign and fix an ODE/DAE in every time interval between

each two discontinuities. In general, when the numerical solver is called, the system of equation should not be changed. During integration, the zero-crossing functions that indicate the conditions for the model change are examined by the solver. In case of any zero-crossing, the Mode variables should be updated to feed another ODE/DAE to the solver. It should be noted that Mode variables are defined in the Scicos simulator and is transparent to the user. In Scicos, a Mode variable is assigned systematically to each discontinuity point, characterized by an If-then-Else block. Scicos considers the system (3) in the following form:

$$0 = \begin{cases} f_1(\dot{x}, x, u, t) & \text{if Mode} = 1 \\ f_2(\dot{x}, x, u, t) & \text{if Mode} = 2 \end{cases}$$

when $(g(x) \geq 0)$ **then** Mode = 1
when $(g(x) < 0)$ **then** Mode = 2

By default, for any discontinuity in the model, a Mode is used. But it should be noted that any If-then-Else, not resulting in a discontinuity, can be used without Mode. If the resulting if-then-else expression is smooth, the modeler has the possibility to give this extra information to the simulator in order to avoid these unnecessary solver reinitialization. That is why there is a parameter in If-then-Else blocks that lets the user define whether the block is used with or without zero-crossing. The Mode variables should not be used in some cases. For example, when a function is not defined everywhere and might be called near the limit of validity. In DAE (4),

$$\begin{aligned} \dot{x} &= -xy - x + y \\ y &= \begin{cases} -1 + \sqrt{x} & \text{if } x \geq 0 \\ -1 + \sqrt{-x} & \text{if } x < 0 \end{cases} \end{aligned} \quad (4)$$

if the Mode variable is used the first model (*i.e.*, \sqrt{x}) is employed until the discontinuity point $x = 0$ is detected. During the search process the solver uses \sqrt{x} for $x < 0$ to localize the crossing point. This will cause a failure in the integration, so the Mode variable should not be used to permit the solver probe beyond the discontinuity point.

3 Implementing differential equations with block diagram approach

Block diagram implementation is an old method to represent differential equations. In this method, through the use of multipliers, adders, integrators, etc. a differential equations is constructed graphically.

Block diagram consists of blocks that are connected by arrows and each block is a transducer that transforms the incoming signals to one or more output signals. A block can represent simple arithmetic operations or functions without memory, but also operations whose results are dependent on previous inputs to the block, *i.e.*, with memory. Several software tools such as Scicos, Simulink, SystemBuild, etc. use block diagram method to model and simulate dynamical systems. As an example the diagram in Figure 2 displays the graphical representation of equation (5).

$$\begin{cases} \dot{x}_1 = -0.04x_1 + 10^4x_2x_3 \\ \dot{x}_2 = 0.04x_1 - 10^4x_2x_3 - 3 \times 10^7x_2^2 \\ \text{if } (1 + \sin(0.1t) - x_2 - x_1 > 0.5) \\ \quad \text{then } x_3 = -10x_1, \\ \quad \text{else } x_3 = 10x_1. \end{cases} \quad (5)$$

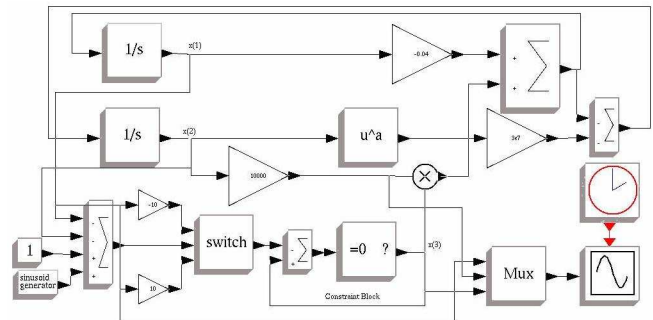


Figure 2: Block diagram implementation of DAE (5) in Scicos

3.1 Shortcomings in the block diagram approach

It is often possible to model differential equations via block diagram approach, but in fact it is not an easy and efficient way. There are several shortcomings. In a block diagram model a small change in the differential equations follows with another study to rearrange the entire structure of the block diagram which may have little similarity to the previous version. Furthermore, most of the general-purpose simulation softwares on the market such as ACSL, Simulink and SystemBuild that use block diagram approach assume that a system can be decomposed into block diagram structures with causal interactions. This means that the models should be expressed as an interconnection of models on semi-explicit form, *i.e.*, $M(t)\dot{x} = f(x, t)$, where the matrix $M(t)$ is singular [18]. Although theoretically any DAE index-1, can be transformed into a semi-explicit DAE,

since such a transformation is done manually, it is time consuming and sometimes it is practically impossible. With these shortcomings, there is a rising need to have an appropriate framework for DAE representation. A convenient way is to work with the DAEs as the text. The reader may think of simulation methods in which DAEs are expressed in a textual environment, such as writing computer programs and invoking the numerical solvers, or writing Scilab or Matlab script files. But these methods are not efficient and do not provide a proper framework to control and interact with the numerical solver. In fact, the numerical solver considers the DAE as a black box and the internal discontinuities remain hidden. For example, if a discontinuous DAE is simulated directly by `Dassl` function in Scilab or by `ode15s` function in Matlab, the simulation would fail. As an example, when we tested DAE (5) with Matlab, the simulation failed and the following error message raised:

```
>> Warning: Failure at t=5.235698e+00.
Unable to meet integration tolerances
without reducing the step size
below the smallest value allowed
(1.860093e-14) at time t.
```

The problem lies in the solver control, *i.e.*, a discontinuous DAE cannot be integrated by the numerical solver without discontinuity handling and a good restart managements. Scicos has recently been developed to simulate non-casual models and the user can write physical models symbolically with a new the Modelica language [10].

4 Modelica language

Modelica³ is a freely available, object oriented, general purpose language for modeling of physical systems, *e.g.*, mechanical, electrical and control systems. The Modelica language allows a direct and convenient specification of systems with continuous-time and discrete-time dynamics. Although Modelica is a rich language having the capacity to handle continuous-time and discrete-time behaviors, in this paper we will focus mainly on modeling hybrid differential equations. A Modelica program or model like any other computer language is composed of a variable or component declaration section and an equation section. Suppose that we want to model DAE (6) in

Modelica.

$$\begin{cases} \dot{x} = x - xy \\ \dot{y} = yx - 2y \\ x(0) = 1 \\ y(0) = 2 \end{cases} \quad (6)$$

This DAE consists of two differential variables, *i.e.*, x and y . They are continuous-time `Real` type variables and their initial value at beginning (time=0) are 1 and 2, respectively. Here is the Modelica program:

```
class Oscillator "Oscillator model"
  Real x(start =1), y(start=2);
equation
  der(x) = x-x*y;
  der(y) = x*y-2*y;
end Oscillator;
```

In the first part of the program, two variables and their initial values are declared. The next part of the program contains the equations. In this section, there are two equations for two unknowns, *i.e.*, $\text{der}(x)$ and $\text{der}(y)$ the time derivatives of x and y . In Modelica, equations are composed of expressions both on the left hand side and the right hand side. It is neither required to write the equations in form of assignments, nor to write the equations in a specified order. It is, however, important to provide equal number of unknowns and equations. For instance, here is the above program that has been rewritten without changing the model mathematically:

```
class Oscillator2 "Oscillator model"
  Real x(start =1), y(start=2), v;
equation
  der(x)-x+v=0;
  0=-der(y)-2*y+ v;
  x*y=v;
end Oscillator2;
```

Note that in this program, v is an algebraic variable. In Modelica the initial value of all variables can be specified. Theoretically specifying initial value of algebraic states is not required. This, however, would help the numerical solver to find the consistent initial condition for highly nonlinear DAEs or in case where there are several solutions it acts as a guess value to help the solver to catch the desired solution [13].

In Modelica models that are in fact the mathematical equations, it is not possible to classify (at least a-priori) the variables as inputs and outputs. This type of models are called *acausal* models. That is in contrast with *causal* models where there are explicit inputs and outputs and the outputs are computed as function of inputs and other internal variables. To make an analogy

³www.Modelica.org

with computer programming languages, causal models correspond to the use of assignment statements where the right-hand sides of the equations are evaluated and the result of the evaluation is assigned to the variables on the left-hand side of the equations [10].

An acausal model cannot be simulated directly, it should be transformed into a causal model. In general, it is possible to convert an acausal model into a causal model by rewriting the equations and finding the appropriate causality structure in equations. In Scicos this is done by the Modelica compiler. The Modelica compiler receives the Modelica program and extracts the necessary information for the numerical simulation and generates a usable C program for Scicos.

5 Hybrid DAE modeling in Modelica

In block diagram approach one cannot model fully-implicit DAEs directly. In Modelica this constraint does not exist and any DAE⁴ can be expressed without making any effort to transform them into an explicit form. A multi-model DAE or a DAE with discontinuity is defined with `If-then-else` constructs. Note that an `If` should always be used with an `else`. As an example, a Modelica Code for the DAE (5) follows:

```
class DAE2
  Real x1(start=1.0), x2 (start=0.0), x3, xs;
equation
  der(x1) = -0.04*x1 + 1e4*x2*x3;
  der(x2) = 0.04*x1 - 1e4*x2*x3 - 3e7*x2*x2;
  x3 = if (xs>0.5) then -10*x1 else 10*x1;
  xs = 1 + sin(0.1*time)-x2-x1;
end DAE2;
```

For this DAE, the Modelica compiler automatically extracts $(x_s - 0.5)$ as the discontinuity or zero-crossing function and assigns a Mode variable during the generation of the C program. `If-then-else` constructs can also be used to define multi-model DAEs. For example, for the following multi-model DAE

$$\text{if } (x \geq 0) \text{ then } \begin{cases} 0 = \dot{x}^3 - xy\dot{x} - x + y^2 + 1 \\ 0 = \dot{y}\dot{x} + yx - x \end{cases}$$

$$\text{else } \begin{cases} 0 = \dot{y} - 2y\dot{x} + y + x^2 - 1 \\ 0 = 5\dot{x} + 2 - 2yx + \dot{y}x + \sin(t) \end{cases}$$

we can write this Modelica code:

```
0=if (x>=0) then der(x)^3-x*y*der(x)-x+y*y+1
  else der(y)-2*y*der(x)+y*x*x-1;

0=if (x>=0) then der(y)*der(x)+y*x-x
  else der(x)*5+2-2*y*x+der(y)*x+sin(time);
```

⁴In the current version of the Modelica compiler of Scicos only index-1 DAEs are accepted

By default, the Modelica compiler of Scicos associate Mode variables with discontinuity points. When a discontinuity does not need any special treatments, the compiler should be informed with `noEvent()` operator. For example, for DAE (4) we write

```
der(x)=-y*x-x+y;
y=if noEvent(x>=0) then -1+sqrt(x)
  else -1+sqrt(-x);
```

In this case, the Modelica compiler does not consider the condition as a zero-crossing functions and during the simulation, the solver does not stop at $x=0$. For $(x>0)$, \sqrt{x} is evaluated and for $(x<0)$, $\sqrt{-x}$ is used. In general, `noEvent()` performs two things: First, it inhibits the solver to probe for solutions beyond the limit of validity. Then, it prevents the solver from halting the integration and doing an unnecessary restart.

Modelica can also be used for mixed continuous-time and discrete equations. For the discrete-time parts, the synchronous data flow principle with the single assignment rule is used. Discrete event and discrete-time models are supported by `when` statements. The equations in a `when` clause are conditionally activated at instants (called event) where the `when` condition becomes true. Here is an example to show the way a discrete-time equation is written in Modelica. The difference equation should be updated whenever x crossed zero with a positive to negative direction, *i.e.*,

$$\begin{cases} \ddot{x} = -4x \\ \text{update } z(k+1) = 0.9 z(k) - 0.2 \end{cases}$$

we can write the following modelica program.

```
class Sine
  Real x(start=1), y(start=0);
  discrete Real z(start=3), z1(start=-1);
equation
  der(x)=y;
  der(y)=-4*x;
  when (x<0) then
    z=0.9*z-0.2;
  end when;
end Sine;
```

The Modelica compiler deduces the direction of the zero-crossing from the condition $(x < 0)$. Because this condition becomes true when the x becomes negative. So far, Scicos provides a minimum support for Modelica discrete models. That is because the discrete time models can be modeled in the Scicos environment. It is however envisaged to improve Modelica compiler of Scicos to support Modelica discrete models.

In `when` clauses, continuous-time variable can also be initialized. A special operator `reinit(state,`

NewValue) can be used to assign new values to the continuous states of a model at an event time. `reinit` can only be employed in the body of a when-clause. As an example, consider the bouncing ball system. Whenever the ball hits ground, *i.e.*, its height becomes negative, the velocity changes sign and dampens down. Here is a Modelica code for this hybrid system.

```
class Bounce
  Real y(start=10), v(start=0);
equation
  der(y) = v;
  der(v) = -9.8;
  when y < 0 then
    reinit(v, -0.9*v);
  end when;
end Bounce;
```

6 Simulation of Modelica programs

Modelica is a language that provides an environment to express the differential and algebraic equations. Note however that the main goal is simulating the models. In order to simulate Modelica models, they should be transformed into a causal program. For that, the Modelica models should be compiled. There are several Modelica compilers such as Dymola⁵ and Open-Modelica⁶. Scicos has its own Modelica compiler called `Modelicac` (acronym of "Modelica compiler") for a subset of the Modelica language. `Modelicac` is an external tool, *i.e.*, it is independent of Scilab. By default, `Modelicac` comes with a module that generates a C code for Scicos blocks. However, since `Modelicac` is free and open source, it is possible to develop code generators for other targets as well.

A Modelica program is associated with a Scicos block. A Scicos block whose behavior is written in Modelica is called an implicit block [10]. With the associated implicit block the input/output variables of the Modelica program can be defined or visualized. This block may be connected to other blocks to build a bigger model, see for example Figure 3 in which a simple electrical circuit has been built with implicit blocks and some output variables are visualized with ordinary or explicit blocks..

The Modelica compiler uses the input and output variables to establish a causality between the variables in the Modelica program. In the next stage, the compiler

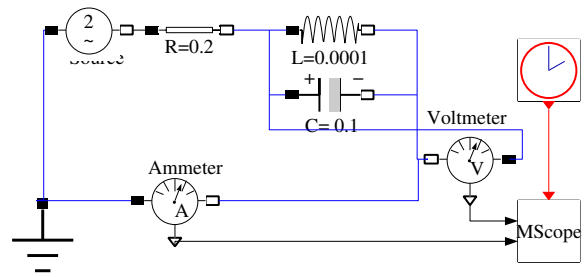


Figure 3: A Modelica block for `Oscillator3.mo`

simplifies the equations and eliminates the unnecessary variables if possible. In the final stage a C program that has the input/output behavior of the Modelica program is generated [13]. Most of the time, the simplification and elimination of variables reduces the size of DAE that consequently reduces the integration time. In addition, a semi-explicit DAE form may be obtained that simplifies the numerical integration [12]. In order to demonstrate the simulation of a complete example, consider this Modelica program

```
class Oscillator3 "Oscillator model"
  Real x(start = 1), y(start=2), u;
equation
  der(x) = x-x*y;
  der(y) = x*y-u*y;
end Oscillator3;
```

where `u` is unknown and is defined by user or another block. To simulate `Oscillator3`, we use a Modelica block (see Fig. 4). In the dialog box of the block (see Fig. 5) an input variable `u` and two output variables `x, y` are defined. After clicking on OK, another window lets the user write the program (see Figure 6).

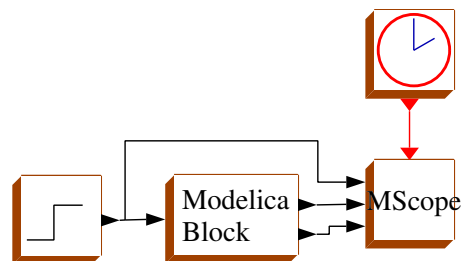


Figure 4: A Modelica block for `Oscillator3.mo`

When the program is compiled, a C program is generated. Here is a fragment of the generated code.

⁵www.dymola.com

⁶www.modelica.org

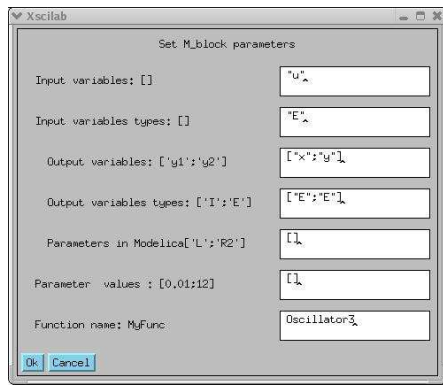


Figure 5: Defining the Modelica program input/output variables

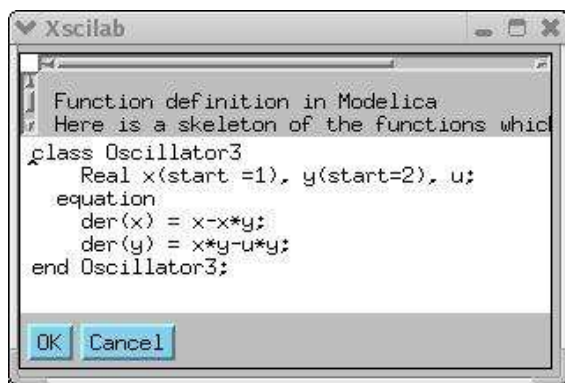


Figure 6: Modelica program in a Modelica block

```

if (flag == 0) { // generated DAE code
  res[0] = xd[0]+x[0]*x[1]-x[0];
  res[1] = u[0]*x[1]+xd[1]-x[0]*x[1];
}else if (flag == 1) { //output update
  y[0][0] = x[0];
  y[1][0] = x[1];
}else if (flag == 4) { // initial values
  x[0] = 1.0;
  x[1] = 2.0;
}
    
```

The simulation result is given in Figure 7.

As another example, consider the DAE (5) whose block diagram implementation is depicted in Figure 2. In this case, the Modelica block has only three outputs, see Figure 8-10. The simulation result is given in Figure 11.

7 Conclusion

Modeling and simulation of DAEs via block diagram approach has several shortcomings. Scicos which is a simulation software based on block approach has recently been developed to provide another approach for

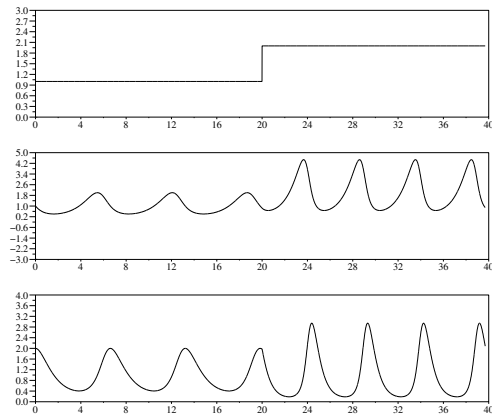


Figure 7: Simulation results for the Scicos diagram in Figure 4

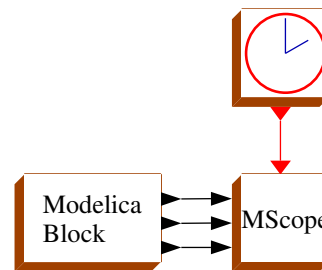


Figure 8: A Modelica block for the DAE (5)

modeling DAEs, *i.e.*, using the Modelica language. In this paper, with some examples we explained the way hybrid DAEs are simulated in Scicos. The Modelica language and its use in Scicos in modeling and simulation of hybrid DAEs were explained. In the last part of the paper, a simple Scicos block is introduced to write and simulate Modelica programs in Scicos.

References

- [1] BRENNAN, K. E., CAMPBELL, S. L., AND PETZOLD, L. R. Numerical solution of initial-value problems in differential-algebraic equations. *SIAM pubs., Philadelphia* (1996).
- [2] BROWN, P. N., HINDMARSH, A. C., AND PETZOLD, L. R. Consistent initial condition calculation for differential-algebraic systems. *SIAM Journal on Scientific Computing* 19, 5 (1998), 1495–1512.
- [3] CAMPBELL, S. L. Numerical methods for unstructured higher index daes. *Annals of Numeri-*

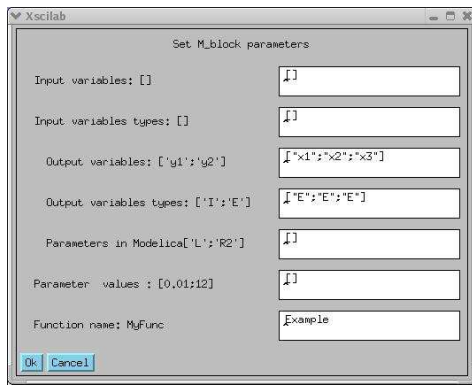


Figure 9: Defining the Modelica program input/output variables

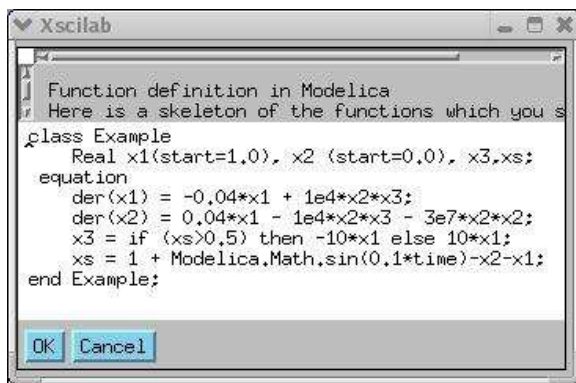


Figure 10: Modelica program in a Modelica block

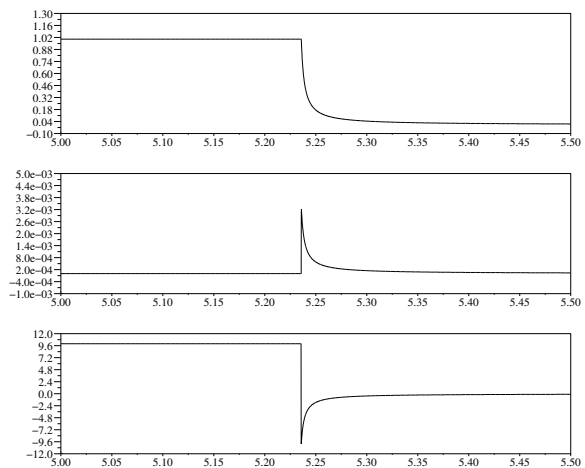


Figure 11: Simulation result for the diagram in Figure 4

[4] CAMPBELL, S. L., CHANCELIER, J.-P., AND NIKOUKHAH, R. *Modeling and simulation Scilab/Scicos*, 1st ed. Springer Verlag, 2005.

[5] CAMPBELL, S. L., MOORE, E., NAKASHIGE, R., ZHONG, Y., AND ZOCHLING, R. Constraint preserving integrators for unstructured higher index daes. *Zeitschrift fuer Angewandte Mathematik und Mechanik (ZAMM)* 76 (1996), 83–86.

[6] CHANCELIER, J. P., DELEBECQUE, F., GOMEZ, C., GOURSAT, M., NIKOUKHAH, R., AND STEER, S. *An introduction to Scilab*, 1st ed. Springer Verlag, Le Chesnay, France, 2002.

[7] GEAR, C. W. Differential-algebraic equation index transformations. *SIAM. J. Sci. Stat. Comp.* 9 (1988), 39–47.

[8] HINDMARSH, A. C. Lsode and lsodi, two new initial value ordinary differential equation

solvers. *ACM-Signum Newsletter* 15 (1980), 10–11.

[9] NAJAFI, M., AZIL, A., AND NIKOUKHAH, R. Implementation of continuous-time dynamics in scicos. *15th ESS Conference, Delft, the Netherlands* (October 2003).

[10] NAJAFI, M., AZIL, A., AND NIKOUKHAH, R. Extending scicos from system to component level simulation. *ESMC2004 international conference, Paris, France* (October 2004).

[11] NAJAFI, M., AND NIKOUKHAH, R. The use of the numerical integrator in scicos, a user friendly graphical based simulation software. *European Journal of Automation (JESA) Special issue on modeling, formalism, methods and simulation tools* (2006), 95–111.

[12] NAJAFI, M., NIKOUKHAH, R., AND CAMPBELL, S. L. The role of model formulation in dae integration: Experience gained in developing scicos. *17th IMACS World Congress Mathematics and Computers in Simulation, Paris, France* (July 2005).

[13] NAJAFI, M., NIKOUKHAH, R., STEER, S., AND FURIC, S. New features and new challenges in modeling and simulation in scicos. *IEEE conference on control application, Toronto, Canada* (2005).

[14] NIKOUKHAH, R., AND STEER, S. Hybrid systems: modeling and simulation. In *COSY: Math-*

ematical Modelling of Complex System, Lund, Sweden (September 1996).

- [15] NIKOUKHAH, R., AND STEER, S. Scicos: A dynamic system builder and simulator, user's guide - version 1.0. Tech. Rep. RT-0207, INRIA Technical Report, Le Chesnay, France, June 1997.
- [16] PETZOLD, L. R. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM J. Sci. Stat. Comput* 4 (1983).
- [17] PETZOLD, L. R. Order results for implicit runge-kutta method applied to differential algebraic systems. *SIAM. J. Numer. Anal.* 23 (1986), 837–852.
- [18] SHAMPINE, L., REICHEL, M. W., AND KIERZENKA, J. A. Solving index-1 DAEs in MATLAB and Simulink. *j-SIAM-REVIEW* 41, 3 (1999), 538–552.

How to dissolve complex dynamic systems for wanted unknowns with *Dymola / Modelica*

Jochen Köhler

ZF Friedrichshafen AG

Graf-von-Soden-Platz 1, D-88046 Friedrichshafen, Germany

Abstract

For developing optimized hybrid driveline strategies a way was found to make complex models of these systems available in an optimization process. The challenge here is to make models built up with *Modelica* available for an optimization process in other tools like *Matlab*. The simulation feature of *Dymola* does not help here because the optimization is done in one point in time.

After describing the optimization problem for these drivelines the special way of implementing it in *Modelica* is emphasized.

When the model is built it had to be exported into *Matlab* as a MEX-funtion to make it available for optimization algorithms.

Keywords: Dissolving equations; C-Interface; dsblock; Matlab Mex File

1 Introduction

One main issue in control theory is to drive a system in an optimal way. One task is to find an optimal trajectory to get to a final desired state that is known. If this final state is not known or if there is no “final” another way is to optimize just the present working point of the system. These optimal working points for different working conditions can be found by an optimization process. To be able to optimize it you have to describe the model mathematically. In principle, *Modelica* is a very good language to describe complex systems in a mathematical form and *Dymola* is a good and easy to use solver for it. With *Modelica* you can build up models very quickly and it’s easy to modify these models.

But the normal way to use *Modelica* is to build models of complex systems to simulate them in time. This does not help here, because we need to dissolve such systems at one point in time for wanted unknowns under the assumption that certain values are known. In this case, it is no simulation issue any-

more! This is not only useful for optimizing working points of dynamical systems (e.g. minimum consumption at a specified output power) but also for finding extreme working points of it (e.g. maximum possible Output torque of a driveline). It would be nice to benefit from *Modelica* also for this type of task.

In principle, *Dymola* does all the critical work to solve this problem: It manipulates the originally given equations to make the numeric system dissolvable and generates C-Code, but there is no standard way to make it useable for the described task.

The following approach to do this is elucidated on the basis of optimizing the operation of hybrid drivelines.

2 Optimizing the work point of a hybrid driveline

The prior task of a hybrid driveline is to save fuel and minimize emissions. Furthermore there are many other conditions that have to be taken into account

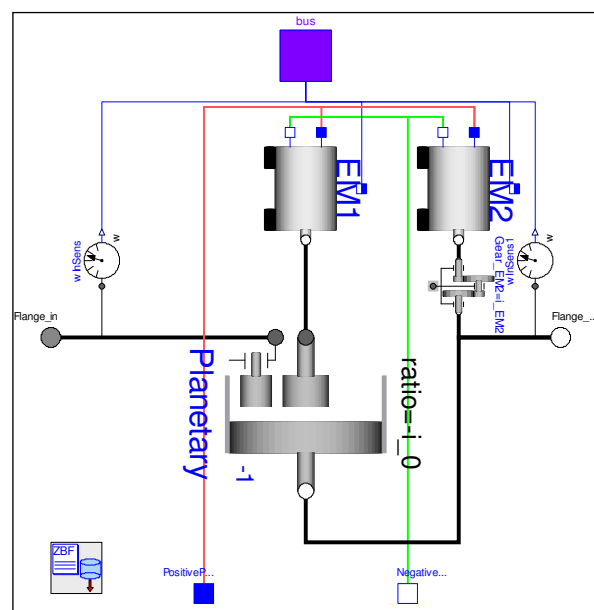


Figure 1: Model of a “Prius” driveline

like State of charge of the battery, performance demands and comfort issues. To get the “best compromise” of these goals you need a driving strategy that operates the hybrid system always on an appropriate working point.

2.1 Task of a hybrid driving strategy

A driving strategy must interpret the driver command given by the accelerator α and brake pedal β as a demanded torque \hat{T}_{Out} at the actual driving speed ω_{Out} .

$$\hat{T}_{Out} = \hat{T}_{Out}(\alpha, \beta) \Big|_{\omega_{Out}}$$

Eq. 1

Dependant on the driveline topology there are many possibilities to perform this torque. Taking the “Toyota Prius system” (Figure 1) as an example of a power split transmission, this torque can be delivered as a combination of torques from the internal combustion engine (ICE), and both electric machines. You could define the work point with some heuristic rules in the driving strategy but then it’s very difficult to get the over all optimum of the whole system. Another approach is to get this optimum using an optimizing process. Therefore you need all the torques and speeds of the ICE and all EM’s.

When desired speed $\hat{\omega}_{ICE}$, acceleration $\hat{\dot{\omega}}_{ICE}$ and torque \hat{T}_{ICE} of the ICE, furthermore the desired acceleration of the vehicle, represented as angular acceleration at the gearbox output $\hat{\dot{\omega}}_{Out}$, are selected as degrees of freedom, a well-defined system can be dissolved to this form.

$$\begin{aligned} & (T_{EM1}, \omega_{EM1}, T_{EM2}, \omega_{EM2}) = \\ & \Phi(\hat{T}_{Out}, \omega_{Out}, \hat{\dot{\omega}}_{Out}, \hat{T}_{ICE}, \hat{\omega}_{ICE}, \hat{\dot{\omega}}_{ICE}) \end{aligned}$$

Eq. 2

You don’t have to describe the system in steady-state! However by setting the accelerations to zero, you get this special case.

This equation can be used to get the torques, speeds (and accelerations) of the EM’s and in consequence

the actual consumption of the ICE and the current through the battery.

2.2 Components that effect a performance index

To be able to optimize the hybrid system a performance index G (Eq. 3) has to be defined. In the end it depends on the actual consumption C of the ICE and the current I through the battery that in turn depend on the demanded torques and speeds given from the driving strategy.

$$G(C, I) = G(\hat{T}_{Out}, \omega_{Out}, \hat{\dot{\omega}}_{Out}, \hat{T}_{ICE}, \hat{\omega}_{ICE}, \hat{\dot{\omega}}_{ICE})$$

Eq. 3

There are a lot of components that have effect on these values:

- Vehicle with its driving resistance
- ICE
- EM’s
- Battery
- Gears in the driveline

The mathematical description of each component can be rather complex. Just as complex is the interaction between these components. For a *Modelica* user the obvious way to address this problem would be to build a model of the complete driveline using detailed models of the components. It’s the task of the *Modelica* Interpreter / Solver to build a system of nonlinear equations that can be dissolved numerically.

3 Modeling the hybrid driveline for numeric solving

3.1 Defining knowns and unknowns

Having the mentioned components as model component in *Modelica* it is easy to build a model of the complete hybrid driveline. By using the block

`Modelica.Blocks.Math.TwoInputs,`

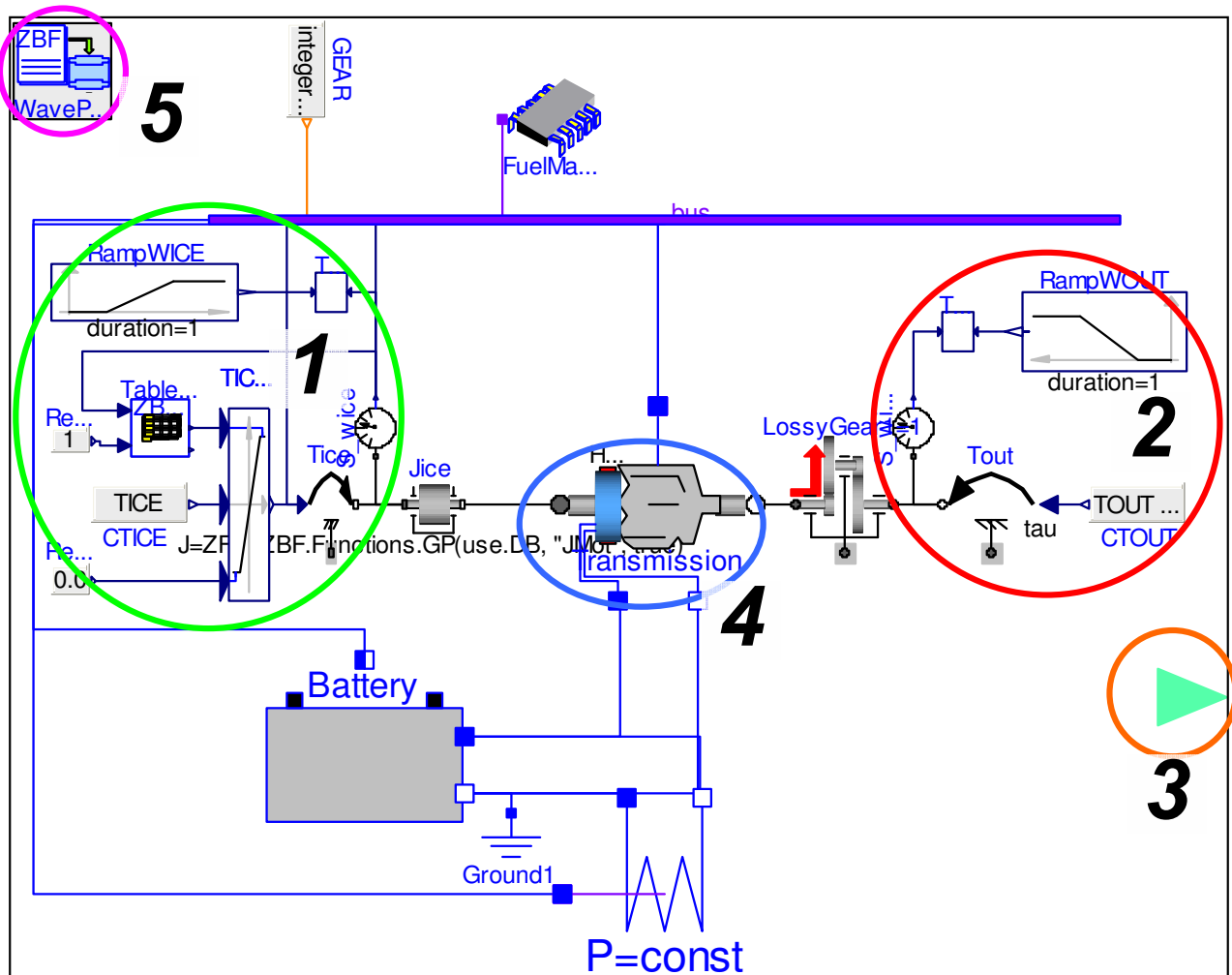


Figure 2: Modelica model of a hybrid driveline with given inputs and out put

The input values from function Φ (Eq. 2) can be defined as inputs to this model. It turned out to be the best way to declare the inputs as parameters in *Modelica*. The block

```
Modelica.Blocks.Sources.Ramp
```

is the best way to define the speeds and accelerations by putting the parameters in here.

In Figure 2 the Ramp for the ICE speed and acceleration can be found in circle No. 1 that is connected to the TwoInputs block. Accordingly the speed of the output shaft is defined in the circle No. 2 of Figure 2. The desired torque of ICE and output are defined in a Modelica expression block.

As output a connector (circle No. 3 in Figure 2) with the wanted unknowns is defined and connected to the appropriate components or bus signals.

The main driveline is represented by the transmission model in circle No. 4 of Figure 2. You can just replace this component by another transmission if you want to. The frame around it just keeps the same.

In this special case, to calculate the unknown values for a certain point in time the initialization feature of *Modelica / Dymola* is used.

3.2 Modifications of components for numeric solving

Depending on the possibly variable structure of the transmission, the number of degrees of freedom can change. This has to be handled in the initial equation block of the complete model. Another possible problem can be “inconsistent speeds” because of any ratios within the driveline or clutches.

The simplest case is a clutch that is known to be stuck. Here it is useful to replace it by a fixed connection.

If there is a clutch used as launching unit in a parallel hybrid, it is a bit more complicated. We have “inconsistent speeds” if the given speeds of the ICE $\hat{\omega}_{ICE}$ and the output ω_{Out} does not match the condition

$$\hat{\omega}_{ICE} = i_{Gear} \cdot \omega_{Out}$$

Eq. 4

where i_{Gear} is the gearbox ratio. In this case the launching clutch has to be in a slipping state. Otherwise it has to be stuck. With the standard clutch model in the Modelica standard library, Dymola is not able to solve the problem. Introducing a “special clutch” just as a torque element, the system is solvable for Dymola (See Figure 3).

Doing it this way, Dymola calculates the needed torque `f_normalized` to satisfy the given inputs.

```

model Clutch_OutT extends
  Modelica.Mechanics.Rotational
    .Interfaces.Compliant;
public
  Modelica.Blocks.Interfaces.RealInput
    f_normalized;
equation
  tau = f_normalized;
end Clutch_OutT;

```

Figure 3: Modelica Code of specialized clutch

Another point to be taken care of are possible delays in form of `FirstOrder` or `SecondOrder` blocks that are inserted in the model to make its behaviour smooth. In normal simulation mode they start normally at a zero initial state and get rather fast to their “steady state”, but if you want to get this “steady” result just at initialization of the model you have to think of eliminating these blocks or to define some extra initial equations to avoid this problem.

4 Generate the numeric solver

After doing the steps described above, a *Modelica* model exists, that can be used to calculate the wanted unknowns with a defined input. You can perform this calculation within *Dymola* interactively by just starting the model for a short period of time and look for the results at initial time. But if you want to optimize working points you have to do this very often. In addition to this you should be able to make this calculation available for an optimizing algorithm as they are implemented for example in *Matlab*. Therefore the *Modelica* model built above has to be exported into an appropriate environment.

4.1 MEX-function generation in *Matlab*

Dymola provides an export to *Matlab Simulink* in form of a so called S-Function. To do this it embeds the C-Code generated during the translation process in the S-Function interface of *Simulink*. But this does not help her because the model shall not be simulated but be called for one point in time with certain inputs. To provide this functionality one reasonable way is to import the generated C-Code into *Matlab* as a so called MEX-function. Instead of the S-Function interface the MEX gateway function

```

void mexFunction(int nlhs, mxArray
  *plhs[],
  int nrhs, const
  mxArray *prhs[])

```

has to be used. To dissolve the hybrid driveline, the code generated from *Dymola* (`dsmodel.c`) is called within this gateway function. In the generated code a function

```

dsblock(&idemand, ..., inputs, para-
  meters, ..., outputs)

```

is implemented that contains all system equations. The function `dsblock` can perform different tasks like initialization, calculating derivatives, handling events and so on. For this task, only the initialization functionality is needed. Here’s the calling sequence:

1. Set `inputs` and `parameters` of the model from `prhs` of mex function.
2. Call initialization of `dsblock`
3. Put the wanted values from `outputs` of `dsblock` into `plhs` of mex function.

You can compile this MEX-gateway function with the MEX-compiler and get a DLL that can be called from the *Matlab* command line or in a *Matlab* M-Script as any other *Matlab* function. In this form you can use it for example within a fitting function for the *Matlab Optimization Toolbox*.

4.2 Parametrizing the function

For one kind of driveline the mex function should be built only once. As a consequence the parameters must not be defined within the model. Therefore all application dependable parameters are defined in ASCII files that can be read by the model itself during initialization. This is described in [1]. When including the generated code from *Dymola*, we have to insert a string variable to define the name of the ASCII file that shall be read by the model, because there is no possibility to define this as a variable within *Modelica*. This string variable is an input variable of the MEX-function and is propagated to the model.

5 Conclusions

The main goal to make the efforts described above is to get optimized driving strategies for many variants of hybrid drivelines in a short period of time.

Modeling complex systems in *Modelica / Dymola* is easy to do and fast. So this is the most efficient way to describe them and make them available for optimization issues. A lot of hybrid drivelines could be provided to optimization process this way. The final results when using the optimized working point in a simulation are very good.

The possibility to dissolve such systems not “only” for time simulation is a great enrichment in many other engineering tasks apart from optimizing hybrid driving strategies.

References

- [1] Köhler J., Banerjee, A. Usage of Modelica in modeling transmissions in ZF, ZF Friedrichshafen AG, 2005.

Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel

Filip H.A. Claeys
Department of Applied Mathematics,
Biometrics and Process Control (BIOMATH)
Ghent University
Coupure Links 653
B-9000 Gent
Belgium
E-mail: fc@biomath.ugent.be

Peter Fritzson
Programming Environments
Laboratory (PELAB)
Linköping University
Campus Valla
SE-581 83 Linköping
Sweden
E-mail: petfr@ida.liu.se

Peter A. Vanrolleghem
modelEAU
Département de génie civil
Université Laval
Pavillon Pouliot
Québec, G1K 7P4
QC, Canada
E-mail: peter@modeleAU.org

Abstract

Tornado is a software kernel for virtual experimentation on the basis of ODE/DAE models. Recently, a model compiler has been developed that converts flat Modelica code to executable models suitable for use with the Tornado kernel. As a result, a subset of Modelica models can now be used for tasks such as parameter estimation, scenario analysis, Monte Carlo simulation, sensitivity analysis and steady-state analysis. The inherent computational complexity of the virtual experiment types implemented by Tornado can be efficiently handled by the kernel's semi-automated distributed execution capabilities.

Keywords: Model compiler; Virtual experimentation; Tornado; Modelica

1 Introduction

Tornado [1] is an advanced kernel for modelling and virtual experimentation (*i.e.*, any evaluation of a model such as simulation, optimization, scenario analysis, ...) that was recently jointly developed by BIOMATH (Ghent University) and HEMMIS N.V. (Kortrijk, Belgium). Although the kernel is generic in nature, it is mostly adopted in the water quality domain. In water quality research, the biological and/or chemical quality of water in rivers, sewers and wastewater treatment plants (WWTP) is studied. Research in this domain is facilitated by a number of models that have received a formal or *de facto* standardization status. Most notable are River Water Quality Model No.1 (RWQM1) [2] and the Activated Sludge Model (ASM) series [3]. Water quality models typically consist of large sets of non-linear Ordinary Differential Equations (ODE)

and/or Differential-Algebraic Equations (DAE). These equations are mostly well-behaved, although discontinuities occur regularly. The complexity of water quality models is therefore not in the nature of the equations, but in the sheer number. In WWTP, smaller models such as the well-known Benchmark Simulation Model (BSM) [4] consist of approximately 150 derived variables. Larger systems have up to 1,000 derived variables and over 10,000 (partly coupled) parameters. On a typical workstation, a simulation run usually lasts minutes to hours.

The modelling language that has thus far been used in the scope of Tornado is MSL (Model Specification Language) [5]. This language is similar to Modelica [6] in the sense that it is high-level, declarative and object-oriented. In fact, both MSL and Modelica were designed according to the ideas resulting from the 1993 ESPRIT Basic Research Working Group 8467 on "Simulation for the Future: new concepts, tools and applications" [7]. Although similar in nature, MSL lacks some of the readability and expressiveness of Modelica. Therefore, it was decided to work towards inclusion of support for Modelica-based modelling in the Tornado framework.

The most recent result of our efforts to bridge the gap between Modelica and Tornado is a model compiler that converts flat Modelica (*i.e.*, a Modelica model description that does not rely on inheritance nor decomposition) to executable models suitable for use with the Tornado kernel. At the moment, this compiler is a prototype that supports basic functionalities of the Modelica language. However, it does allow for a subset of Modelica models to be used in the context of Tornado. Since solutions already exist that generate flat Modelica from full Modelica (*e.g.*, omc - the OpenModelica Compiler), only the conversion from flat Modelica to

executable model code had to be implemented. The sequel of this paper is structured as follows: Section 2 and Section 3 respectively provide a further introduction to Tornado and its complex virtual experimentation capabilities. Subsequently, Section 4 explains how Modelica models can be used in Tornado. Section 5 discusses two simple Modelica models for which virtual experiments were run with Tornado. Finally, Section 6 contains some conclusions and references to future work.

2 Tornado

The Tornado kernel for modelling and virtual experimentation attempts to offer a compromise between the computational efficiency of custom hard-coded (typically FORTRAN or C) model implementations and the flexibility of less computationally efficient generic tools such as MATLAB. In Tornado, hierarchical models are specified in high-level, declarative, object-oriented modelling languages such as MSL [5] and - since recently - also Modelica. From these high-level specifications, efficient executable code is generated by a model compiler. Using the dynamically-loadable executable models generated by the model compiler, Tornado allows for running a variety of so-called *virtual experiments*. Virtual experiments are the virtual-world counterpart of real-world experiments, similar to the way models relate to real-world systems. A highly simplified conceptual diagram of Tornado is shown in Figure 1.

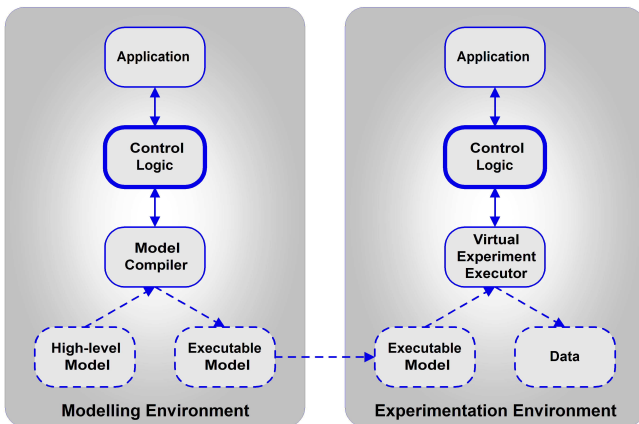


Figure 1: Tornado Conceptual Diagram

The Tornado kernel relies on a flexible input provider and output acceptor mechanism to deal with I/O for virtual experiments. Input can be provided by any combination of data files, internal data buffers and data

generators. Output will be accepted by any combinations of data files, internal data buffers and plot handles (Note: since Tornado is merely a kernel, it does not have any data visualization interface of its own). In order to allow for the kernel to be deployed in a diverse array of applications, it has been equipped with multiple interfaces. Next to its native C++ interface, Tornado currently also has a C, .NET and MATLAB MEX interface (*cf.* Figure 2). The kernel is portable across platforms and was designed according to the three-tier principle. Most persistent representations of information types are XML-based. The grammar of these representations is expressed in XSD (XML Schema Definition) format and mimics very closely the internal representation of the respective types of information. An interesting feature of Tornado is the fact that it allows for dynamic loading of numerical solvers for tasks such as integration, optimization and Latin Hypercube Sampling. In order to support this principle, a generalized framework has been set up [8].

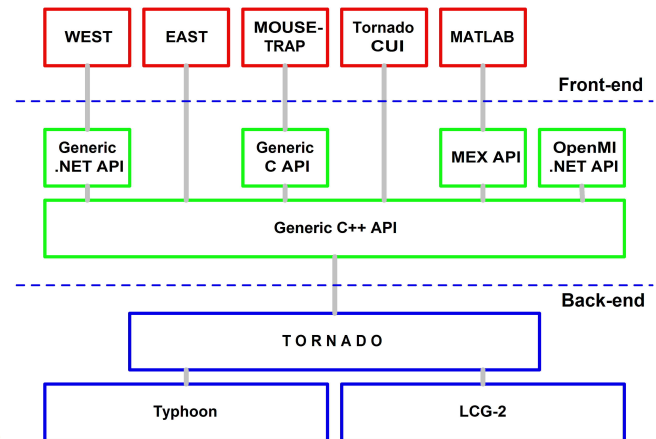


Figure 2: Tornado-based Interfaces and Applications

Several applications (graphical and other) can be built on top of Tornado. Examples include the next generation of the WEST[®] [5] commercial modelling and simulation tool for WWTP's, its research-oriented counterpart named EAST and DHI's MOUSE-TRAP (*cf.*, <http://www.dhigroup.com/Software/Urban.aspx>). However, the most direct way of using the kernel is through the Tornado CUI (Command-line User Interface) suite, which is a comprehensive set of tools that is included with the kernel distribution. Full-fledged graphical applications such as WEST[®] are conceived to be used by all types of users (expert, intermediate, novice). The Tornado CUI suite however focuses on experts only. Table 1 gives an overview of the most commonly used command-line tools. The results dis-

cussed further in this paper were obtained through the Tornado CUI suite.

Table 1: Tornado CUI Suite

Program	Description
<code>tbuild</code>	Compiles and links executable model code to a dynamically-loadable binary object (.dll / .so)
<code>tcreate</code>	Creates an empty XML description of a virtual experiment
<code>texec</code>	Executes virtual experiments described in XML
<code>tinitial</code>	Dumps all model quantity values after initialization
<code>tmsl</code>	Compiles a high-level MSL model to executable model code
<code>tobj</code>	Computes aggregation functions and other criteria from simulation trajectories
<code>tproject</code>	Manages sets of related experiments and connection graphs
<code>tsort</code>	Sorts a Tornado-generated data file
<code>t2msl</code>	Converts a connection graph to MSL code

3 Complex Virtual Experimentation

Tornado consists of strictly separated modelling and virtual experimentation environments. Virtual experiments can either be *atomic* or *compound*. The latter are hierarchically structured whereas the first cannot be further decomposed. Atomic experiment types that are available in Tornado are dynamic simulation and steady-state analysis. The most straightforward types of compound experiments are optimization, scenario analysis, Monte Carlo analysis (e.g. using Latin Hypercube Sampling) and sensitivity analysis. More convoluted types of compound experiments are also available, such as combinations of scenario / Monte Carlo analysis and optimization. Thanks to the object-oriented nature of Tornado, new virtual experiment types can easily be added. Several types of virtual experiments are based on the computation of objective values. As far as possible, the same set of objective types is available for each objective-based experiment type, thereby promoting orthogonality.

Given the hierarchical nature of compound virtual experiments, computational complexity can be substantial. Tornado therefore allows for coarse-grained gridification of certain types of compound virtual experiments. Supported distributed execution environments include BIOMATH's Typhoon cluster software [9] and CERN's LCG-2 grid middleware (cf., [\[egee.org\]\(http://egee.org\)\). Tornado generates generic job descriptions for dynamic execution. Typhoon is capable of directly interpreting these generic job descriptions, whereas for LCG-2, an additional conversion step has to be applied.](http://public.eu-</p>
</div>
<div data-bbox=)

Using Tornado's powerful complex virtual experimentation capabilities, large risk/cost/benefit analyses for integrated water systems were carried out, including Latin Hypercube Sampling from multi-dimensional parameter spaces and the automated execution of 1,000's of simulations [10], each requiring an average of 0.5h of computation time.

4 Using Modelica Models in Tornado

In Tornado, executable models consist of two distinct parts. The first part is represented in C and is made up of the actual model equations, in addition to a number of flat arrays containing data containers for parameters and variables. The second part is a XML representation of meta-information, *i.e.*, information regarding names, descriptions, units, constraints, ... of parameters, variables and models. The relationship between these hierarchically structured meta-information items and the respective elements of the flat C arrays is also expressed in XML. The availability of meta-information in executable models allows for the latter to be self-describing, which is a requirement given the strict separation between modelling and experimentation in Tornado.

In the Tornado framework, model compilers are to generate executable models in the format that was described above. The MSL model compiler that is part of the Tornado suite generates these executable models directly from MSL input. In the case of Modelica however, the approach is two-phased. During the first phase, the OpenModelica Compiler is used to generate flat Modelica (.mof) from full Modelica input (.mo). During the second phase, a new Tornado CUI tool called *mof2t* is used to convert flat Modelica to the Tornado executable model format. This approach was mainly inspired by practical considerations (lack of resources for the re-implementation of the non-trivial full-to-flat Modelica conversion). At the moment *mof2t* only supports a subset of flat Modelica.

For the development of the *mof2t* compiler, the same technologies and libraries were used as for the remainder of the Tornado framework, *i.e.*, C++, flex/bison, and Elcel Technologies OpenTop (cf., <http://www.elcel.com>). The *mof2t* compiler nicely

completes the Tornado CUI suite, which in all consists of approximately 20 tools. The relationship between *mof2t* and the most important other CUI tools is depicted in Figure 3.

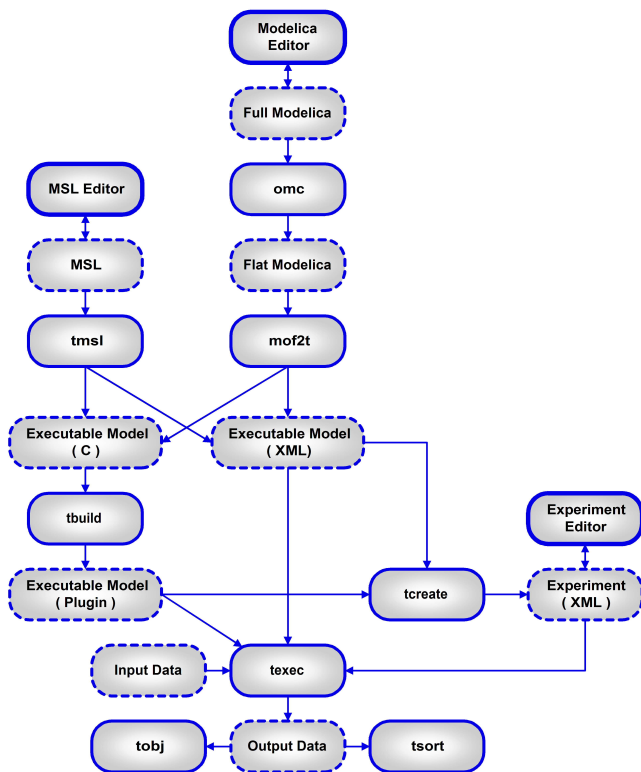


Figure 3: Relationship between the Main Tornado CUI Tools

5 Examples

In this section, two simple cases are presented that illustrate the use of Modelica models in Tornado. The first case is based on the ubiquitous *Van der Pol* system, which is frequently found as an example in modelling and simulation textbooks when stiff systems are discussed. The second case is based on the *ARGESIM - CI* simulator comparison. In both cases, results were obtained using the Tornado CUI suite. Evidently, when using Tornado through one of the GUI applications that it supports, most of the technical details shown below are hidden from the user.

5.1 Van der Pol

5.1.1 Model

The *Van der Pol* system can be described in Modelica as follows:

```

VanDerPol.mof:
---
fclass VanDerPol
Real x(start = 1.0);
Real y(start = 1.0);
parameter Real mu = 1;
equation
  der(x) = y;
  der(y) = -x + mu * (1.0 - x * x) * y;
end VanDerPol;
---
    
```

Given the fact that this model does not rely on inheritance nor decomposition, there is no difference between its full and flattened version. In order to generate executable code for Tornado and convert this code into a dynamically-loadable object, the *mof2t* and *tbuild* CUI tools are to be used:

```

> mof2t VanDerPol.mof
Flat Modelica to Tornado Convertor (Build: Jun 23 2006)

I Loading license spec: Tornado.lic|.Tornado
I Checking MAC address...
I Starting executable model code generation...
I Executable model code generation ended
I Total execution time: 0 seconds

> tbuild -p win32-msvc7.1 VanDerPol
Tornado Model Builder (Build: Jun 23 2006)

I Loading license spec: Tornado.lic|.Tornado
I Checking MAC address...
I Starting build...
I Build ended
I Total execution time: 0 seconds
    
```

The executable C code and XML meta-information that is generated by *mof2t* is as follows:

```

VanDerPol.c:
---
#include <math.h>
#include <stdlib.h>

#include "Tornado/EE/Common/DLL.h"
#include "Tornado/EE/MSLE/MSLE.h"

#define _mu_pModel->Params[0]
#define _time_pModel->IndepVars[0]
#define _x_pModel->DerVars[0]
#define _D_x_pModel->Derivatives[0]
#define _y_pModel->DerVars[1]
#define _D_y_pModel->Derivatives[1]

void ComputeInitial(struct TModel* pModel) {}

void ComputeState(struct TModel* pModel)
{
  _D_x_ = _y_;
  _D_y_ = -_x_ + _mu_ * (1 - _x_ * _x_) * _y_;
}

void ComputeOutput(struct TModel* pModel) {}

void ComputeFinal(struct TModel* pModel) {}

void* GetID() { return (void*)L"Tornado.MSLE.Model.VanDerPol"; }

void* Create()
{
  struct TModel* pModel;

  pModel = (struct TModel*)malloc(sizeof(struct TModel));

  pModel->Type = L"ODE";

  pModel->NoParams = 1;
  pModel->NoIndepVars = 1;
  pModel->NoInputVars = 0;
  pModel->NoOutputVars = 0;
  pModel->NoAlgVars = 0;
  pModel->NoDerVars = 2;
  pModel->NoDerivatives = 2;
  pModel->NoPrevious = 0;
  pModel->NoResidues = 0;
  pModel->NoSolveSets = 0;
  pModel->NoEvents = 0;
}
    
```

```

pModel->Params =
  (double*) malloc(sizeof(double) * pModel->NoParams);
pModel->IndepVars =
  (double*) malloc(sizeof(double) * pModel->NoIndepVars);
pModel->InputVars =
  (double*) malloc(sizeof(double) * pModel->NoInputVars);
pModel->OutputVars =
  (double*) malloc(sizeof(double) * pModel->NoOutputVars);
pModel->AlgVars =
  (double*) malloc(sizeof(double) * pModel->NoAlgVars);
pModel->DerVars =
  (double*) malloc(sizeof(double) * pModel->NoDerVars);
pModel->Derivatives =
  (double*) malloc(sizeof(double) * pModel->NoDerivatives);
pModel->Previous =
  (double*) malloc(sizeof(double) * pModel->NoPrevious);
pModel->Residues =
  (double*) malloc(sizeof(double) * pModel->NoResidues);
pModel->SolveSets =
  (TSolveSetP) malloc(sizeof(struct TSolveSet) *
    pModel->NoSolveSets);
pModel->Events =
  (TEventP) malloc(sizeof(struct TEvent) * pModel->NoEvents);

pModel->ComputeInitial = ComputeInitial;
pModel->ComputeState = ComputeState;
pModel->ComputeOutput = ComputeOutput;
pModel->ComputeFinal = ComputeFinal;

return (void*)pModel;
}
---
```

```

VanDerPol.SymbModel.xml:
---
<Tornado>
  <Model>
    <Exec FileName="VanDerPol"/>
    <Symb>
      <Model Name="">
        <Params>
          <Param Name="mu" DefaultValue="1"/>
        </Params>
        <IndepVars>
          <IndepVar Name="time" DefaultValue="0"/>
        </IndepVars>
        <InputVars>
        </InputVars>
        <OutputVars>
        </OutputVars>
        <AlgVars>
        </AlgVars>
        <DerVars>
          <DerVar Name="x" DefaultValue="1"/>
          <DerVar Name="y" DefaultValue="1"/>
        </DerVars>
        <Models>
        </Models>
      </Model>
    </Symb>
    <Links>
      <Link Name=".mu" ValueType="Params" ValueIndex="0"/>
      <Link Name=".time" ValueType="IndepVars" ValueIndex="0"/>
      <Link Name=".x" ValueType="DerVars" ValueIndex="0"
        DerivativeType="Derivatives" DerivativeIndex="0"/>
      <Link Name=".y" ValueType="DerVars" ValueIndex="1"
        DerivativeType="Derivatives" DerivativeIndex="1"/>
    </Links>
  </Model>
</Tornado>
---
```

The exact semantics of these representations are beyond the scope of this paper and will therefore not be further discussed. Important to note however is that the format of the generated C code has been kept as simple as possible in order to be able to compile the code with as many C compilers as possible. The compilers that have been shown to work so far are Borland C++ 5.5, MS Visual C++ 6.0, 7.1 & 8.0, LCC, INTEL C++ 9.0 and g++.

5.1.2 Simulation

In order to simulate the generated model code, a simulation experiment spec is to be provided to the exper-

iment executor. Specs must conform to the respective XML schemas that have been defined in the scope of Tornado. When using the Tornado CUI suite, empty specs can be generated by the `tcreate` program and must then be further completed manually. Below is a simulation experiment spec for the *Van der Pol* model that was generated by invoking `tcreate -t Simul VanDerPol` and further completed through manual editing:

```

VanDerPol.Simul.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="Simul">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Wed Jun 28 14:58:02 2006"/>
      <Prop Name="Desc" Value="Van der Pol simulation"/>
      <Prop Name="FileName"
        Value="VanDerPol.Simul.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value="" />
    </Props>
    <Simul>
      <Model Name="VanDerPol" CheckBounds="false">
        <Quantities>
          <Quantity Name=".mu" Value="100"/>
          <Quantity Name=".x" Value="2"/>
        </Quantities>
      </Model>
      <Inputs Enabled="false">
      </Inputs>
      <Outputs Enabled="true">
        <Output Name="*Calc*">
          <CalcVars Enabled="false">
          </CalcVars>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
          </Plot>
        </Output>
        <Output Name="File">
          <File Name="VanDerPol.Simul.out.txt" Enabled="true">
            <Props>
              <Prop Name="CommInt" Value="0"/>
              <Prop Name="CommIntType" Value="Linear"/>
              <Prop Name="DecSep" Value="."/>
              <Prop Name="Interpolated" Value="false"/>
              <Prop Name="Precision" Value="8"/>
              <Prop Name="StartTime" Value="-INF"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
          </Output>
        </Output>
      </Outputs>
      <Time>
        <Start Value="0"/>
        <Stop Value="300"/>
      </Time>
      <Solve>
        <Integ Method="CVODE">
          <Props>
            <Prop Name="AbsoluteTolerance" Value="1e-006"/>
            <Prop Name="CVBandLowerBandwidth" Value="0"/>
            <Prop Name="CVBandUpperBandwidth" Value="0"/>
            <Prop Name="CVSPGMRGStype" Value="ModifiedGS"/>
            <Prop Name="IterationMethod" Value="Newton"/>
            <Prop Name="LinearMultistepMethod" Value="BDF"/>
            <Prop Name="LinearSolver" Value="Diag"/>
            <Prop Name="MaxNoSteps" Value="0"/>
            <Prop Name="RelativeTolerance" Value="1e-005"/>
          </Props>
        </Integ>
      </Solve>
    </Simul>
  </Exp>
</Tornado>
---
```

For a simulation experiment, XML specs basically allow for specifying initial values (hereby overruling initializations that were specified through the model's meta-information), defining input providers / output acceptors, specifying the simulation start / stop time and configuring integrator solver settings. In this case, initial values were given for `.mu` and `.x`, input was disabled and one output file acceptor was defined. The simulation will be run from 0 to 300 and the CVODE stiff system solver (*cf.*, <http://www.llnl.gov/CASC/sundials>) will be used as an integrator. Important to note is that the settings of the CVODE integrator are given through a flexible attribute-value pair mechanism instead of through tags that are part of the XML grammar. This is required to support dynamic loading of solver plugins. The fragment below shows the output of the experiment executor, when applied to the *VanDerPol* simulation spec. One will notice that before execution starts, a number of solver plugins are dynamically loaded (in this case only a subset of the 35 solver plugins that are provided with Tornado are loaded):

```
> texec VanDerPol.Simul.Exp.xml

Tornado Experiment Executor (Build: Jun 13 2006)

I Loading license spec: Tornado.lic|.Tornado
I Checking MAC address...
I Loading main spec: Tornado.Main.xml|.Tornado
I Loading plugin: Tornado.Solve.Integ.CVODE
I Loading plugin: Tornado.Solve.Integ.Euler
I Loading plugin: Tornado.Solve.Integ.RK4
I Loading plugin: Tornado.Solve.Integ.RK4ASC
I Loading plugin: Tornado.Solve.Optim.GA
I Loading plugin: Tornado.Solve.Optim.Praxis
I Loading plugin: Tornado.Solve.Optim.SA
I Loading plugin: Tornado.Solve.Optim.Simplex
I Loading plugin: Tornado.Solve.Root.Broyden
I Loading plugin: Tornado.Solve.Root.Hybrid
I Loading plugin: Tornado.Solve.Scen.Cross
I Loading plugin: Tornado.Solve.Scen.Fixed
I Loading plugin: Tornado.Solve.Scen.Grid
I Loading plugin: Tornado.Solve.Scen.Plain
I Loading plugin: Tornado.Solve.Scen.Random
I Loading plugin: Tornado.Solve.Sens.Plain
I Loading plugin: Tornado.Solve.CI.Nelder
I Loading plugin: Tornado.Solve.CI.Richardson
I Loading plugin: Tornado.Solve.MC.IHS
I Loading plugin: Tornado.Solve.MC.CVT
I Loading plugin: Tornado.Solve.MC.LHS
I Loading plugin: Tornado.Solve.MC.PR
I Main information:
I Author = PCFC1\fc
I Date = Thu Oct 13 16:08:06 2005
I Desc = Main spec
I EnableHashOutputHeaders = true
I EnableWESTInputHeaders = true
I EnableWESTOutputHeaders = false
I FileName =
I KernelAuthor = Filip Claeys, Dirk De Pauw
I KernelDesc = Advanced Kernel for Modelling and Virtual Ex...
I KernelVersion = 0.22
I LimitMRE = 0.7
I LimitsRE = 0.0235
I Precision = 8
I New job: VanDerPol.Simul.Exp.xml
I Starting thread...
I Loading experiment spec: VanDerPol.Simul.Exp.xml|.Tornado
I Loading simulation experiment spec: VanDerPol.Simul.Exp.xml...
I Loading symbolic model spec: VanDerPol.SymbModel.xml|.Tornado
I Loading executable model: Tornado.MSLE.Model.VanDerPol
I Executable model information:
I Type = ODE
I #Params = 1
I #IndepVars = 1
I #InputVars = 0
I #OutputVars = 0
I #AlgVars = 0
```

```
I #DerVars = 2
I #Derivatives = 2
I #Previous = 0
I #Residues = 0
I #SolveSets = 0
I #Events = 0
I Building model symbol table...
I Checking model linkage...
I Creating simulator...
I Setting integration solver: Tornado.Solve.Integ.CVODE
I Experiment information:
I Type = Simul
I Embedded = true
I Author = PCFC1\fc
I Date = Wed Jun 28 14:58:02 2006
I Desc = Van der Pol simulation experiment
I FileName = VanDerPol.Simul.Exp.xml|.Tornado
I UnitSystem =
I Initializing model...
I Opening simulation output file: VanDerPol.Simul.out.txt
I Simulation from 0 to 300
I Starting simulation...
I Simulation ended
I Closing simulation output file: VanDerPol.Simul.out.txt
I Executable model statistics:
I #ComputeInitials: 1
I #ComputeStates: 3240
I #ComputeOutputs: 1686
I #ComputeFinals: 1
I Total execution time: 0 seconds
I Thread ended
I Unloading plugins
```

The Tornado CUI suite does not contain any data visualization mechanism, however one can easily use tools such as MS Excel, MATLAB or GNUPlot to display the simulated trajectories. Figure 4 shows the result of invoking the following commands in GNUPlot:

```
set xlabel "t"
set ylabel ".x"
plot 'VanDerPol.Simul.out.txt' using 1:2 with lines
```

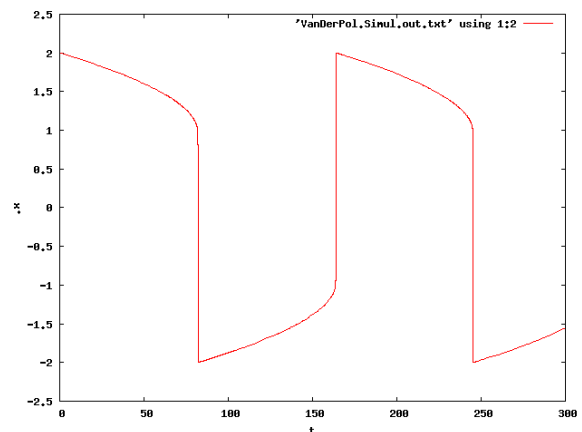


Figure 4: Van Der Pol for $\mu = 100$

5.1.3 Parameter variation

More interesting it becomes if we wish to run the same simulation for different initial values. For instance, suppose we wish to run the simulation for values of `.mu` that are logarithmically spaced between 1 and 100 with a spacing of 2. Suppose also that for each simulation, we want to determine the maximum value and

standard deviation of the trajectory of y , in addition to the value of y at $t = 50$. The scenario analysis experiment shown below provides a solution to this problem:

```

VanDerPol.Scen.Exp.xml :
---
<Tornado>
  <Exp Version="1.0" Type="Scen">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Wed Jun 28 15:37:57 2006"/>
      <Prop Name="Desc" Value="VanDerPol scenario analysis"/>
      <Prop Name="FileName"
        Value="VanDerPol.Scen.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <Scen>
      <Obj>
        <Exp Version="1.0" Type="Simul"
          FileName="VanDerPol.Simul.Exp.xml|.Tornado"/>
        <Props>
          <Prop Name="CommInt" Value="0"/>
          <Prop Name="DecSep" Value="."/>
          <Prop Name="EnableNoComputeStates" Value="false"/>
          <Prop Name="EnableRetrieval" Value="false"/>
          <Prop Name="EnableStorage" Value="true"/>
          <Prop Name="Interpolated" Value="false"/>
          <Prop Name="OutputFileName"
            Value="VanDerPol.Scen.Simul.out.txt.{}/>
          <Prop Name="Precision" Value="8"/>
          <Prop Name="ThousandSep" Value="."/>
          <Prop Name="TyphoonBaseName" Value="Typhoon"/>
        </Props>
        <Quantities>
          <Quantity Name=".y">
            <Props>
              <Prop Name="Criterion" Value="AbsSquared"/>
              <Prop Name="EnableAvg" Value="false"/>
              <Prop Name="EnableDiffMax" Value="false"/>
              <Prop Name="EnableDiffSum" Value="false"/>
              <Prop Name="EnableEndValue" Value="false"/>
              <Prop Name="EnableInt" Value="false"/>
              <Prop Name="EnableMax" Value="true"/>
              <Prop Name="EnableMin" Value="false"/>
              <Prop Name="EnableStdDev" Value="true"/>
              <Prop Name="EnableTIC" Value="false"/>
              <Prop Name="EnableValueOnTime" Value="true"/>
              <Prop Name="Time" Value="50"/>
              <Prop Name="Weighted" Value="false"/>
            </Props>
          </Quantity>
        </Quantities>
      </Obj>
      <Log Name="VanDerPol.Scen.log.txt" Enabled="true">
        <Props>
          </Props>
      </Log>
      <Inputs Enabled="false">
        </Inputs>
      <Outputs Enabled="true">
        <Output Name="*File*">
          <File Name="VanDerPol.Scen.out.txt" Enabled="true">
            <Props>
              <Prop Name="DecSep" Value="."/>
              <Prop Name="Precision" Value="8"/>
            </Props>
          </File>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
            <Props>
              <Prop Name="Info" Value=""/>
            </Props>
          </Plot>
        </Output>
      </Outputs>
      <Vars>
        <Var Name=".mu">
          <Props>
            <Prop Name="DistributionMethod" Value="Logarithmic"/>
            <Prop Name="LowerBound" Value="1"/>
            <Prop Name="NoValues" Value="0"/>
            <Prop Name="RefValue" Value="1"/>
            <Prop Name="Spacing" Value="2"/>
            <Prop Name="StdDev" Value="0"/>
            <Prop Name="UpperBound" Value="100"/>
            <Prop Name="UpperBoundPolicy"
              Value="IncludeUpperBound"/>
            <Prop Name="Values" Value=""/>
          </Props>
        </Var>
      </Vars>
      <Solve>
        <Scen Method="Grid">
          <Props>
            <Prop Name="EnableRef" Value="false"/>
            <Prop Name="Generate" Value="true"/>
          </Props>
        </Scen>
      </Solve>
    </Exp>
  </Tornado>

```

```

    <Prop Name="UseTyphoon" Value="false"/>
  </Props>
</Scen>
</Solve>
</Scen>
</Exp>
</Tornado>

```

As one can see, this scenario analysis spec refers to a simulation spec that resides in an external file (`VanDerPol.Simul.Exp.xml`). Directly embedding the XML content of this file into the scenario analysis experiment is however also possible. One will also notice that other types of objectives and aggregation functions (next to the *Min*, *StdDev*, *ValueOnTime* functions that are needed for our application) such as *Avg* (average) and *Int* (integral) are also possible. Table 2 shows the contents of the `VanDerPol.Scen.out.txt` file that is generated during the execution of the scenario analysis.

Table 2: Results of the `VanDerPol.Scen.Exp.xml` Experiment

RunNo	.mu	Max(.y)	StdDev(.y)	ValueOnTime(.y)
1	1	2.6865596	1.4272097	-1.5137494
2	2	3.8300373	1.4645402	-0.034409599
3	4	6.3463996	1.5316644	0.58337344
4	8	11.553678	1.593544	-0.092195286
5	16	22.097001	1.6317544	0.058550465
6	32	43.305241	1.6404037	0.052380761
7	64	85.828672	1.5529595	-0.0439915
8	100	133.68028	1.4986709	-0.010200556

5.2 ARGESIM - C1

ARGE Simulation News (*cf.*, <http://www.argesim.org>) is a non-profit working group providing the infrastructure and administration for dissemination of information on modelling and simulation in Europe. ARGESIM is located at Vienna University of Technology, Dept. Simulation and publishes Simulation News Europe (SNE), which features a series on comparisons of simulation software. Based on simple, easily comprehensible models special features of modelling and experimentation within simulation languages, also with respect to an application area, are compared. Features are, for instance: modelling technique, event-handling, numerical integration, steady-state calculation, distribution fitting, parameter sweep, output analysis, animation, complex logic strategies, submodels, macros, statistical features *etc.* Approximately 20 comparisons have thusfar been defined, the first was

published in November 1990, the last in December 2005.

5.2.1 Model

As a second example of the use of Modelica models in Tornado, the *C1 ARGESIM* comparison will be used. The model that is at the basis of this comparison can be represented in Modelica as follows:

```
C1.moF:
---
fclass C1
  parameter Real kr = 1;
  parameter Real kf = 0.1;
  parameter Real lf = 1000;
  parameter Real dr = 0.1;
  parameter Real dm = 1;
  parameter Real p = 0;
  Real f(start = 9.975);
  Real m(start = 1.674);
  Real r(start = 84.99);
equation
  der(r) = -dr * r + kr * m * f;
  der(m) = dr * r - dm * m + kf * f * f - kr * m * f;
  der(f) = dr * r + 2 * dm * m - kr * m * f -
    2 * kf * f * f - lf * f + p;
end C1;
---
```

The comparison requires the following tasks to be performed:

- *Simulation of the stiff system over [0,10].*
- *Parameter variation of lf from 1.0e2 to 1.0e4 and a plot of all f(t); lf), logarithmic steps preferred.*
- *Calculation of steady states during constant bombardment ($p(t) = pc = 1.0E4$) and without bombardment ($p(t) = 0$).*

5.2.2 Simulation

As in the first example, a dynamically-loadable executable model for Tornado can be generated using `mof2t` and `tbuild`. Afterwards, an empty simulation experiment can be generated with `tcreate` and then be completed through manual editing:

```
C1.Simul.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="Simul">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 12:28:12 2006"/>
      <Prop Name="Desc" Value="" />
      <Prop Name="FileName"
        Value="C1.CVODE.Simul.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value="" />
    </Props>
    <Simul>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
          </Quantities>
        </Model>
        <Inputs Enabled="false">
          </Inputs>
        <Outputs Enabled="true">
          <Output Name="*Calc*">
            <CalcVars Enabled="false">
              </CalcVars>
            </Output>
          <Output Name="*Plot*">
```

```

      <Plot Enabled="false">
        <Props>
          <Prop Name="CommInt" Value="0"/>
          <Prop Name="Info" Value="" />
          <Prop Name="Interpolated" Value="false"/>
          <Prop Name="StartTime" Value="-INF"/>
          <Prop Name="StopTime" Value="+INF"/>
          <Prop Name="UseDisplayUnits" Value="true"/>
        </Props>
        <Quantities>
          </Quantities>
        </Plot>
      </Output>
      <Output Name="File">
        <File Name="C1.Simul.out.txt" Enabled="true">
          <Props>
            <Prop Name="CommInt" Value="1.2"/>
            <Prop Name="CommIntType" Value="Logarithmic"/>
            <Prop Name="DecSep" Value="" />
            <Prop Name="Interpolated" Value="true"/>
            <Prop Name="Precision" Value="8"/>
            <Prop Name="StartTime" Value="1e-007"/>
            <Prop Name="StopTime" Value="+INF"/>
            <Prop Name="UseDisplayUnits" Value="true"/>
          </Props>
          <Quantities>
            <Quantity Name=".f"/>
            <Quantity Name=".m"/>
            <Quantity Name=".r"/>
          </Quantities>
        </File>
      </Output>
    </Outputs>
    <Time>
      <Start Value="0"/>
      <Stop Value="10"/>
    </Time>
    <Solve>
      <Integ Method="CVODE">
        <Props>
          <Prop Name="AbsoluteTolerance" Value="1e-006"/>
          <Prop Name="CVBandLowerBandwidth" Value="0"/>
          <Prop Name="CVBandUpperBandwidth" Value="0"/>
          <Prop Name="CVSPGMRGStype" Value="ModifiedGS"/>
          <Prop Name="IterationMethod" Value="Functional"/>
          <Prop Name="LinearMultistepMethod" Value="Adams"/>
          <Prop Name="LinearSolver" Value="Dense"/>
          <Prop Name="MaxNoSteps" Value="0"/>
          <Prop Name="RelativeTolerance" Value="1e-006"/>
        </Props>
      </Integ>
      <Root Method="Broyden">
        <Props>
          <Prop Name="MaxNoSteps" Value="0"/>
          <Prop Name="MaxStepSize" Value="1"/>
        </Props>
      </Root>
    </Solve>
  </Simul>
</Exp>
</Tornado>
---
```

Important to notice in this simulation experiment is that for the output file acceptor, the communication interval type (*CommIntType*) was set to logarithmic. In this case, logarithmic spacing of output timepoints is required in order to be able to accurately represent the dynamics of the simulated trajectories during the initial phase of the simulation (without generating huge amounts of irrelevant data). After running the simulation with `texec`, one could for instance use GNUPlot to display the results (see Figure 5) onto logarithmic axes using the following commands:

```
set logscale xy
set xlabel "t"
set ylabel ".f"
plot 'C1.Simul.out.txt' using 1:2 with lines
```

5.2.3 Parameter variation

The parameter variation that is requested by the comparison can easily be implemented in Tornado us-

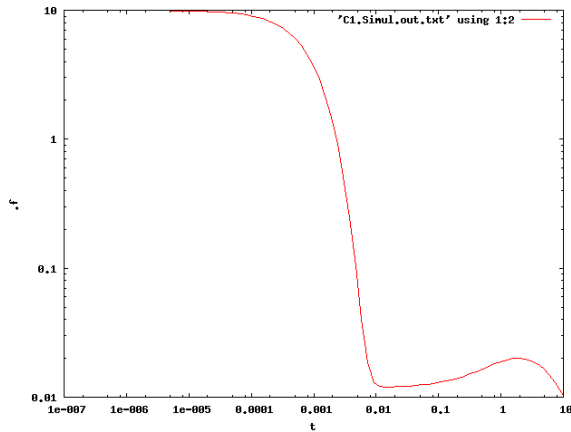


Figure 5: Simulation results for the ARGESIM C1 model

ing the scenario analysis experiment type. However, in contrast to the *Van der Pol* example, no post-processing functions (such as *Min*, *StdDev*, ...) are needed in this case. Important however is that the variation of *lf* is to be set to *Logarithmic*, as requested. Figure 6 shows the results of a 10-shot scenario analysis experiment defined in this way.

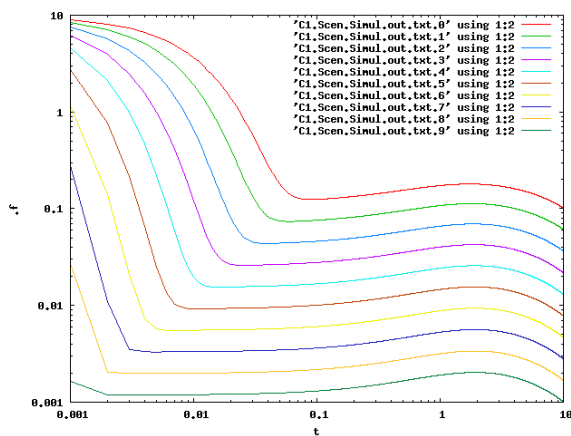


Figure 6: Scenario analysis results for the ARGESIM C1 model

5.2.4 Calculation of steady states

For the calculation of steady states, the steady-state (SS) experiment type can be used. In Tornado, the steady-state of a system is directly computed through the application of a root finding solver to the system equations, where the derivatives (*i.e.*, the left hand sides) of state equations are used as residues (that are to be brought to zero).

The following describes a steady-state experiment for the ARGESIM C1 model where $p = 1e4$:

```
C1.p=1e4.SS.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="SS">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 15:39:27 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName" Value="C1.p=1e4.SS.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <SS>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
          <Quantity Name=".p" Value="1e4"/>
        </Quantities>
      </Model>
      <Solve>
        <Root Method="Hybrid">
          <Props>
            <Prop Name="Tolerance" Value="1e-008"/>
          </Props>
        </Root>
      </Solve>
    </SS>
  </Exp>
</Tornado>
---
```

Execution of this experiment with `texec` will instantly yield the correct steady state values for *f*, *m* and *r* :

```
> texec "C1.p=1e4.SS.Exp.xml"

Tornado Experiment Executor (Build: Jun 13 2006, 10:32:40)

I Loading license spec: Tornado.lic|.Tornado
I Checking MAC address...
I Loading main spec: Tornado.Main.xml|.Tornado
I Loading plugin: Tornado.Solve.Integ.CVODE
I Loading plugin: Tornado.Solve.Integ.Euler
I Loading plugin: Tornado.Solve.Integ.RK4
I Loading plugin: Tornado.Solve.Integ.RK4ASC
I Loading plugin: Tornado.Solve.Optim.GA
I Loading plugin: Tornado.Solve.Optim.Praxis
I Loading plugin: Tornado.Solve.Optim.SA
I Loading plugin: Tornado.Solve.Optim.Simplex
I Loading plugin: Tornado.Solve.Root.Broyden
I Loading plugin: Tornado.Solve.Root.Hybrid
I Loading plugin: Tornado.Solve.Scen.Cross
I Loading plugin: Tornado.Solve.Scen.Fixed
I Loading plugin: Tornado.Solve.Scen.Grid
I Loading plugin: Tornado.Solve.Scen.Plain
I Loading plugin: Tornado.Solve.Scen.Random
I Loading plugin: Tornado.Solve.Sens.Plain
I Loading plugin: Tornado.Solve.CI.Nelder
I Loading plugin: Tornado.Solve.CI.Richardson
I Loading plugin: Tornado.Solve.MC.IHS
I Loading plugin: Tornado.Solve.MC.CVT
I Loading plugin: Tornado.Solve.MC.LHS
I Loading plugin: Tornado.Solve.MC.PR
I Main information:
I   Author = PCFC1\fc
I   Date = Thu Oct 13 16:08:06 2005
I   Desc = Main spec
I   EnableHashOutputHeaders = true
I   EnableWESTInputHeaders = true
I   EnableWESTOutputHeaders = false
I   FileName =
I   KernelAuthor = Filip Claeys, Dirk De Pauw
I   KernelDesc = Advanced Kernel for Modelling and Virtual...
I   KernelVersion = 0.22
I   LimitMRE = 0.7
I   LimitSRE = 0.0235
I   Precision = 8
I New job: C1.p=1e4.SS.Exp.xml
I Starting thread...
I Loading experiment spec: C1.p=1e4.SS.Exp.xml|.Tornado
I Loading steady-state analysis experiment spec: C1.p=1e4...
I Loading symbolic model spec: C1.SymbModel.xml|.Tornado
I Loading executable model: Tornado.MSLE.Model.C1
I Executable model information:
I   Type = ODE
I   #Params = 6
I   #IndepVars = 1
I   #InputVars = 0
I   #OutputVars = 0
I   #AlgVars = 0
I   #DerVars = 3
I   #Derivatives = 3
```

```

I #Previous = 0
I #Residues = 0
I #SolveSets = 0
I #Events = 0
I Building model symbol table...
I Checking model linkage...
I Creating steady-state analyser...
I Setting root solver: Tornado.Solve.Root.Hybrid
I Experiment information:
I Type = SS
I Embedded = true
I Author = PCFC1\fc
I Date = Fri Jun 30 15:39:27 2006
I Desc =
I FileName = C1.p=1e4.SS.Exp.xml|.Tornado
I UnitSystem =
I Initializing model...
I Initializing model...
I Starting steady-state analysis...
I Steady-state analysis ended
I Executable model statistics:
I #ComputeInitials: 8
I #ComputeStates: 8
I #ComputeOutputs: 0
I #ComputeFinals: 0
I Final variable values:
I .f = 10
I .m = 10
I .r = 1000
I Total execution time: 0 seconds
I Thread ended
I Unloading plugins

```

For $p = 0$, one can proceed in a similar way. However, in this case the process is more sensitive to the initial value of the state variables. The experiment below therefore shows that for f , a differing initial value had to be chosen to ensure convergence of the algorithm.

```

C1.p=0.SS.Exp.xml|.Tornado:
---
<Tornado>
  <Exp Version="1.0" Type="SS">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 15:39:20 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName" Value="C1.p=0.SS.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <SS>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
          <Quantity Name=".f" Value="0.1"/>
        </Quantities>
      </Model>
      <Solve>
        <Root Method="Hybrid">
          <Props>
            <Prop Name="Tolerance" Value="1e-008"/>
          </Props>
        </Root>
      </Solve>
    </SS>
  </Exp>
</Tornado>
---
...
I Fri Jun 30 15:59:44 2006 Final variable values:
I Fri Jun 30 15:59:44 2006 .f = 0
I Fri Jun 30 15:59:44 2006 .m = 0
I Fri Jun 30 15:59:44 2006 .r = -2.47032822920623e-323
I Fri Jun 30 15:59:44 2006 Total execution time: 0 seconds
...

```

6 Conclusions and Future Work

Through the development of the *mof2t* compiler, Tornado's powerful complex virtual experimentation capabilities have become available for a subset of Modelica models. To facilitate maintenance and further integration, *mof2t* was implemented using the same technologies as the remainder of the Tornado framework.

In the forthcoming months, the *mof2t* will be further stabilized and enhanced.

Acknowledgement

Peter A. Vanrolleghem is Canadian Research Chair in Water Quality Modelling.

References

- [1] F. Claeys, D. De Pauw, L. Benedetti, I. Nopens, and P.A. Vanrolleghem. Tornado: A versatile efficient modelling & virtual experimentation kernel for water quality systems. In *Proceedings of the iEMSs 2006 Conference*, Burlington, VT, 2006.
- [2] P. Reichert, Borchardt D., Henze M., Rauch W., Shanahan P., Somlyódy L., and P.A. Vanrolleghem. *River Water Quality Model No.1*. Scientific and Technical Report No.12. IWA Publishing, London, UK, 2001.
- [3] M. Henze, W. Gujer, T. Mino, and M. van Loosdrecht. *Activated Sludge Models ASM1, ASM2, ASM2d, and ASM3*. Scientific and Technical Report No.9. IWA Publishing, London, UK, 2000.
- [4] J.B. Copp, editor. *The COST simulation benchmark*. European Commission, 2002.
- [5] H. Vanhooren, J. Meirlaen, Y. Amerlinck, F. Claeys, H. Vangheluwe, and P.A. Vanrolleghem. WEST: modelling biological wastewater treatment. *Journal of Hydroinformatics*, 5(1):27–50, 2003.
- [6] P. Fritzon. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, February 2004.
- [7] H. Vangheluwe, F. Claeys, S. Kops, F. Coen, and G.C. Vansteenkiste. A modelling simulation environment for wastewater treatment plant design. In *Proceedings of the 1996 European Simulation Symposium*, Genoa, Italy, October 24–26 1996.
- [8] F.H.A Claeys, P.A. Vanrolleghem, and P. Fritzon. A generalized framework for abstraction and dynamic loading of numerical solvers. In *Proceedings of the 2006 European Modeling and Simulation Symposium*, Barcelona, Spain, 2006.
- [9] F. Claeys, M. Chtepen, L. Benedetti, B. Dhoedt, and P.A. Vanrolleghem. Distributed virtual experiments in water quality management. *Water Science and Technology*, 53(1):297–305, 2006.
- [10] L. Benedetti, D. Bixio, F. Claeys, and P.A. Vanrolleghem. A model-based methodology for benefit/cost/risk analysis of wastewater systems. In *Proceedings of the iEMSs 2006 Conference*, Burlington, VT, 2006.

Session 2d

Mechanical Systems and Applications 2

Leaf spring modeling

Niklas Philipson

Modelon AB

Ideon Science Park SE-22370 Lund, Sweden

niklas.philipson@modelon.se

Abstract

Although leaf springs are one of the oldest suspension components they are still frequently used, especially in commercial vehicles. Being able to capture the leaf spring characteristics is of significant importance for vehicle handling dynamics studies. The conventional way to model leaf springs is to divide the spring into several rigid links connected to each other via rotational stiffnesses. This can easily be done with the Modelica Standard Library, but it results in hard-to-use models with long simulation times. The models in this paper are designed as generalized force elements where the position, velocity and orientation of the axle mounting gives the reaction forces in the chassis attachment positions.

Keywords: Leaf spring; Vehicle dynamics; Commercial vehicle suspensions

1 Introduction

The commercial VehicleDynamics Library [1] is currently undergoing expansions to suite heavy vehicles (figure 1), requiring models of new components such as leaf springs. This paper covers one technique to generate a leaf spring that has good simulation performance and still captures the following characteristics.

- The axle attachment position will deflect in an arc shape in the longitudinal-vertical plane under vertical loading conditions [3].
- Leaf spring suspension designs have two anti roll bar effects. The springs are stiff in roll (twist) which counteracts the vehicles roll motion if the spring is mounted to a rigid axle as in figures 8 and 10. If the axle is mounted asymmetrically, that is not centered on the middle of the spring, the axle will twist as the vehicle rolls. This will resist vehicle roll as well [4].

- The effective length of the leaf spring varies with deflection causing a varying spring rate. The models in this paper require large deflections for the effect to be seen, but this effect can be higher for other shapes and mounting types of the spring [3].

The basic idea for the model is to use five massless links connected with rotational elasticities with the axle mounted at the center of the middle link. A massless approximation is reasonable since the masses involved in rigid axles, wheels and the body of the vehicle are considerably higher than the mass of the leaf spring. The implemented leaf spring can easily be extended with masses connected to the frames at the leaf springs three mounting positions. The shape of the

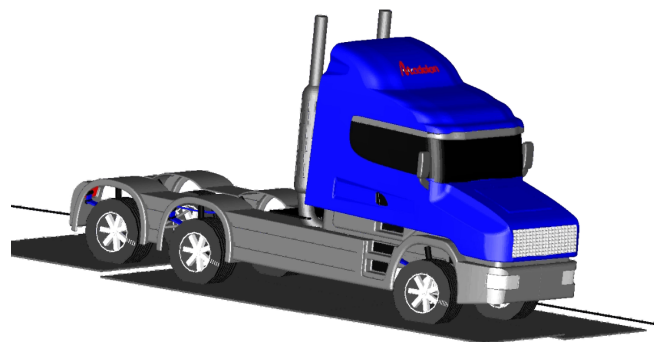


Figure 1: Tractor with leaf spring suspension in a shaker rig

leaf spring will be determined by the rotations between each link, except for the roll angles. These angles are left out of the equations of motion since they have very little impact on the leaf spring's shape. The roll resistance is handled as a rotational stiffness added to the torque equilibrium equations instead.

2 Reference MultiBody model

The model used as a basis for comparison is designed with components from the Multi-body package. The model consists of six rigid links connected by rotational stiffnesses that allow the center position to deflect in a plane. This design forms a planar loop and is only useful for vertical plane comparison.

Figure 2 illustrates a primitive suspension model assembled from two multi-body leaf springs. A translational joint is used to handle the distance variations in length between the mounting positions against the leaf springs. A spring is applied to the translational joint to control the lateral motion of the axle. The simulation time increases significantly with a stiffer translational spring. A more realistic model can be assembled by adding revolute joint for the lateral and roll motion as well, but the simulation time for just one planar leaf spring is already long.

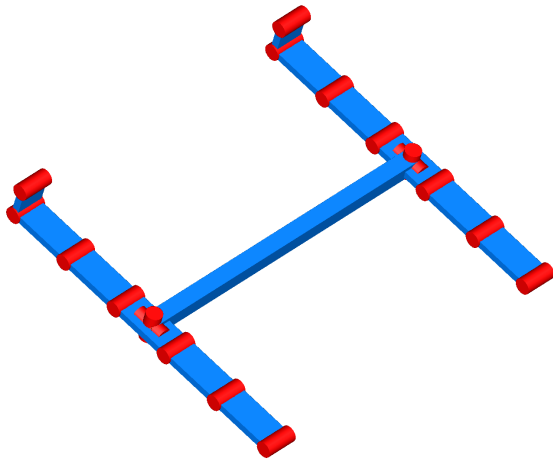


Figure 2: Multi body rigid axle suspension

3 Equations of motion

Lagrange’s method, equation (1), is used to derive the equations of motion resolved in the generalized coordinates $(p1_y \dots p4_y, p1_z \dots p4_z)$ as illustrated in figure 3. Together with the stiffnesses indicated in figure 4, these form the expressions for the potential energy U in equation (2).

$$\frac{dL}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} + \frac{\partial R}{\partial \dot{q}} = F_{q_i} \quad (1)$$

$$L = T - U \quad (2)$$

In the sequel, it is assumed that the spring is massless giving $T = 0$. Viscous damping is applied over the

generalized coordinates, giving

$$R = \sum_{i=1}^i \frac{1}{2} \cdot d_i \cdot \dot{q}_i^2 \quad (3)$$

where q_i and d_i denotes each generalized coordinate and the corresponding damping coefficient.

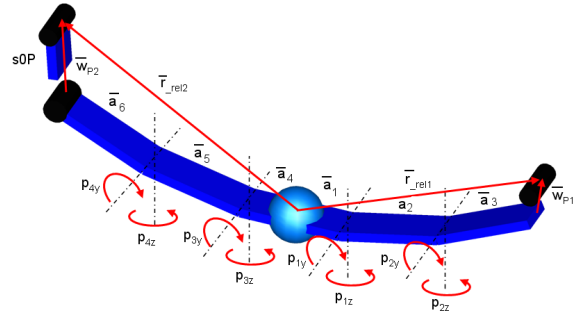


Figure 3: Definition of generalized coordinates and geometry properties.

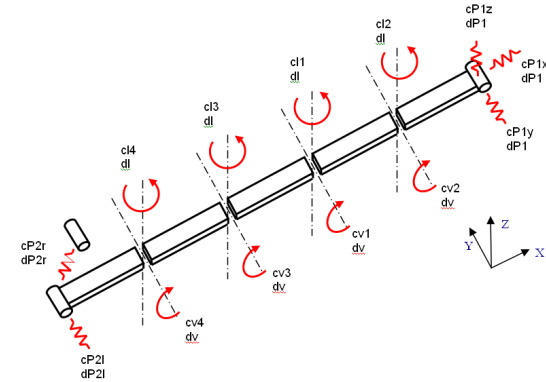


Figure 4: Parameters for stiffness and damping.

The potential energy stored in the spring is given by

$$\begin{aligned} U = & \frac{1}{2} c_{P1_x} w_{P1_x}^2 + \frac{1}{2} c_{P1_y} w_{P1_y}^2 + \frac{1}{2} c_{P1_z} w_{P1_z}^2 \\ & + \frac{1}{2} cv_1 p_{2_y}^2 + \frac{1}{2} cv_2 p_{2_y}^2 + \frac{1}{2} cv_3 p_{3_y}^2 \\ & + \frac{1}{2} cv_4 p_{4_y}^2 + \frac{1}{2} cl_1 p_{1_z}^2 + \frac{1}{2} cl_2 p_{2_z}^2 \\ & + \frac{1}{2} cl_3 p_{3_z}^2 + \frac{1}{2} cl_4 p_{1_z}^2 + \frac{1}{2} c_{P2_l} w_{P2_l}^2 \\ & + \frac{1}{2} c_{P2_r} w_{P2_r}^2 \end{aligned} \quad (4)$$

where $w_{P1_{x,y,z}}$ and $c_{P1_{x,y,z}}$ are the displacements and stiffnesses of the front eye bushing. $w_{P2_{r,l}}$ denote the lateral and radial displacement of the shackle with $c_{P2_{r,l}}$ as the corresponding stiffnesses.

A non linear bushing description including a linear and a cubic stiffness gives the forces

$$F = \int_0^w \underbrace{k_1 \cdot w^2 + k_2}_c dw \quad (5)$$

and the potential energy

$$E = \int_0^w F dw \quad (6)$$

which can be used in equation (2) instead of the linear model. The forces will in this case depend on

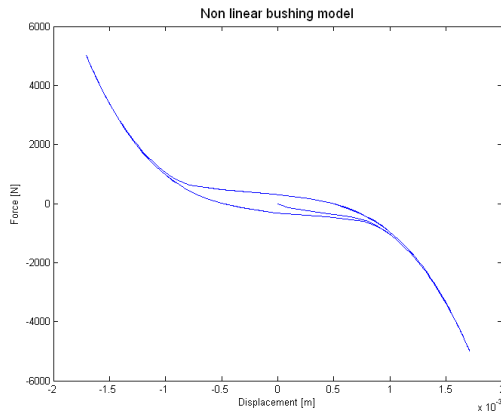


Figure 5: Force deflection diagram for front bushing

$c(p1y,p2y,p1z,p2z)$ and $w(p1y,p2y,p1z,p2z)$ for the front bushing. This results in a force deflection diagram for the front bushing seen in figure 5.

The displacements for the front eye bushing and the shackle are given by

$$\bar{w}_{P1} = \bar{r}_{rel1} - (\bar{a}_1 + T_{y1} \cdot T_{z1} \cdot \bar{a}_2 + R_{y1} \cdot R_{z1} \cdot T_{y2} \cdot T_{z2} \cdot \bar{a}_3) \quad (7)$$

and

$$\bar{w}_{P2} = \bar{r}_{rel2} - (\bar{a}_4 + T_{y3} \cdot T_{z3} \cdot \bar{a}_5 + T_{y3} \cdot T_{z3} \cdot T_{y4} \cdot T_{z4} \cdot \bar{a}_6) \quad (8)$$

respectively. Since the leaf spring is assumed to be rigidly mounted to the axle, it is convenient to resolve the equations for motion and force balance in the axle frame. The vectors \bar{r}_{rel1} and \bar{r}_{rel2} are expressed in the axle frame's coordinate system. The transformation matrices used to describe the end positions depending on the generalized coordinates used in equation (7) and (8) are given by

$$T_y = \begin{pmatrix} \cos(pX_y) & 0 & \sin(pX_y) \\ 0 & 1 & 0 \\ -\sin(pX_y) & 0 & \cos(pX_y) \end{pmatrix} \quad (9)$$

$$T_z = \begin{pmatrix} \cos(pX_y) & -\sin(pX_y) & 0 \\ \sin(pX_y) & \cos(pX_y) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (10)$$

where X represents the respective generalized coordinate. L from equation (2) is now completely described with the eight generalized coordinates and the stiffness parameters. When solving equation (1) with respect to the generalized coordinates, two sets of non-linear equations appear. Since these sets do not depend on each other but only are a function of the axle's position, they can be used separately if there is a need to model half a leaf spring in conjunction with, for instance, an air spring.

The complexity of the equation system increases rapidly with added degrees of freedom. If, for instance, roll stiffness is added to the leaf spring in the same way as the other elasticities, it will expand the equation systems from two systems with four unknowns to two systems with six unknowns. Each expression in the equation systems will also expand since equations 7 and 8 must be modified with additional transformation matrices for the roll angles.

Instead of adding the roll degree of freedom, the roll torque is added externally as described in the next section by terms in equation 17 and 18. This approach is considered valid since the roll angles are small under normal operation conditions and the spring is relatively stiff in roll compared to the bushings.

Five links seem to be a reasonable compromise that achieve a fast simulated model but still captures the essential spring characteristics, this discretization is also used in [2]. Possibly, a larger number of links could be used if the equations of motion were to be linearized. This might cause problems with the initial curvature which requires large angles between the links.

4 Force generation

The displacements and the displacement's derivative together with the stiffness and damping coefficients give the forces in the mounting positions to the chassis. The forces in the chassis mounts are given by equation (11) and (14),

$$\bar{f}_{P1} = \mathbf{C}_{P1} \cdot \bar{w}_{P1} + \bar{w}_{P1} \cdot d_{P1} - \bar{f}_{0P1} \quad (11)$$

$$f_{P2r} = c_{P2r} \cdot (w_{P2r} - sP0) + \dot{w}_{P2r} \cdot d_{P2r} - f_{0P2r} \quad (12)$$

$$f_{P2i} = c_{P2i} \cdot w_{P2i} + \dot{w}_{P2i} \cdot d_{P2i} \quad (13)$$

$$\bar{f}_{P2} = f_{P2_r} \cdot \hat{n}_{P2_r} + f_{P2_l} \cdot \hat{n}_{P2_l} \quad (14)$$

where \mathbf{C} is a diagonal (3x3) matrix with the translatory stiffnesses for the front eye bushing. The damping of the front bushing is currently set as one value for all directions. $\hat{n}_{P2_{l,r}}$ denote unit vectors in the radial and lateral directions of the shackle. The length of the shackle used for describing the shackles radial displacement in equation (12) is named sOP .

The force and torque equilibria are given by

$$\bar{\mathbf{0}} = \bar{f}_{P4} + \bar{f}_{P1} + \bar{f}_{P2} \quad (15)$$

and

$$\bar{\mathbf{0}} = \bar{t}_{P4} + \bar{t}_{P1} + \bar{t}_{P3} + \bar{r}_{rel1} \times \bar{f}_{P1} + \bar{r}_{rel2} \times \bar{f}_{P3} \quad (16)$$

respectively. The roll stiffness is modeled as a rotational spring and added to the torque equilibrium. The roll angle is the only variable that has an impact on the torque acting on the front bushings. This gives

$$p_r \cdot c_r \cdot \hat{n}_x = \bar{t}_{P1}. \quad (17)$$

as the resulting torque. The force is calculated in the lower shackle mount and must be transformed as

$$\bar{t}_{P2} = \bar{f}_{P3} \times \hat{n}_{P2} \cdot sOP + p_r \cdot c_r \hat{n}_x \quad (18)$$

p_r is the spring's roll angle with the corresponding rotational stiffness c_r . The unit vector in the x direction of the axle frame's coordinate system is denoted \hat{n}_x .

5 Implementation

The primitive model of the leaf spring requires geometry positions in a two dimensional plane. For the model to be useful, a wrapper is needed to translate the initial three dimensional positions to parameters for the leaf spring. This is done according to equation 22 though 29. The location of the primitive model's

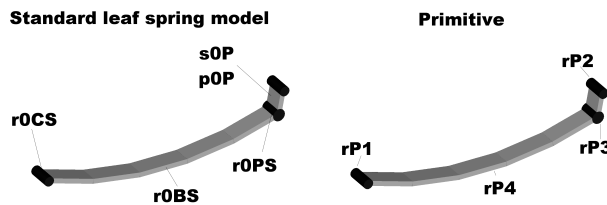


Figure 6: Leaf spring primitive and wrapped model

hard points are illustrated in figure 6.

The transformation matrix T in which the planar positions are resolved is given by the base vectors (nx, ny) .

$$\bar{n}x = \bar{r}_{0CS} - \bar{r}_{0PS} \quad (19)$$

$$\bar{n}y = (\bar{r}_{0PS} - \bar{r}_{0BS}) \times (\bar{r}_{0CS} - \bar{r}_{0BS}) \quad (20)$$

Equation (20) is unsolvable when the vectors are parallel. This is taken care of by an assertion which encourages the user to enter ny manually.

$$\bar{r}1 = T (\bar{r}_{0PS} - \bar{r}_{0CS}) \quad (21)$$

$$\bar{r}2 = T (\bar{r}_{0BS} - \bar{r}_{0CS}) \quad (22)$$

The vectors $(r1, r2)$ resolved in T have y-values equal to zero, and can thus be used to extract the positions for the four hard points used in the primitive model according to equations (23) though (26).

$$r\bar{p}1 = \bar{\mathbf{0}} \quad (23)$$

$$\bar{r}_{P2} = \bar{r}_{P3} + sOP \cdot (\sin(pOP), \cos(-pOP)) \quad (24)$$

$$\bar{r}_{P3} = (r1_x, r1_z) \quad (25)$$

$$\bar{r}_{P4} = (r2_z, r2_z) \quad (26)$$

The leaf spring's curvature is defined as $1/R$ where R is the radius of the leaf springs shape. The implemented models have a curvature that depends on the hard points for the three mounting positions. There is one curvature for the rear part generated from the axle and the lower shackle mounts position and one for the front part generated in the same way as for the rear.

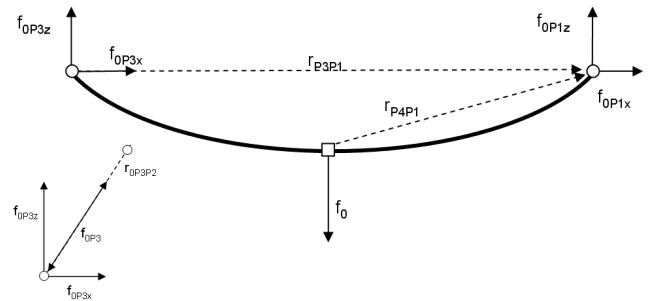


Figure 7: Pretension forces

To enable an easy way to determine the shape of the leaf spring and the ride height of the vehicle at the design configuration it is necessary to specify a pretension value corresponding to the load when the vehicle is at rest. The forces from pretension are given by

$$f_{0P1_x} + f_{0P3_x} = 0 \quad (27)$$

$$f_{0P1_z} - f_0 + f_{0P3_z} = 0 \quad (28)$$

$$f_{0P3_z} \cdot r_{P3P1_x} - f_{0P3_x} \cdot r_{P3P1_z} - f_0 \cdot r_{P4P1_x} = 0 \quad (29)$$

$$f_{0P3_z} = f_{0P3} \cdot \frac{r_{0P3P2_z}}{|r_{0P3P2}|} \quad (30)$$

$$f_{0P3_x} = f_{0P3} \cdot \frac{r_{0P3P2_x}}{|r_{0P3P2}|} \quad (31)$$

and indicated in figure 7. These forces are calculated initially and added as static values in equation (11) and (12).

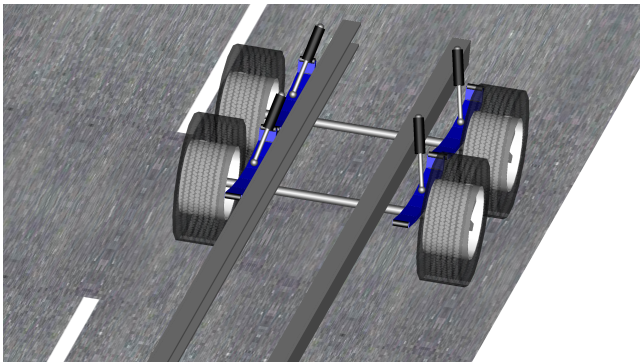


Figure 8: Semi trailer bogie suspension

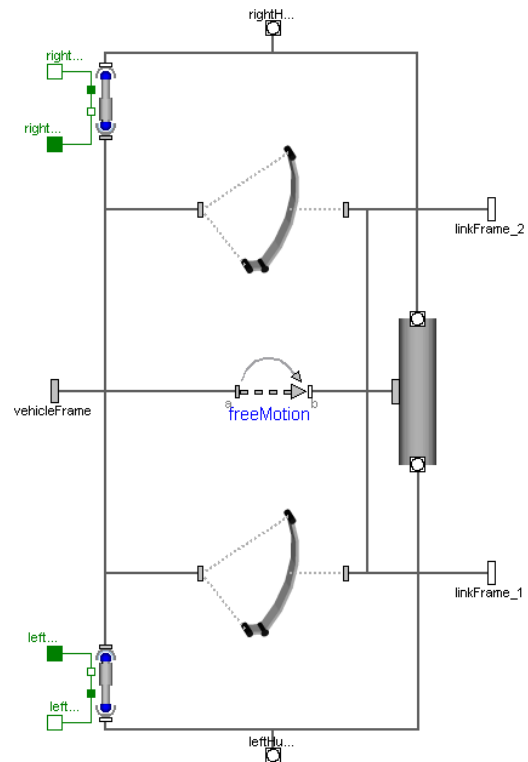


Figure 10: Diagram view of leaf spring axle carriage

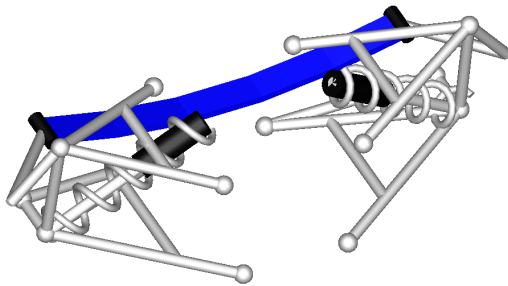


Figure 9: Leaf spring and double wishbone suspension

The implemented leaf spring models can be used in numerous designs, here presented in a semi trailer bogie suspension, figure 8, and in a double wishbone design, figure 9. One of the axles in the semi trailer suspension is assembled as shown in figure 10. Figure 9 illustrates another leaf spring model without a shackle mounted between two wishbones and with the center attachment mounted to the chassis. This model is based on the same technique as the standard leaf spring model.

6 Parametrization

The parameters needed for the leaf spring consists of positions, stiffnesses, dampings, and animation properties.

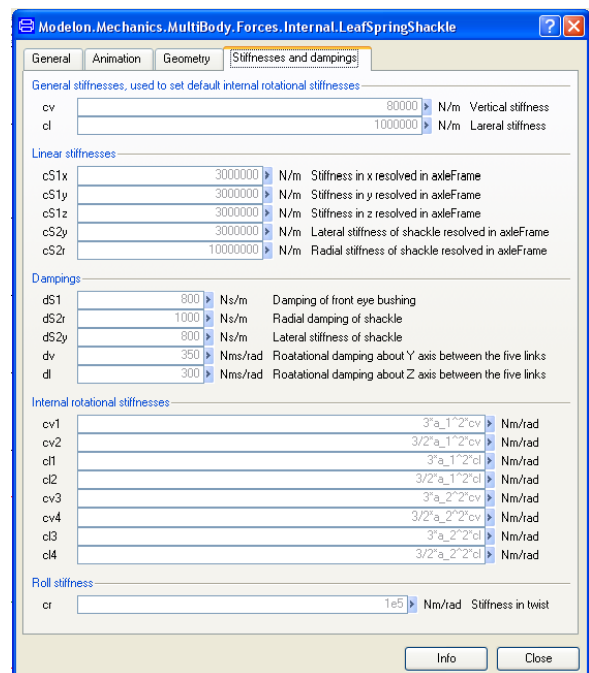


Figure 11: Elasticity parameters

Figure 11 displays the elasticity parameters for the primitive model. The default value for the internal rotational stiffnesses are calculated from the vertical and lateral stiffness under the assumption that the leaf spring will deflect in the shape of an arc. The internal rotational stiffnesses can be set manually to enable the user to customize the deflection profile.

The parametrization can easily be changed to suite different specific types of leaf springs in terms of shape and asymmetric stiffness.

7 Validation and results

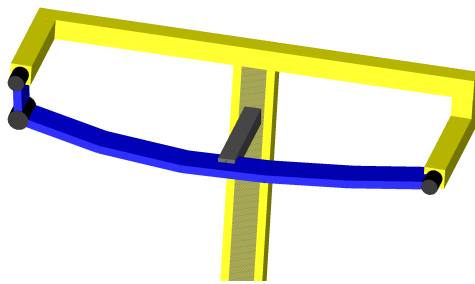


Figure 12: Leaf spring test rig

The validation of the standard leaf spring has been carried out by comparing the model to a reference multi-body model described in section 2. A test rig, figure 12, has been used to generate the dynamic and kinematic comparison. As seen in figure 13 the verti-

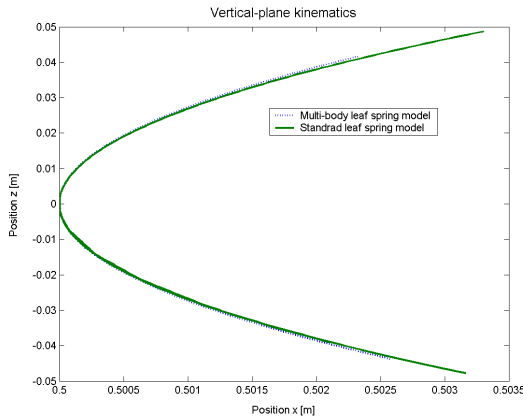


Figure 13: Vertical plane kinematics comparison

cal plane kinematics of the leaf spring modeled with rigid elements are virtually the same as for the model described by Lagrange’s equation. Both models are damped via viscous damping over each generalized coordinate and corresponding revolute joint for the

multi-body model. The vertical plane dynamics for the different models are very similar to each other as long as the excitation does not consist of high frequency components as in figure 14.

The fact that the standard model has both stiffness and damping in the mount positions makes it a bit complicated to compare these results, but without fine tuning of the stiffness and damping they perform as shown in figure 13. The differences can easily be related to the bushings in the mount positions and the massless approximation used in the standard leaf spring.

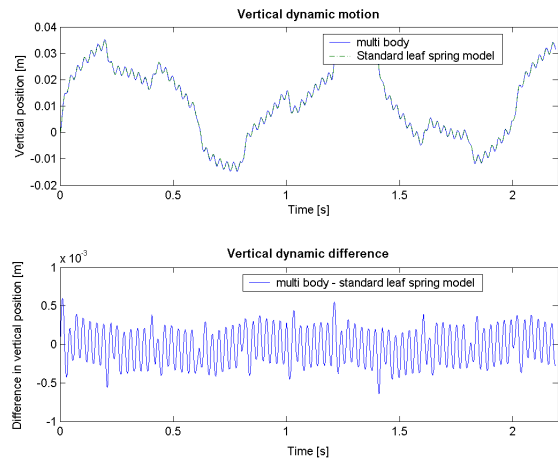


Figure 14: Vertical plane dynamics

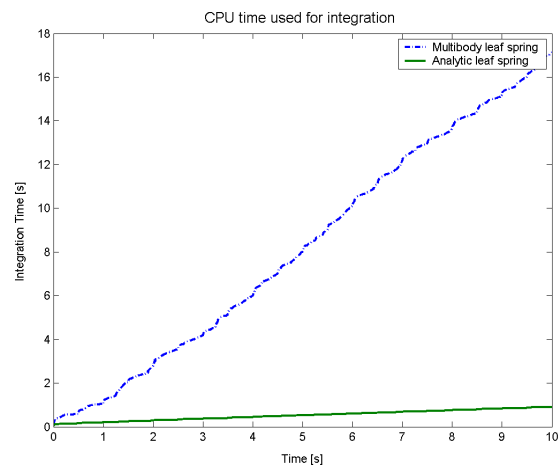


Figure 15: Cpu time used for simulation of the different suspension models

A suspension assembled as in figure 10 simulates approximately 18 times faster then the reference suspension, as shown in figure 15.

A comparison of the kinematic and dynamic behavior of two multi body leaf springs with five versus

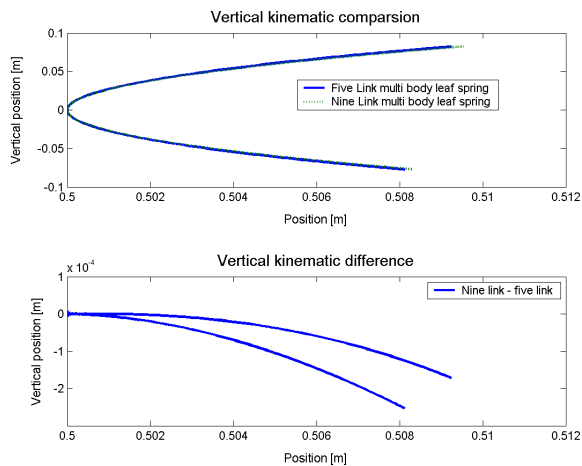


Figure 16: Kinematic comparison between nine and five link leaf spring

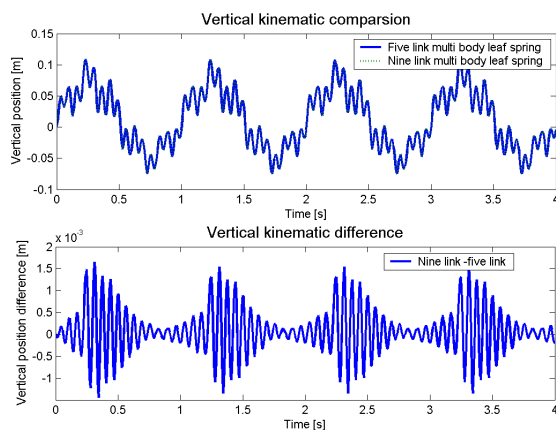


Figure 17: Dynamic comparison between nine and five link leaf spring

nine links is illustrated in figure 16 and 17. The differences between the models are small which implies that the five link leaf spring meets the requirements for vehicle handling simulations.

The shackle has big influence on the leaf spring's kinematics. The shape of the leaf spring in the comparison results in larger deflection in bounce than in rebound, figure 16. This because the shackle's lower mount towards the spring always moves upwards with deflection. Other geometries would give different results.

8 Summary

The leaf spring model is essential for heavy vehicle handling dynamics simulations. The proposed model

is superior to the multi-body reference model with respect to simulation time and it is much easier to parameterize the geometry positions and to implement it in suspension designs. The model is equipped to deal with the specific characteristics of a leaf spring. It is possible to add forces through the same equations as the pretension but varying over time. This enables a user to add additional force elements as damping via hysteresis or air springs. The standard leaf spring model fulfills all the requirements specified in section 1.

References

- [1] *J. Andreasson, M. Gävert.* The VehicleDynamics Library — Overview and Applications Modelon., Homepage: <http://www.modelon.se/>. In *Proceedings of Modelica'2006*, Vienna, Sep. 2006.
- [2] *Georg Rill, Norbert Kessing, Olav Lange and Jan Meier:* Leaf Spring Modelling for Real Time Applications In the *18th IAVSD-Symposium in Atsugi, Japan 2003*, 2003.
- [3] *SAE:* Spring Design Manual ISBN: 1-56091-680-X, 1996.
- [4] *A grimm, C. Winkler and ,R. Sweet* Mechanics of Heavy Duty Truck Systems. University of Michigan transportation research institute, UK , 2004.

Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation

Ivan I. Kosenko¹, Maria S. Loginova², Yaroslav P. Obraztsov², Mayya S. Stavrovskaya¹

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

²Moscow State Academy of Instrument Making and Computer Science
Department of Applied Mathematics, Stromynka str. 20, Moscow, 107646, Russia

Abstract

Using an example of the snakeboard, a vehicle with four wheels and nonholonomic constraints, the process of construction and verification for the sparse dynamical models of the multibody systems is analyzed. Two approaches for the formal representation of the models: object-oriented, and bond graph based are considered. Energy based similarities between these approaches are analyzed.

A detailed description of the bond graph representation for the most general type of constraint is presented. It turned out the resulting total bond graph model of the multibody system dynamics always has exactly a canonical junction structure. This representation has a tight correspondence with our recent object-oriented implementation of the mechanical constraint architecture. As an example Modelica implementation of the joint classes family is investigated. Finally these classes are applied to construct the snakeboard dynamic model.

Keywords: vehicle; nonholonomic; disc; wheelset; snakeboard; object-oriented modeling; bondgraph; canonical junction structure; joint; servoconstraint

1 Introduction

When developing a computer model of the multibody system (MBS) dynamics it is interesting to have a unified technology to construct the models in an efficient way. It turns out Modelica language provides a tools to resolve such a problem successively step by step using its natural approaches. One of them is connected tightly with the so-called multiport representation of the models initially based on the bond graph use. These latter in turn based on the idea of energy interaction, and substantially on energy conservation

for physically interconnected subsystems of any engineering type.

Moreover, Modelica introduces the notions similar to ones of the bond graph theory, but in a way more natural for the usual engineering approaches with forces, interfaces, parameters, equations etc. Consider in the sequel a technology to construct a model of MBS dynamics with constraints of any specific type in a unified way. Note that the unilateral constraints can also be included in the further consideration process.

2 Constraint representation via bond graphs

Previously, when considering a unified model of the constraint, or, in a more general way, any physical interaction between two rigid/deformable bodies we defined [1, 2] two classes of the kinematic and the effort ports. These ones are the kinematic and wrench connectors. It turned out the connections of such types make it possible to construct a model of the bodies interactions based on the causality physically motivated. Namely, the constraint object imports the kinematic information accepting it from the objects of interacting bodies and reciprocally exports it in the opposite direction. Thus the constraint “computes” an efforts the bodies interact by.

On the other hand geometric formalisms to represent the MBS dynamics are known [3] which operates with the similar information objects: twists and wrenches. In our approach twist is defined by the `KinematicPort` class, and wrench obviously corresponds to our `WrenchPort` class. The representation under consideration is tightly connected with the power based approach to modeling, so-called bond graphs [4].

Indeed, let the rigid body kinematics be defined by the

twist $(\mathbf{v}, \boldsymbol{\omega})$, where \mathbf{v} is the mass center velocity, and $\boldsymbol{\omega}$ is the body angular velocity. Further let all the forces acting upon the body be reduced to the wrench (\mathbf{F}, \mathbf{M}) with the total force \mathbf{F} and the total torque \mathbf{M} . Thus the total power of all the forces acting on the body is computed by the known formula

$$W = (\mathbf{v}, \mathbf{F}) + (\boldsymbol{\omega}, \mathbf{M})$$

using to represent a multibond in the bond graphs simulating the MBS dynamics. We have in such the case an evident canonical duality between twists and wrenches.

Sometimes wrenches are selected as flow variables. In other cases twists play this role. For instance similarities between electricity and mechanics cause the parallelism for electric current and forces/torques in one dimensional powertrains of mechanisms. In this case we can set a correspondence between the Kirchhoff law for currents and the d’Alembert principle for external forces and forces of inertia “acting” upon the body.

In our opinion it may be interesting enough to apply an approach dual to the first one mentioned above. Such an approach is more natural in traditional classical mechanics and assumes twist for the flow variable in the multibond. In the further course we present an illustration for this approach and demonstrate its convenience to construct the mechanical constraints of different types. Moreover, object-oriented implementation may be interpreted in both above dual approaches in a symmetric ways.

Let us trace now the similarities between the bond graphs and our MBS models. Evidently the pair of classes `KinematicPort/WrenchPort` plays a role of the multiport notion, and corresponding pairs of connections in Figure 1 stand for the notion of a bond.

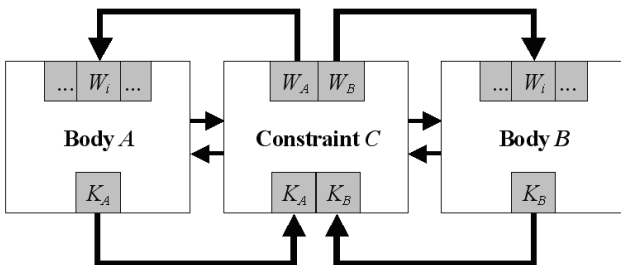


Figure 1: Architecture of Constraint

Furthermore, in this way we can associate an object of the `RigidBody` class with 1-junction, while 0-junction is associated with the object of the class `Constraint`. The relevant general bond graph rep-

resentation of the constraint in any MBS may be depicted as it shown in Figure 2.

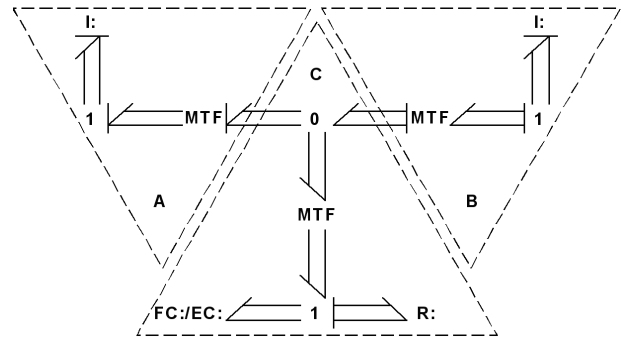


Figure 2: Architecture of Constraint: Bond Graph Representation

All multibonds here consist of the twist $(\mathbf{v}, \boldsymbol{\omega})$ signals representing the flow component, and the wrench (\mathbf{F}, \mathbf{M}) signals as an effort. Causality of an inrtance elements arranges according to the Newton–Euler system of ODEs. Left and right transformers are to shift the twist from the mass center to the contact point according to the known Euler formula: $(\mathbf{v}, \boldsymbol{\omega}) \mapsto (\mathbf{v} + [\boldsymbol{\omega}, \mathbf{r}], \boldsymbol{\omega})$, where the vector \mathbf{r} begins at the corresponding center of mass and ends at the contact point. Reciprocally the wrenches shift to the body mass center from point of the contact in a following way: $(\mathbf{F}, \mathbf{M}) \mapsto (\mathbf{F}, \mathbf{M} + [\mathbf{r}, \mathbf{F}])$. As one can see easily the transformers conserve the power.

Central transformer is responsible for the transfer to orthonormal base at the contact point with the common normal unit vector and two others being tangent ones to both contacting bodies’ surfaces supposed regular enough. For definity we interpret here the case of usual contact interconnection between the bodies by their outer/inner surfaces. If the inertial coordinates of these vectors compose columns of the orthogonal rotational matrix Q then shifting from bottom to top across the transformer in Figure 2 we will have for the flow signals: $(\mathbf{v}, \boldsymbol{\omega}) \mapsto (Q\mathbf{v}, Q\boldsymbol{\omega})$. Likewise when shifting in a reverse direction we have a transformation of the efforts: $(\mathbf{F}, \mathbf{M}) \mapsto (Q^{-1}\mathbf{F}, Q^{-1}\mathbf{M})$ also conserving the power. Organization of the 0-junction depicted in Figure 2 provides a possibility to compute exactly the relative velocities at the constraint contact point.

Note that it is a usual practice to attach the inrtance element to 1-junction, in particular because of its causality nature, see for example [5, 6]. Figure 2 in some degree can remind us an element of the lumped model for the flexible beam dynamics.

Causality for some multibonds inside the constraint object is defined individually for each particular scalar

bond [7] depending on the type of the constraint and is assigned finally after the whole MBS model compilation. For instance, if the constraint is of the slipping type at a contact then supposing decompositions of the relative velocities and contact forces $\mathbf{v} = \mathbf{v}_n + \mathbf{v}_\tau$, $\mathbf{F} = \mathbf{F}_n + \mathbf{F}_\tau$ we have the following flow constraint, element FC, $\mathbf{v}_n = \mathbf{0}$ representing one scalar kinematic equation for the normal relative velocity, and the effort constraint, element EC, $\mathbf{F}_\tau = \mathbf{0}$, $\mathbf{M} = \mathbf{0}$ representing two scalar equations for the tangent contact force plus three scalar equations for the contact torque. Nonzero tangent force at the contact may arise due to the resistive element, see the bottom right multibond. If we will continue to build the bond graph model for the whole MBS in a proposed way then finally we can arrive exactly to the so-called canonical junction structure [7] useful for the formal procedures of the bond graph optimal causality assignment. For this we have to add an intermediate 0-junctions for elements attached to 1-junction in the constraint component C , see Figure 2.

Leaving some multibonds without the causality assignment and trusting this work to compiler we apply a so-called acausal modeling [8]. On the other hand if we will act in a manner close to the real cases of constraints with the flexibility then instead of the constraint elements FC/EC, we have to use an element of the compliance with the causality uniquely determined, see Figure 3.

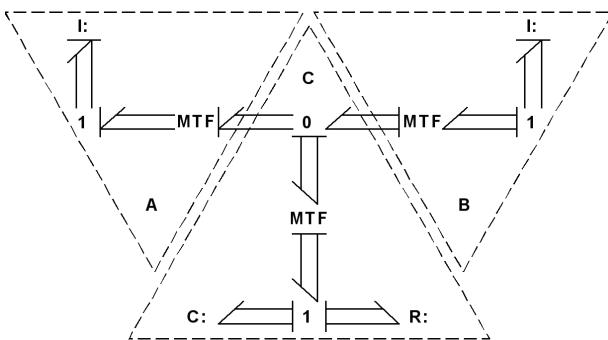


Figure 3: Bond Graph of Constraint with Compliance

Further we analyze one example of the constraint frequently occurring in engineering applications: we consider an object classification of the joint constraint.

3 Implementation of the joint constraint

For simplicity and clearness we will apply the component library to simulate the dynamics of MBSs with bi-

lateral constraints [1]. Application of the components for the unilateral constraints [2] doesn't change anything in principle. The only difference is that dynamics of the moving bodies becomes more complicated. For example in the latter case a vehicle under simulation get an ability to bounce over the uneven surface it rolls on. In addition, its wheels can slip while moving. Thus in frame of the current paper we suppose that nonholonomic constraints implemented exactly, without any slip or separation with respect to (w. r. t.) the surface.

Remind that according to our technology of the constraint construction [1] two connected bodies are identified by convention with the letters A and B fixed for each body. All kinematic and dynamic variables and parameters concerned one of the bodies are equipped with the corresponding letter as a subscript.

Class `Joint` plays a key role in the future model of a vehicle we will build. `Joint` is a model derived from the base class `Constraint`. Remind [2] that in order to make a complete definition of the constraint object behavior for the case of rigid bodies one has to compose a system of twelve algebraic equations w. r. t. to twelve coordinates of vectors \mathbf{F}_A , \mathbf{M}_A , \mathbf{F}_B , \mathbf{M}_B constituting the wrenches acting upon the connected bodies. First six equations always present in the base model `Constraint` due to Newton's third law. For definitivity suppose these six equations are used to express six components of \mathbf{F}_B , \mathbf{M}_B depending on \mathbf{F}_A , \mathbf{M}_A . Thus six components of \mathbf{F}_A , \mathbf{M}_A remain as unknowns. To determine them each constraint of rigid bodies need in six additional independent algebraic equations. These equations can include components of force and torque directly, or be derived from the kinematic relations corresponding to specific type of the constraint.

In the case of the joint constraint being investigated here let us represent the motion of the body B as a complex one consisting of the body A convective motion w. r. t. an inertial frame of reference which is similar to the Modelica Standard MultiBody Library model `world`, and a relative motion w. r. t. the body A . An absolute motion is one of the body B w. r. t. inertial system.

Define the joint constraint with help of the following parameters: (a) a unit vector \mathbf{n}_A defining in the body A an axis of the joint; (b) a vector \mathbf{r}_A fixed in the body A and defining a point which constantly stays on the axis of the joint; (c) a vector \mathbf{r}_B fixed in the body B and defining a point which also constantly stays on the axis of the joint. The main task of the base joint class is to keep always in coincidence the geometric axes

fixed in each of the bodies.

First of all one has to compute the radii vectors of the points fixed in the bodies w. r. t. inertial system

$$\mathbf{R}_\alpha = \mathbf{r}_{O_\alpha} + T_\alpha \mathbf{r}_\alpha \quad (\alpha = A, B),$$

where [2] \mathbf{r}_{O_α} is the position of the α -th body center of mass, T_α is its current matrix of rotation. The joint axis has the following components

$$\mathbf{n}_{Ai} = T_A \mathbf{n}_A$$

in the inertial frame of reference. According to the equation for relative velocity for the marked point of the body B defined by the position \mathbf{R}_B we have

$$\begin{aligned} \mathbf{v}_{Ba} &= \mathbf{v}_{Be} + \mathbf{v}_{Br}, \\ \mathbf{v}_{Ba} &= \mathbf{v}_{OB} + [\boldsymbol{\omega}_B, T_B \mathbf{r}_B], \\ \mathbf{v}_{Be} &= \mathbf{v}_{OA} + [\boldsymbol{\omega}_A, \mathbf{R}_B - \mathbf{r}_{OA}], \end{aligned} \quad (1)$$

where \mathbf{v}_{Ba} , \mathbf{v}_{Be} , \mathbf{v}_{Br} are an absolute, convective, and relative velocities of the body B marked point, $\boldsymbol{\omega}_A$, $\boldsymbol{\omega}_B$ are the bodies angular velocities.

Furthermore, according to the computational experience of the dynamical problems simulation the pre-compiler work is more regular if the kinematic equations are expressed directly through accelerations. Indeed, otherwise the compiler tries to perform the formal differentiation of equations for the velocities when reducing an index of the total DAE system. Frequently this leads to the problems either in time of translation or when running the model.

In the first case usually diagnostics of the compiler essentially helps the developer. In the second case the model has an unpredictable behavior, and only manual preliminary reduction “regularizes” the simulation process. Thus we differentiate equations (1) and obtain an equations for the relative linear acceleration in the form

$$\begin{aligned} \mathbf{a}_{Ba} &= \mathbf{a}_{OB} + [\boldsymbol{\epsilon}_B, T_B \mathbf{r}_B] + [\boldsymbol{\omega}_B, [\boldsymbol{\omega}_B, T_B \mathbf{r}_B]], \\ \mathbf{a}_{Be} &= \mathbf{a}_{OA} + [\boldsymbol{\epsilon}_A, \mathbf{R}_B - \mathbf{r}_{OA}] + [\boldsymbol{\omega}_A, [\boldsymbol{\omega}_A, \mathbf{R}_B - \mathbf{r}_{OA}]], \\ \mathbf{a}_{Ba} &= \mathbf{a}_{Be} + 2[\boldsymbol{\omega}_A, \mathbf{v}_{Br}] + \mathbf{a}_{Br}, \\ \mathbf{a}_{Br} &= \mu \mathbf{n}_{Ai}, \end{aligned} \quad (2)$$

where \mathbf{a}_{Ba} , \mathbf{a}_{Be} , \mathbf{a}_{Br} are an absolute, convective, and relative accelerations of the body B marked point, $\boldsymbol{\epsilon}_A$, $\boldsymbol{\epsilon}_B$ are the bodies angular accelerations.

We also need in an analytic representation of the conditions that the only projections of the bodies angular velocities and accelerations having a differences are ones onto the joint axis. Corresponding equations have a form

$$\begin{aligned} \boldsymbol{\omega}_B &= \boldsymbol{\omega}_A + \boldsymbol{\omega}_r, \\ \boldsymbol{\epsilon}_B &= \boldsymbol{\epsilon}_A + [\boldsymbol{\omega}_A, \boldsymbol{\omega}_r] + \boldsymbol{\epsilon}_r, \\ \boldsymbol{\epsilon}_r &= \lambda \mathbf{n}_{Ai}, \end{aligned} \quad (3)$$

where $\boldsymbol{\omega}_r$, $\boldsymbol{\epsilon}_r$ are the relative angular velocities and accelerations.

The Modelica code of the class `Joint` reads

```

partial model Joint
  extends Constraint;
  parameter Real [3] nA;
  parameter SI.Position [3] rA;
  parameter SI.Position [3] rB;
  SI.Position [3] RA;
  SI.Position [3] RB;
  SI.Velocity [3] vBa;
  SI.Velocity [3] vBe;
  SI.Velocity [3] vBr;
  SI.Acceleration [3] aBa;
  SI.Acceleration [3] aBe;
  SI.Acceleration [3] aBr;
  SI.AngularVelocity [3] omegar;
  SI.AngularAcceleration [3] epsilonNr;
  Real [3] nAi;
  SI.Force F; // Force along axis
  SI.Torque M; // Torque about axis
  SI.Acceleration mu;
  SI.AngularAcceleration lambda;
equation
  RA = InPortA.r + InPortA.T*rA;
  RB = InPortB.r + InPortB.T*rB;
  nAi = InPortA.T*nA;
  vBa = InPortB.v +
    cross(InPortB.omega,
          InPortB.T*rB);
  vBe = InPortA.v +
    cross(InPortA.omega,
          RB - InPortA.r);
  vBa = vBe + vBr;
  aBa = InPortB.a +
    cross(InPortB.epsilon,
          InPortB.T*rB) +
    cross(InPortB.omega,
          cross(InPortB.omega,
                InPortB.T*rB));
  aBe = InPortA.a +
    cross(InPortA.epsilon,
          RB - InPortA.r) +
    cross(InPortA.omega,
          cross(InPortA.omega,
                RB - InPortA.r));
  aBa = aBe + aBr +
    2*cross(InPortA.omega, vBr);
  aBr = mu*nAi;
  omegar = InPortB.omega -
    InPortA.omega;
  epsilonNr = InPortB.epsilon -
    InPortA.epsilon -
    cross(InPortA.omega, omegar);
  epsilonNr = lambda*nAi;
  F = OutPortA.F*nAi;
  M = OutPortA.M*nAi;
  OutPortA.P = RA;

```

```

    OutPortB.P = RA;
end Joint;

```

Besides the kinematic scalars μ, λ we will need in their reciprocal values $F = (\mathbf{F}_A, \mathbf{n}_{Ai}), M = (\mathbf{M}_A, \mathbf{n}_{Ai})$ correspondingly. Note that the class described above is a partial one and can be used to produce any imaginable model of the joint type constraint. To obtain a complete description of the joint model one has to add to the behavioral section exactly two equations. One of them is to define one of the values μ, F (translatory case). Other equation is intended to compute one of the values λ, M (rotary case).

Regarding the general scheme depicted in Figure 2 we can conclude that the equations (1), (2), (3) together implement implicitly the constraint transformer to the joint local coordinate system and four scalar flow constraints forbidding relative translatory and rotary motions in the direction orthogonal to the joint axis. For derived classes only two free scalar bonds remain.

Here we encounter the known complementarity rules once more in a way similar to one described in [2]. In our context the variables in the pairs $(\mu, F), (\lambda, M)$ are mutually complement, where one of μ, λ is to be utilized for the flow constraint and one of F, M is used to compose the effort constraint. All the variables mentioned complete the set of constraints for the remaining yet unused joint axis creating thus two final scalar constraint elements in the bond graph of Figure 2.

Namely, the equations (2) implementing the Coriolis theorem for accelerations simultaneously implement, in an implicit manner, two scalar flow constraints, FC-elements, from the bottom left corner of the multi-bondgraph model in Figure 2. These flow constraints due to compiler restrictions constructed using accelerations instead of the velocities being used in a classic bond graph approach. The constraints have an obvious kinematic sense: they prevent the relative motion of the body B marked point in two directions normal to the joint axis fixed in the body A .

In addition, the equations (3) implement two other scalar flow constraints, this time for the rotary motion. These constraints forbid the relative rotation of the body B w. r. t. body A about two axes each normal to the joint axis mentioned above which is rigidly connected with the body A .

Note, that the construct of equations (2) and (3) is such that they allow the body B relative motion along and about the joint axis of the body A thus implementing the kinematic pair with two DOFs. Returning to Figure 2 of the general constraint multi-bondgraph we can conclude that the vertical multibond attached to 0-

junction implements flow variables corresponding to the relative body B motion w. r. t. body A in inertial coordinates. Such a description supposes an existence of the special coordinates reference frame connected with the body A at its joint constraint marked point. The transformation to these coordinates is implemented exactly via corresponding transformer, central in the triangle block C . The transformer itself nests in formulae of equations (2) and (3).

Consider several examples of the classes derived from the `Joint` model for the several particular types of joints. The model `FixedIdealJoint` is defined by the equations

$$\mu = 0, \quad M = 0$$

and prevents the relative motion along the joint axis but allows free rotation about it. It is exactly a revolute joint without any control for the rotary motion. The model `FreeIdealJoint` is defined by the equations

$$F = 0, \quad M = 0$$

permitting free translation along and free rotation about the joint axis. Class `SpringIdealJoint` described by the equations

$$F = cv + d\dot{v}, \quad M = 0, \quad \ddot{v} = \mu$$

with an initial data $v(t_0) = 0, \dot{v}(t_0) = 0$ for the relative translatory position v provides a viscoelastic compliance with the stiffness c and damping d . The rotary motion remains free. This model is useful to simulate almost rigid constraints to avoid the potential problems with so-called statically undefinable systems of forces acting upon the ideal rigid bodies.

The model `FixedControlledJoint` with the behavior defined by the equations

$$\mu = 0, \quad M = f(t, \varphi, \dot{\varphi}), \quad \ddot{\varphi} = \lambda \quad (4)$$

provides the rotating torque as a control effort with the prescribed control function $f(t, \varphi, \dot{\varphi})$. Initial data $\varphi(t_0) = \varphi_0, \dot{\varphi}(t_0) = \dot{\varphi}_0$ are prepared according to the initial data concerning the joint. From the bond graph viewpoint the second equation in (4) can be implemented as a combination of the source effort, compliance, and resistance elements. This type of joint corresponds to the `Revolute` joint constraint of Modelica Standard Library from the `ModelicaAdditions` package. Such a joint can be driven by the electromotor.

The model `FreeSlideJoint` defined by the equations

$$F = 0, \quad \lambda = 0$$

provides free, without any resistance, relative sliding along the joint axis without any rotation about it. As one can see this is a prismatic type of joint.

We can reformulate the `FixedControlledJoint` model creating the model `FixedServoJoint` in a following useful way

$$\mu = 0, \quad \lambda = f(t, \varphi, \dot{\varphi}), \quad \ddot{\varphi} = \lambda$$

thus composing a kinematic restricting constraint, so-called servoconstraint. The function $f(t, \varphi, \dot{\varphi})$ supposed as a prescribed one. Initial data for the angle φ of the relative rotation are prepared in the same way as for (4). It is clear one can create a lot of other different combinations of equations to construct the joint constraints needed in engineering applications.

The derived joint classes described here are to close the system of kinematic equations (2) and (3) completing them mainly by two scalar additional equations, each playing a role of an either FC-element, like $\mu = 0$, or EC-element, like $F = 0$. Any time to be able to construct a consistent system of equations for the total model we have to follow the guidelines of the complementarity rules.

These latter correspond to the notions of the bond graph theory in a natural way. Indeed, the theory of bond graphs is based on the energy interactions. Every our multibond being an energy/power conductor reflects complementarity by its twist/wrench duality. To close the total DAE system for the model under development we have to “close” or rather to “seal” each free scalar bond in EC/FC-element of the block *C* in Figure 2 by the corresponding one scalar equation for flow or effort variable. Thus here we outline the main rule to compose equations for the models of constraints for MBS of any type in a consistent way when applying the object-oriented approach. In the further course we present an example for the systematic application of the rules mentioned.

4 Example of the snakeboard

The snakeboard [9], see Figure 4, represents a four wheeled vehicle moving in field of gravity on a horizontal surface due to the servocontrol of a relative rotation of wheelsets and a flywheel located at the midpoint of the coupler and having a vertical axis of rotation. The flywheel simulates a torso of the snakeboard rider.

We will construct the model hierarchically step by step verifying and integrating the parts into an assembly units. Ideal mechanical system of the snakeboard has

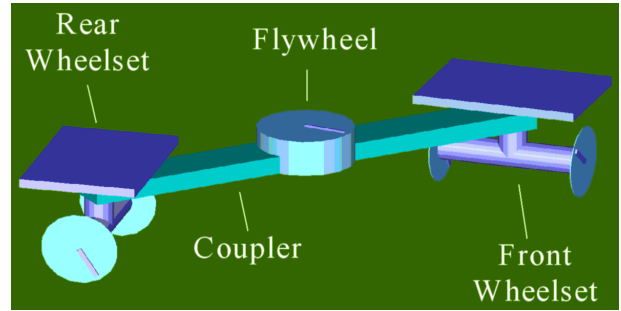


Figure 4: The Snakeboard

three degrees of freedom (DOF). But we will add new DOFs on some stages of modeling either to make the model more physically oriented or to apply any procedures of regularization.

4.1 Dynamics of the rolling disc

This problem is a classic one of dynamics [10] and has a visual representation depicted in Figure 5

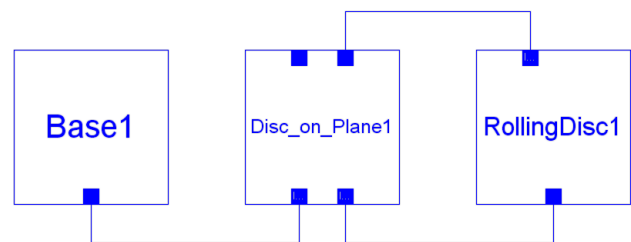


Figure 5: Visual Model of the Rolling Disc

Disc, the Body *B*, supposed an axisymmetric rigid body which is able to roll on the another body, horizontal surface, only by the curve fixed in the Body *B*. In our case this curve supposed a circle relocated in the plane

$$z_B = 0 \tag{5}$$

of the Body *B* coordinate system $O_Bx_By_Bz_B$ and has the fixed radius *R*, see Figure 6. In the current paper we assume that the nonholonomic constraints are implemented in an accurate sense as bilateral constraints.

The horizontal plane, Body *A*, is defined by its normal unit vector such that radius vector $\mathbf{r}_P = \{x_P, y_P, z_P\}$ of the contact point *P* has to satisfy an equation of the horizontal plane

$$(\mathbf{r}_P, \mathbf{n}_A) = 0. \tag{6}$$

Further denoting the Body *B* current orientation matrix by T_B and by \mathbf{r}_{O_B} its center of mass position vector we

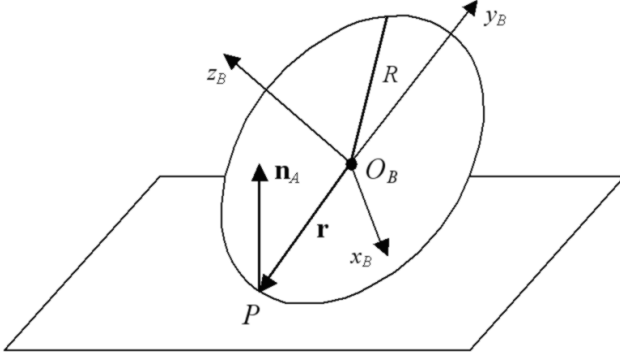


Figure 6: Rolling Disc

obtain the system of three equations

$$T_B \mathbf{r} = \mathbf{r}_P - \mathbf{r}_{O_B} \quad (7)$$

defining the dependence between the vector \mathbf{r}_P and the vector \mathbf{r} of the contact point position in the Body B coordinate system.

On the other hand the vector $\boldsymbol{\tau}$ tangent to the circle at the contact point can be expressed in the disc coordinates as $\boldsymbol{\tau} = \{-y_B, x_B, 0\}$ because the vectors $\boldsymbol{\tau}$ and $\mathbf{r} = \{x_B, y_B, z_B\}$ are to be orthogonal mutually and to be situated in the disc plane permanently. In addition, in inertial system the path vector $T_B \boldsymbol{\tau}$ has to lie in the horizontal plane. Then also holds the condition

$$(\mathbf{n}_A, T_B \boldsymbol{\tau}) = 0. \quad (8)$$

The system of six equations (5), (6), (7), (8) together compose the one w. r. t. six variables $x_P, y_P, z_P, x_B, y_B, z_B$ and implements in a simple and effective way the model `Disc_on_Base` derived from the class `Roll`[1]. Verification of the model outlined above was based on the comparison of its simulation results with ones obtained for the corresponding classic problem defined by the system of ODEs [10]

$$\begin{aligned} \dot{\mathbf{M}} &= [\mathbf{M}, \boldsymbol{\omega}] + m[\dot{\mathbf{r}}, [\boldsymbol{\omega}, \mathbf{r}]] + mg[\mathbf{r}, \boldsymbol{\gamma}], \\ \dot{\boldsymbol{\gamma}} &= [\boldsymbol{\gamma}, \boldsymbol{\omega}] \end{aligned}$$

expressed w. r. t. the Body B rotating system. Here $\mathbf{M} = I\boldsymbol{\omega} + m[\dot{\mathbf{r}}, [\boldsymbol{\omega}, \mathbf{r}]]$ is the vector of the disc angular momentum computed w. r. t. the contact point, $I = \text{diag}(I_{xx}, I_{yy}, I_{zz})$ is the central principal inertia tensor of the disc, $\boldsymbol{\omega}$ is its angular velocity, \mathbf{r} is the vector already mentioned above, $\boldsymbol{\gamma}$ is the unit vector \mathbf{n}_A but expressed w. r. t. the Body B system such that satisfy the relations

$$x_B = -\frac{R\gamma_x}{\sqrt{1-\gamma_z^2}}, \quad y_B = -\frac{R\gamma_y}{\sqrt{1-\gamma_z^2}}, \quad z_B = 0.$$

The simulations showed a high degree of accordance between the two above models of the rolling disc dynamics. Errors increase inevitably and for the vectors $\boldsymbol{\omega}$, \mathbf{M} , $\boldsymbol{\gamma}$ components are of the order 10^{-7} over the time interval of the several hundreds units.

4.2 Model of the wheelset

This model plays an important role when constructing the simplest vehicle models. It is assembled using the considered model of the rolling disc. Visual model of the wheelset depicted in Figure 7, where the Rotate and Flip commands were applied to symmetrize the diagram. Application of the model `FixedIdealJoint` for the joints connecting the wheels and a rod of the wheelset axis is impossible due to the uncertainty for forces acting along this axis. If the contact points with a floor supposed without slipping then introduction of the compliance in the joints is a natural way to avoid the degeneracy mentioned. Making this we add two DOFs to the mechanical system of the wheelset. One else additional DOF has the rod rotating independently about its, and of the wheelset, axis. Compliances are implemented by the model `SpringIdealJoint`.

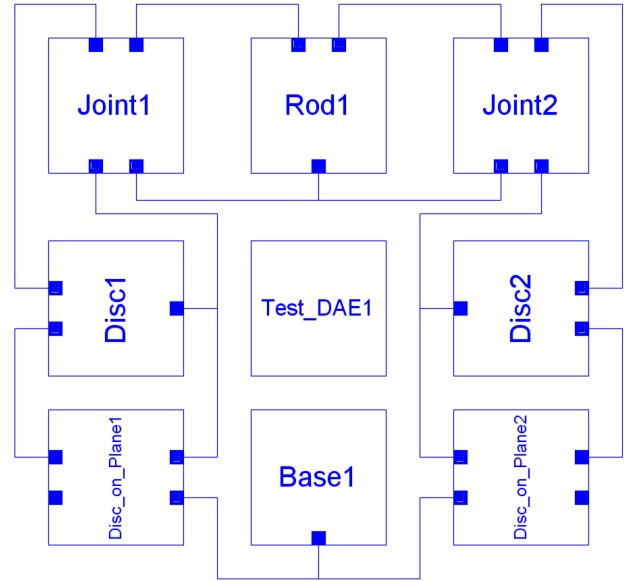


Figure 7: Visual Model of the Isolated Wheelset

To verify the wheelset model built the following system of DAEs was applied

$$\begin{aligned} ma_x &= X_1 + X_2 + X_{\text{ext}}, & ma_y &= Y_1 + Y_2 + Y_{\text{ext}}, \\ ma_z &= Z + Z_{\text{ext}}, \end{aligned} \quad (9)$$

$$a_x = -\frac{R}{2}(\ddot{\phi}_1 + \ddot{\phi}_2), \quad a_y = 0, \quad a_z = \frac{R^2}{2L}(\ddot{\phi}_1^2 - \ddot{\phi}_2^2), \quad (10)$$

$$I_{dz}\ddot{\varphi}_1 = RX_1, \quad I_{dz}\ddot{\varphi}_2 = RX_2, \quad I_{rz}\ddot{\varphi}_r = 0, \quad (11)$$

$$\begin{aligned} \dot{\varphi} [I_{dz}(\dot{\varphi}_1 + \dot{\varphi}_2) + I_{rz}\dot{\varphi}_r] &= \frac{L}{2}(Y_2 - Y_1) - RZ + M_{\text{ext}x}, \\ I_{rz}\ddot{\varphi} &= \frac{L}{2}(X_1 - X_2) + M_{\text{ext}y}, \end{aligned} \quad (12)$$

$$L\dot{\varphi} = R(\dot{\varphi}_1 - \dot{\varphi}_2) \quad (13)$$

which is written w. r. t. moving coordinate system connected with the wheelset according to Figure 8 in an evident way. This system of coordinates performs a convective motion tracing the motion of the rod which plays a role of the wheelset axis shaft.

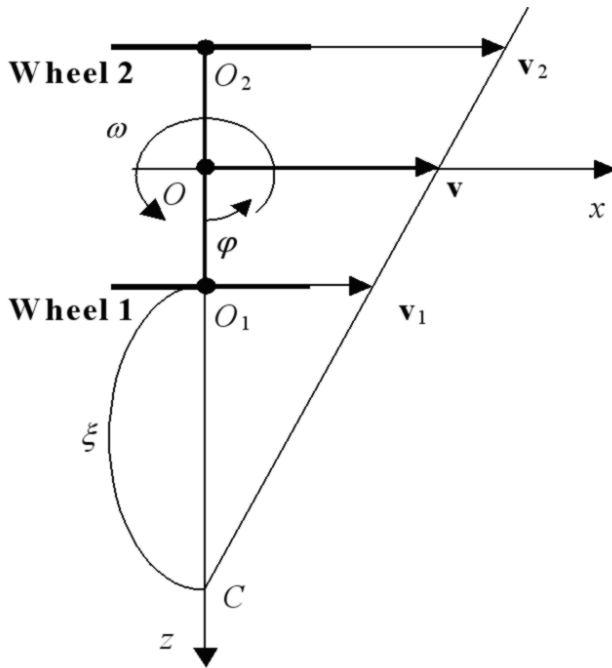


Figure 8: Top View of the Wheelset

The DAE system consists of twelve equations w. r. t. twelve unknowns: $\varphi_1, \varphi_2, \varphi, \varphi_r, a_x, a_y, a_z, X_1, X_2, Y_1, Y_2, Z$. Let us give a more detailed explanations to these DAEs. The subsystem of equations (9) represents the theorem for the center of mass motion of the wheelset. Here $m = 2m_d + m_r$ is the total mass of the wheelset, m_d, m_r are the masses of the wheel simulated by the disc and the rod of the wheelset axis, R is the wheels radius, L is the rod length. The variables a_x, a_z, a_y are correspondingly the tangent, normal, and binormal components of the masscenter acceleration.

The variables X_1, X_2, Y_1, Y_2 are the projections to the x, y axes of the contact forces acting to the wheels from the surface. The value $Z = Z_1 + Z_2$ is used because z -projections of the contact forces can't be computed individually for the reason of degeneration of the problem along the z -axis. This discussed above problem is resolved due to the compliance introduced for the joints.

In the kinematic equations (10) all signs are adjusted such that φ_1, φ_2 are the angles of the wheels relative rotation, $\dot{\varphi}_1, \dot{\varphi}_2, \ddot{\varphi}_1, \ddot{\varphi}_2$ are their relative angular velocities and angular accelerations. Point C is a center of velocities for the rigid planar convective motion of the Ozx coordinate system.

The equations (11) represent z -projections of the Euler dynamic equations for the discs and the rod considered separately. We conclude from the third equation that the rod relative angular velocity is the integral of the motion: $\dot{\varphi}_r = \text{const}$. Remind we consider rotations in the joints as an ideal, without friction, ones.

First of the equations (12) is the dynamical one for the angular momentum of the whole wheelset w. r. t. the axis Ox . The second equation is the projection of the same vector equation to the axis Oy . Further the parameters I_{dz}, I_{rz} are the moments of inertia for the wheel and rod w. r. t. the axis $Oz, I_y = 2I_{dy} + I_{ry}$ where I_{dy}, I_{ry} are the moments of inertia for the disc and shaft w. r. t. the axis Oy . The angle φ is one of the convective rotation about the Oy axis. The kinematic equation (13) is derived from a simple geometric considerations, see Figure 8.

We can add an external force $\mathbf{F}_{\text{ext}} = \{X_{\text{ext}}, Y_{\text{ext}}, Z_{\text{ext}}\}$ and rotating torque $\mathbf{M}_{\text{ext}} = \{M_{\text{ext}x}, M_{\text{ext}y}, M_{\text{ext}z}\}$ to the right hand sides of equations (9), (11), (12). Regarding the equations (11) one can distribute the torque $M_{\text{ext}z}$ between all three bodies of the wheelset in an any desirable way.

Computational experiments show a high degree of concordance between our “physically oriented” model of the wheelset and the ideal model described above if the parameters of stiffness c and damping d in the joint objects of class `SpringIdealJoint` are large enough. Namely, in simulations we have used the values $c = 1000, d = 5000$.

4.3 Model of the vehicle

Let us construct at last a complete model of the snakeboard. Its visual representation see in Figure 9, where rotation and flipping were applied to the graphic images of the objects as it has been done for the wheelset visual model. Similar to the wheelset case we have here a static indeterminacy along the coupler axis if one supposed a rigid body. To avoid this degeneration we splitted it into two equal parts and connected them via viscoelastic joint, with an axis along the coupler, using the model `SpringIdealJoint` with the stiffness and damping large enough for the longitudinal compliance of the snakeboard.

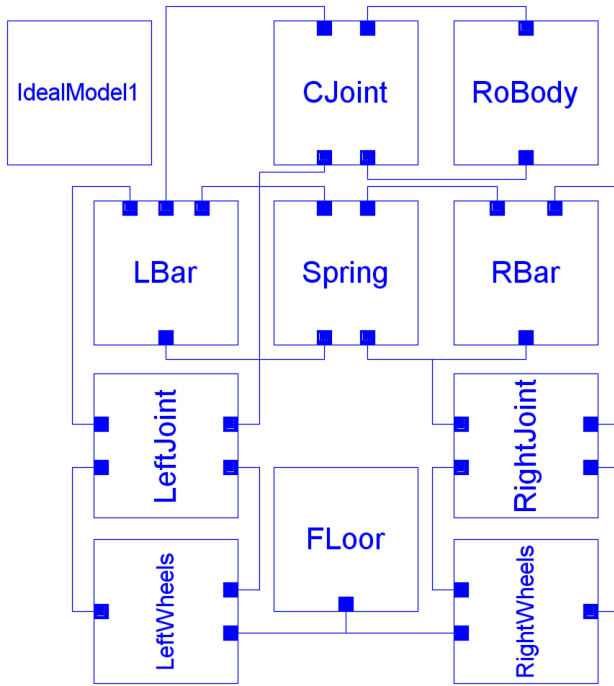


Figure 9: Visual Model of the Snakeboard

To perform a comparison with the known results [9] three servoconstraints were introduced to the model. These servoconstraints imitate the control of the robot-snakeborder and are implemented by the `FixedServoJoint` class which defines a relative rotation of the bodies by the prescribed angle. To be more precise in the class mentioned the control is given by a law of the relative acceleration with a proper initial values of the angle and the angular velocity.

Servoconstraints are mounted at the joints between the coupler and the wheelsets, and between the flywheel and, for definity, the left part of the coupler. The joints mentioned correspond to the objects `LeftJoint`, `RightJoint`, and `CJoint` in Figure 9. All three servoconstraints can be described by the equations

$$\begin{aligned}\varphi_f &= a_f \sin(\omega_f t + \beta_f), & \varphi_b &= a_b \sin(\omega_b t + \beta_b), \\ \psi &= a_\psi \sin(\omega_\psi t + \beta_\psi),\end{aligned}$$

where φ_f , φ_b are the angles of the front and rear (back) wheelsets relative to the coupler rotation correspondingly, ψ is the angle of the flywheel rotation w. r. t. the coupler, to be more exact relative to its left (rear) part, the object `LBar` in Figure 9, a_f , a_b , a_ψ are the corresponding amplitudes of libration, ω_f , ω_b , ω_ψ are their frequencies, and β_f , β_b , β_ψ are their initial phases.

According to [9] three types of the snakeboard gait were under verification:

1. “drive”: $a_b = -a_f$, $\omega_f = \omega_b = \omega_\psi$;
2. “rotate”: $a_b = -a_f$, $2\omega_f = 2\omega_b = \omega_\psi$;
3. “parking”: $a_b = -a_f$, $3\omega_f = 3\omega_b = 2\omega_\psi$.

The simulations results showed a full coincidence of the gait types for our regularized model and the idealized model of the paper [9]. All types of the behavior are demonstrated in Figures 10, 11, 12, where the flywheel masscenter projections to the xz plane are presented. In [9] for the ideal model when deriving the DAEs of the snakeboard motion for simplicity of the model the wheels rotary motion and the wheelsets translatory motion weren’t taken into account. In such a sense from the dynamical point of view our model is more complete.

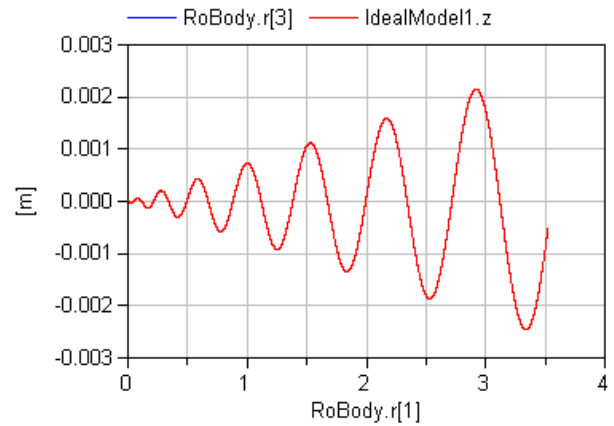


Figure 10: Masscenter Trajectory for “Drive” Gait

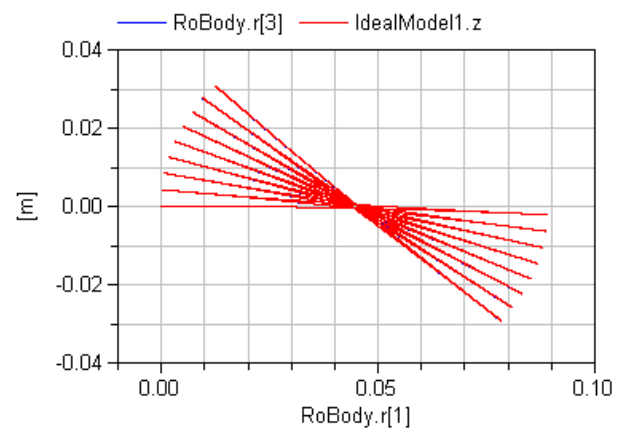


Figure 11: Masscenter Trajectory for “Rotate” Gait

If we introduce a small parameter playing a role of the scaling multiplier for the inertia moments and masses for the motion types neglected in [9] then if its value is small enough, 10^{-7} for our simulations, the motions compared become practically indistinguishable,

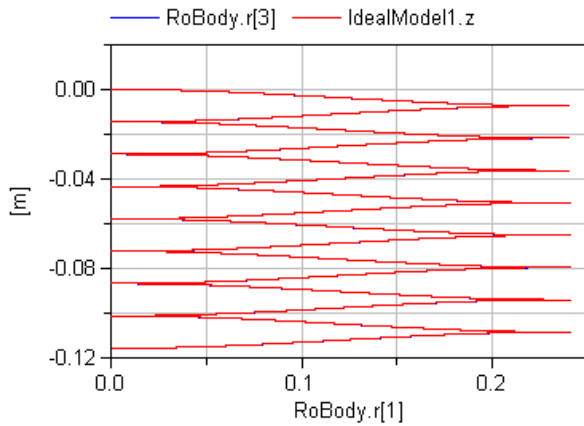


Figure 12: Masscenter Trajectory for “Parking” Gait

see for instance the plot of the snakeboard masscenter x-components difference for the models under comparison, Figure 13, in the case of the “Rotate” gait. Animation shot in the case of the ”drive” type gait see in Figure 14.

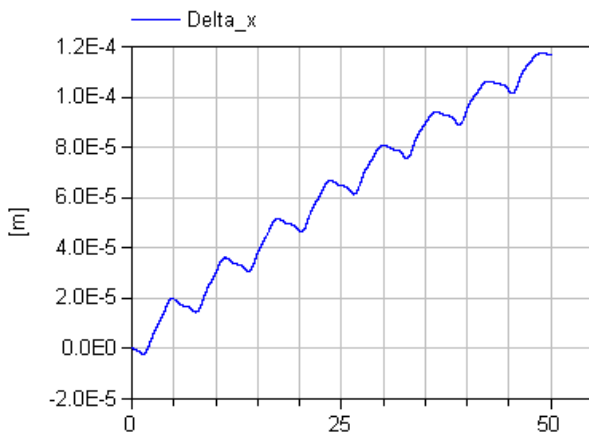


Figure 13: Closeness of Models

A set of different laws of the snakeboard control performed by the robot-snakeborder generating the mass-center trajectories like astroid, cycloid, eight, 3-rose, 4-rose is presented in [11]. The port-controlled Hamiltonian representation of the simplified ideal snakeboard model from [9] with its bond graph implementation is investigated in [12].

Considering balance of energy in the total model one can remark here that servodrives applied between the coupler on one side and the flywheel and wheelsets on the other one are implemented correspondingly in the objects

```
CJoint, LeftJoint, RightJoint
```

of the class `FixedServoJoint`. Such kinematic constraints are known in the bond graph theory to be able

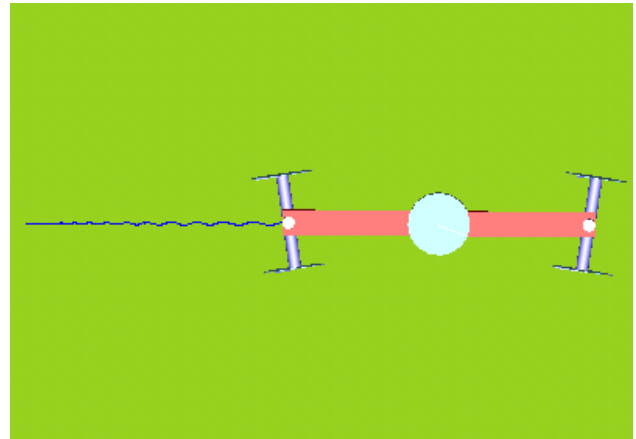


Figure 14: Animation of “Drive” Gait

to inject into the system any amount of energy needed to hold the desired motion. On the other hand energy loses due to the resistance elements encapsulated in the objects

```
Spring,
LeftWheels.Joint1, LeftWheels.Joint2,
RightWheels.Joint1, RightWheels.Joint2,
```

of the class `SpringIdealJoint`.

Thus the class `FixedServoJoint` implements two scalar FC-elements from the general bond graph depicted in Figure 2 for the rotary and translatory motions, while the class `SpringIdealJoint` implements one C-element, ideal elastic compliance, in combination with R-element, resistance due to viscosity, for the translatory motion plus one EC-element for the rotary motion. Remind that all motions supposed here as a relative ones of the body *B* w. r. t. body *A* in each of the constraint objects considered.

5 Conclusion

We can make now our brief list of conclusions in the following form:

- A unified multi-bondgraph representation of the MBS dynamics in a sufficiently simple way with the canonical junction structure is possible.
- The representation depicted in Figure 2 can be used as a guideline to construct the consistent system of DAEs in a systematic way. In other words we can say that multi-bondgraph constructs like ones of Figure 2 are to be used as a regular basis for more informal object-oriented approach.

- An object-oriented representation makes it possible to develop the constraints models adopted to the specific types of the bodies interconnections in a fast and effective manner implementing the corresponding bond graph formalisms in a more natural and informal way mainly by chains of inheritance for the behavior (equations) and properties thus gradually filling the complete multi-bondgraph description.
- An acausal modeling accelerates the modeling releasing a developer from the problem of causality assignment if s/he takes into account some requirements like complementarity rules.
- Introducing the compliance into the model may be useful and effective preserving the principal properties of the MBS like anholonomy etc.

Enumerate also some possible directions of the further work:

- Development of the vehicle models more complicated then considered here.
- Development of the more complicated contact models taking into account friction and a unilateral nature of the constraints.
- Account of the road uneven surfaces of different types.

6 Acknowledgement

The paper was prepared with partial support of Russian Foundation for Basic Research, projects 05–08–65470, 05–01–00454, SS–6667.2006.1.

References

- [1] Kossenko, I. I., and Stavrovskaja, M. S., How One Can Simulate Dynamics of Rolling Bodies Via Dymola: Approach to Model Multibody System Dynamics Using Modelica // Proceedings of the 3rd International Modelica Conference, Linköpings universitet, Linköping, Sweden, November 3–4, 2003, pp. 299–309.
- [2] Kossenko, I. I., Implementation of Unilateral Multibody Dynamics on Modelica // Proceedings of the 4th International Modelica Conference, Hamburg University of Technology, Hamburg–Harburg, Germany, March 7–8, 2005, pp. 13–23.
- [3] Stramigioli, S., Blankenstein, G., Duindam, V., Bruyninckx, H., Melchiorri C., Power Port Concepts in Robotics. The Geometrical–Physical Approach. Tutorial at 2003 IEEE International Conference on Robotics and Automation. — IEEE, 2003.
- [4] Paynter, H. M., Analysis and Design of Engineering Systems. — The M. I. T. Press: Cambridge, Massachusetts, 1961.
- [5] Cellier, F. E., Continuous System Modeling. — Springer–Verlag: New York, 1991.
- [6] Mukherjee, A., Karmakar, R., Modelling and Simulation of Engineering Systems through Bondgraphs. — Alpha Science International Ltd.: 2000.
- [7] Golo, G., Interconnection Structures in Port-Based Modelling: Tools for Analysis and Simulation. PhD Thesis. — University of Twente: Enschede, The Netherlands, 2002.
- [8] Dymola. Dynamic Modeling Laboratory. User’s Manual. Version 5.3a. — Lund: Dynasim AB, Research Park Ideon, 2004.
- [9] Lewis, A. D., Ostrowski, J. P., Murray, R. M., and Burdick, J. W. Nonholonomic Mechanics and Locomotion: The Snakeboard Example // Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, May 1994, IEEE, pp. 2391–2400.
- [10] Borisov, A. V., Mamaev, I. S., Kilin, A. A., Dynamics of Rolling Disk // Regular and Chaotic Dynamics, 2003, Vol. 8, No. 2, pp. 201–212.
- [11] Golubev, Yu. F., Motion Design for a Robot-Snakeboard // Keldysh Institute of Applied Mathematics, the Russian Academy of Science, Preprint No. 65, 2004.
- [12] Duindam, V., Blankenstein, G., Stramigioli S., Port-Based Modeling and Analysis of Snakeboard Locomotion // Sixteenth International Symposium on Mathematical Theory of Networks and Systems, Katholieke Universiteit Leuven, Belgium, July 5–9 2004.

NowaitTransit[®] concept assessment.

Modeling of trains on complex track geometry

Jan Tuszyński, Niklas Philipson, Johan Andreasson, Magnus Gäfvert
Nowaittransit AB, Modelon AB

jan@nowaittransit.com, niklas.philipson@modelon.se, johan.andreasson@modelon.se, magnus.gafvert@modelon.se

Abstract

This paper presents modeling and verification of Nowaittransit's concept of mass passenger transportation for big cities. Cars of the train, coupled in a closed loop, move continuously along the track but slow down in station areas due to the special scheme of car folding. Concept verification through modeling was requested by the investors.

Keywords: Mass city transportation; Train dynamics modeling; Vehicles in complex constraints; Investment in new technology

1 Introduction

1.1 Background

Nowaittransit AB has developed and patented new concept of mass passenger transportation for big cities. The closed loop of interconnected cars moves continuously along the track slowing down in station areas by the special scheme of folding cars.

The concept has advantages of high transport capacity of modern subway but at much lower costs for investment and exploitation providing that way attractive alternative to subway. The company has several Asian cities interested, but closing final contracts requires formal assessment proving that proposed functionality of the system can be achieved. The assessment is run presently in two stages; through computer modeling and through physical verification on the test track in Sweden. The computer modeling reported here was performed by Nowaittransit and Modelon and had two main objectives:

To prove that the concept has no "show stoppers" which may be hidden in dynamics of the long chain of the cars moving along the complex rail geometry. Requirements on the orderly start-up and braking shall be achieved and potential hazards identified.

To show that design team of the Nowaittransit has capacity and tools to handle complexity of mechanical and control systems of the train.

The interesting aspect of the project presented was that modeling was required by investors as a proof that a new concept is trustworthy enough to invest in the test track.

1.2 Short introduction of the NowaitTransit[®]

The underlying principle of the NowaitTransit[®] invention is a continuous train movement with a reduction of the traveling speed at the stations.



Figure 1: Nowaittransit mass transportation system

The length of the *NowaitTransit[®]* car is reduced through a 90-degree horizontal folding shown in Figure 1. The cars traveling out of the station areas have a normal transport speed, which during the folding is reduced by the factor 1:12 when the cars enter the station. The passengers can now board and disembark the train at the end of each slowly moving car. This speed at the station corresponds to the high end of normal walkway speeds. The passengers' entry and exit is further facilitated through use of moving walkways, making it acceptable for disabled persons.

Due to this continuous boarding/disembarking principle, station loads will be evenly distributed with no crowds of passengers waiting. Traditional systems have passengers gathering on the platforms and whole trainloads of people boarding and disembarking simultaneously. Accordingly NowaitTransit® stations can be smaller with capacities of stairs, elevators and stairways reduced. Small stations, light modular structures and simplicity allow reduction of investment costs to approximately 25 MEUR/km compared to 85-125 MEUR for conventional subway.

2 Objectives of the modeling and tests performed

2.1 Objectives

The main objective for modeling was proving that the concept can be realized. We concentrated accordingly on the following:

- Analysis of train structural singularity or system over-determination.
Mainly aspects of the train dynamics where movements of the interconnected cars are bound to 3D geometry of the rail
- Structural loads on the train components.
Load analysis was required to prove that the train can be accelerated and slowed down, and that the main train structures will hold intact through pre-identified hazard situations
- Identification and evaluation of factors of passenger comfort and train controllability

Assessment of the *NowaitTransit*® concept reported here should be seen as the final stage of the concept introduction to investors. The Modelica models will be further developed and verified to become eventually design tool of the full scale projects of the future.

2.2 Requirements on tests and introduction to main models developed

The objectives above were addressed by specifying simulation experiments required, which led to the final requirements on train models.

A couple of cars moving on rail of free definable geometry and coupled by a single distance beam was found the basic structure of the model. That structure was pretty straight forward to model, but it showed up quickly that the main problems came from the

long chains of cars acting on each. Long chains meant huge number of equations to solve and difficulties in defining initial conditions, which led to extremely long simulation times. We had to simplify the whole train model worked accordingly with two main types of models:

Centre Rail Model (CRM): where the cars, which travel on the central rail can turn round z-axis vertical to the rail line. The cars can now be forced into folding and un-folding according to angle α , pre-define function of car position (s) on the track. This scheme replaces here car-turning forced by changing of the rail gauge of the original concept.

CRM model will not allow studies of the rail boggy interaction but is a good approximation for studies of car propulsion and car interaction in open and closed loops of coupled cars covering at least one car transition zone (e.g. station of folding zone, station platform zone and un-folding zone).

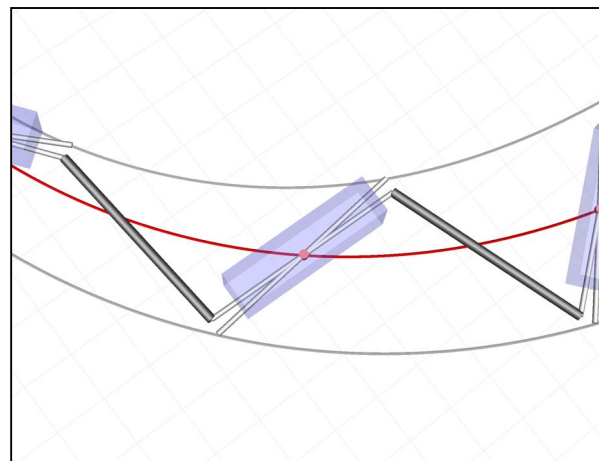


Figure 2: Cars and distance beams of the CRM model

Figure 2 shows part of the CRM animation, with middle (red) line representing centre rail, and two external lines showing trajectory of wheel-rail contact points.

Final Rail Model (FRM): where at least one basic car structure is run through various rail zones linked

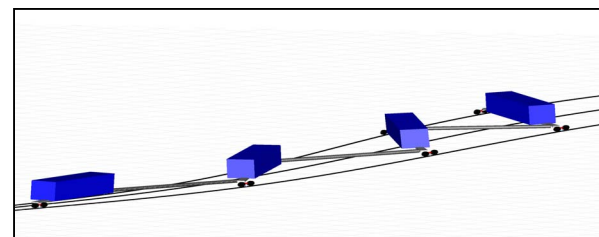


Figure 3: Two car couples ascending rail of FRM

with rest of the train represented by CRM. This model allows us study of car wheel interaction with

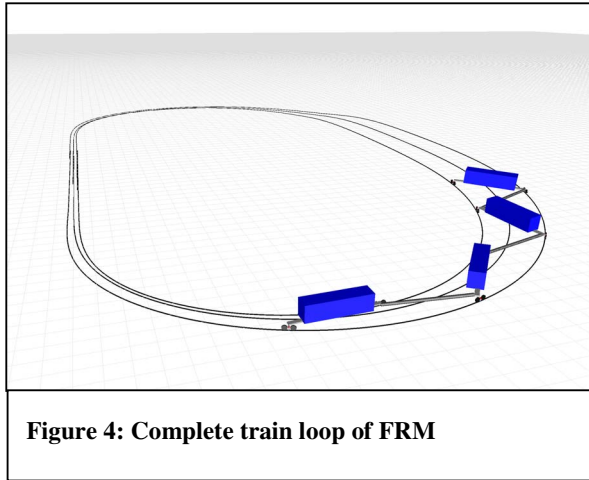


Figure 4: Complete train loop of FRM

the rail. Figure 3 shows FRM of the cars ascending rail, while Figure 4 shows complete rail loop of the same simulation.

3 Selected aspects of Nowait models

3.1 General

We present here the following aspects of Nowait-Transit® modeling:

- Tools for selection and testing of track geometry
- NowaitLib: Modelica library for simulation of train cars on the complex track geometry
- Car and rail components of the library

3.2 Generating track geometry

As it was already implied train of Nowait cars is basically over-determined. Due to the folding and unfolding schemes length of the closed loop of the train can vary. It becomes accordingly crucial to match dimensions of car components and car geometry to reduce those length variations until they could be absorbed by elasticity of train components and rail.

This complex process of selecting car geometry was facilitated by development of the package of Matlab functions, transforming input data (e.g. lengths of the zones) into output matrix 'trackTab' with description of rail geometry. The 'trackTab' is read as a table of Modelica models.

Two basic forms for track building are available:

- Polynomial, of continuous 1st, 2nd and 3rd derivatives

- Exponential, according to function $y = C \cdot \exp(-|s-s_0|^n \text{Exp})$

Exponential form allows symmetrical track geometry only, and has a convenient property of continuous derivatives. The track geometry may be varied in x-, y- and z-direction of the world.

Changes in z-direction (altitude) are required mainly for slowing down / acceleration of the folding / unfolding cars by transfers between kinetic and potential car energy. This z-compensation can be complete (100% kinetic energy at normal train velocity transferred to the potential energy) or partial only. Examples of the polynomial and exponential geometry are shown in Figures 5 and 6, presenting car folding angle, velocity and elevation of both cases.

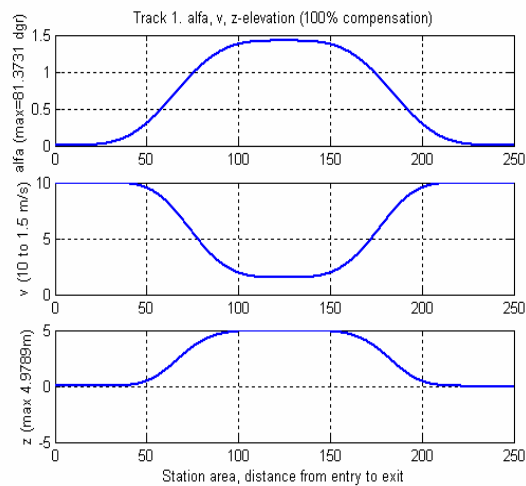


Figure 5: Profiles of car folding angle, velocity and z-elevation for exponential track geometry

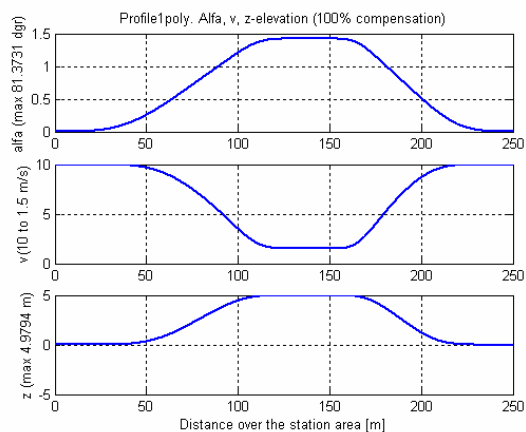


Figure 6: Profiles as car angle, v and z, for polynomial track geometry

Matrix 'trackTab' is used for centre and final rail models. It can be noted from Figure 2, that required geometry of each rail (upper and lower lines) can be

unequivocally decided from centre line geometry, dimensions of car components and angles of the car in relation to car centre. All this information is available in matrix 'trackTab'.

3.3 Testing of track geometry

Matlab-generated track geometry must be tested. Initial testing is done using special Matlab functions, the final testing through Modelica models. There are three criteria which can be evaluated in Matlab: 1) length of the train between any two cars holding the same angle shall be constant, 2) number of cars between any two points of the track shall be constant, and 3) velocity of moving cars shall comply with constant car passing frequency (number of cars / sec).

Results from testing track geometry of Figure 6 are shown in Figure 7.

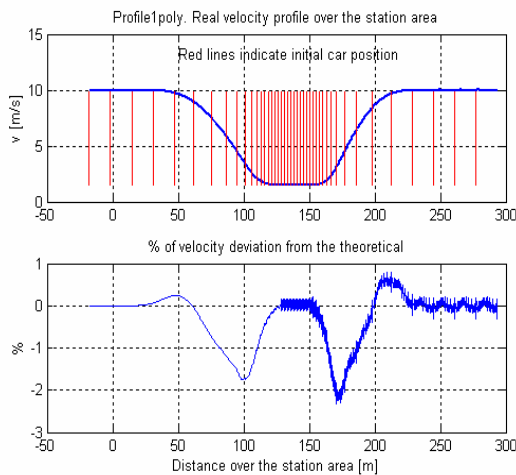


Figure 7: Testing results of polynomial geometry of Figure 6

The testing showed positive results concerning third criteria; final velocity variation was contained in approximately +/- 0.1 %. The vertical lines on the upper plot of Figure 7, indicate initial position of cars in the transition zone. Vector of initial positions is used for initialization of Modelica simulations.

3.4 NowaitLib Library

All modeling *NowaitTransit*[®] trains follow formal Modelica instructions, considering initial development of the system library including formal verification of library modules. The name of the library is NowaitLib and library structure is shown in Figure 8.

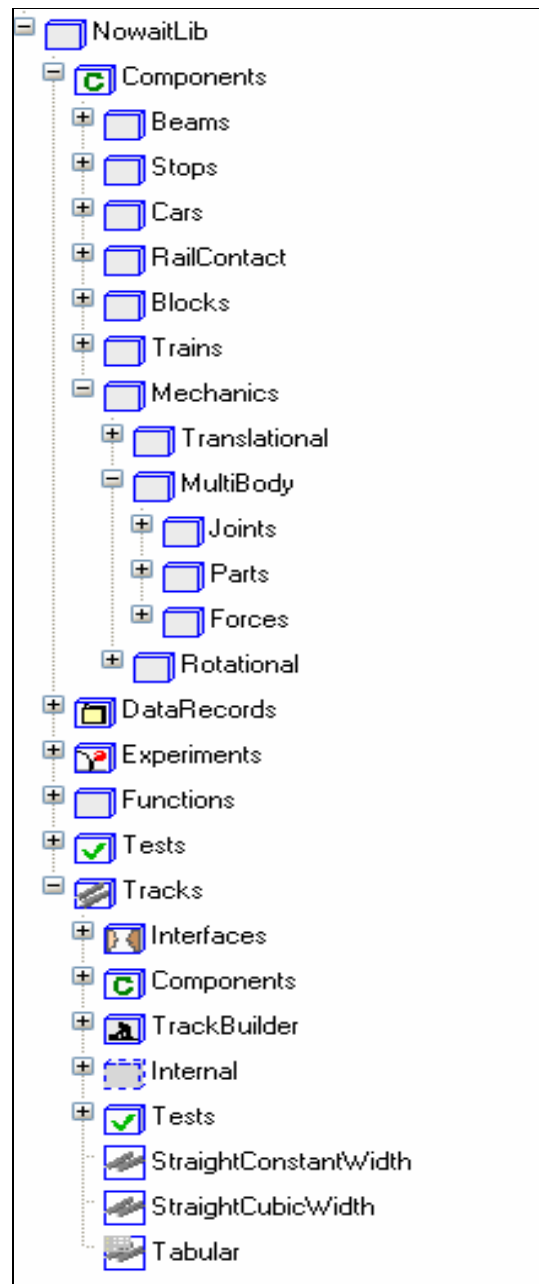


Figure 8: Structure of the NowaitLib library

The components of the library are universal in a sense they can be applied to various kind of bodies moving in boundaries of any track geometry defined in pre-defined table (here 'trackTab')

3.5 Main models of the NowaitLib

Main train components modeled are cars, distance beams for car coupling and finally track/rail elements. Models of the cars and beams built on the standard Modelica Mechanics of MultiBody Library.

One of the main problems encountered was placing free body of the car in constraints of the track.

generalizing modification of the MultiBody.Joints.Internal.Pismatic joint where $r_b = n*s + r_a$. The following modifications are

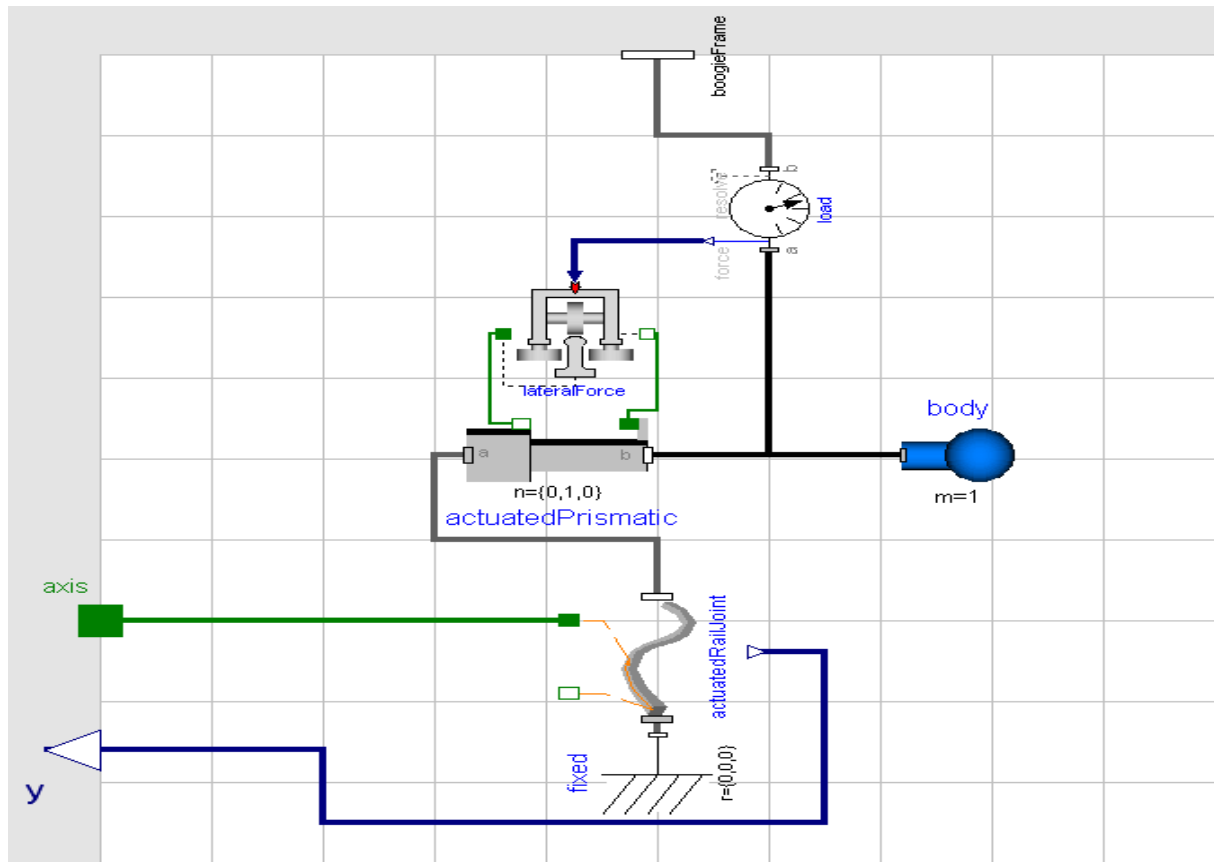


Figure 9: Main components of the track - car interconnection

Figure 9 shows main components of the Contact-Point model used in FRM for interconnection of the wheel contact point (CP) and rail, where:

- CP of the wheel is connected through MultiBody frame 'bogieFrame'
- ActuatedPrismatic component models lateral (y-direction) movements of the rail
- LateralForce component calculates forces acting on the CP (and car bogie), there mainly centering force of wheel crowning, and friction
- Track geometry is enforced through 'actuated-RailJoint' getting 'trackTab' data through 'axis' connector.

'RailJoint' model describes the (translational) motion along a track defined in space by vectors of frames 'a' and 'b' (Figure 10), as $r_b = r(s) + r_a$. It can essentially be considered as a

essential for train modeling:

- The track is able to follow any (two times differentiable) trajectory in space given via the outer function 'track.position'. This means that also the orientation of 'frame_b' relative 'frame_a' must be given (for the prismatic it is assumed that there is no relative rotation), here via the outer function 'track.orientation'.

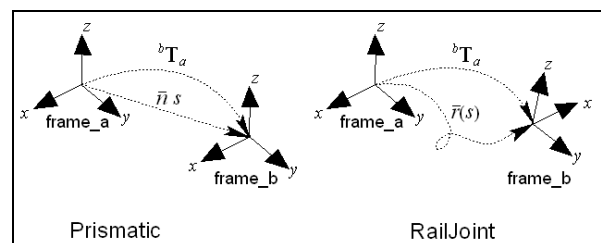


Figure 10: RailJoint as extension of Prismatic joint

- To allow efficient symbolic manipulation of the functions ‘track.position’ and ‘track.orientation’, the differentiations are provided through ‘track-Tab’ table. Note especially that for this application, it is assumed that dr_{rel}/ds points in the x-direction of frame_b. The differentiation has to be possible at least three times to resolve the motion equations on acceleration level of both rotational and translational loops.

As a result, the relative velocity and d’Alembert’s principle are always in the heading direction dr/ds which is the x-axis of frame_b. This model is extended as MultiBody.Joints.Internal.Prismatic to allow addition of Translational flanges.

4 Summary of experiments run on the models

The experiments run on the models covered three main groups:

- Tests/generation of the track geometry and dimensioning of cars and car coupling elements
- Tests of the longitudinal forces acting between cars in dynamic situations of train runs, start-ups, and different cases of braking down
- Tests of complete train sets including bogie and wheels moving along the complex rail geometry.

The first group was run through package Matlab functions as reported already above. The Matlab results were confirmed by Modelica/Dymola experiments.

Tests of the longitudinal forces required models of the long chain of cars and were run accordingly on the centre rail models.

Tests of complete train sets required simulations on the final rail models.

All tests were run on the ‘representative track geometry’, covering mainly two basic profiles; one of complete station zone, and second of 90° curve with cars folding and un-folding in order to manage curves of relatively small radii required for adoption of the train track to existing city environment.

4.1 Experiments run on the centre rail model

Experiments of this group concentrate on demonstration of forces between cars, and forces required to drive/brake cars through raising and falling parts of

the track ($\delta z/\delta s$), and cars on the curves in x-y plane. The following groups of experiments were run:

- Cars run in constant base velocity,
- Cars of the starting train; accelerated from $v=0$ to base v ,
- Cars of the braking train; slow down from base v to 0.
- Cars in “let free” situation, cars are left over without any external forces testing car behaviour on the raising and falling part of the track
- Train emergency brake down (according to international standards)
- Train in accidental brake down (i.e. hazard)

In addition to the above tests concerning mainly identification of the dynamic behaviour of the NowaitTransit train, the following tests were added:

- Test of passenger comfort. Comfort factors were selected measuring forces acting on passengers.
- Test of car controllability (only initial evaluations)

4.2 List of experiments run on the final rail model

Experiments run on the final rail model were essentially the same as for CRM. The focus of testing was moved here on the following:

- Verification of compatibility between CRM and FRM (does CRM reflect real behavior of cars on the variable gauge track?)
- Verification of the wheel movement on the rail. E.g. prove that wheel friction across the rail will limit lateral sliding of the wheel
- Verification of train controllability during selected phases of the train operation. This to estimate complexity of the train control system (Note: here identify potential problems only)

5 Examples of experiments run on the centre rail track

Experiments listed above are exemplified here on three case only:

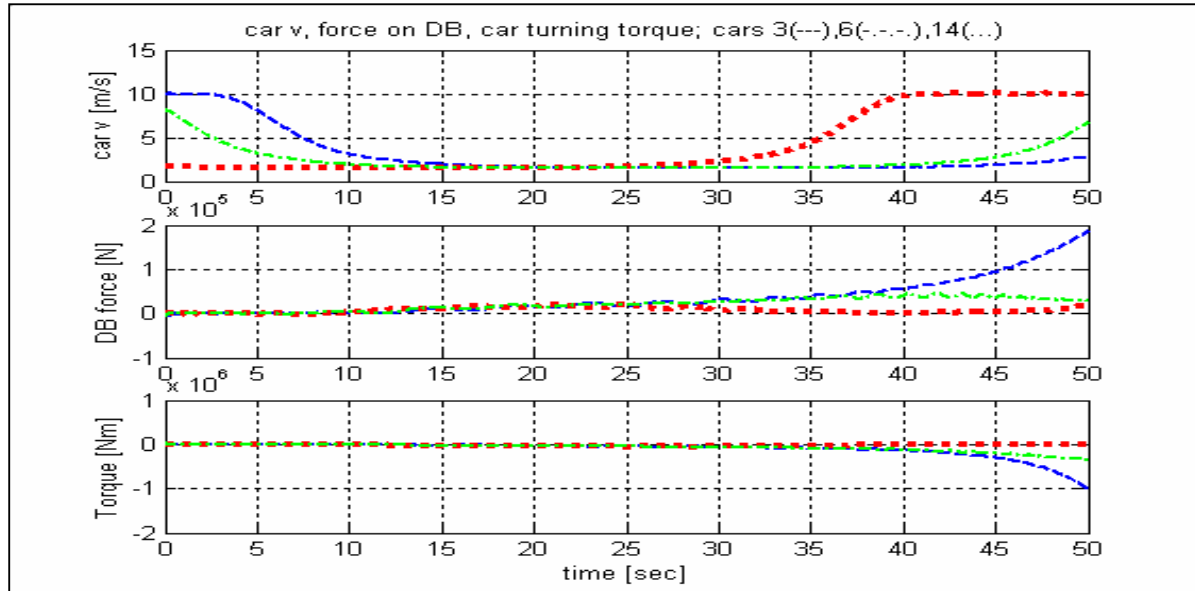


Figure 11: Cars of the CRM in normal operation passing station area

Case 1 (Figure 11): shows, velocities of cars, forces on distance beams and car turning torques during a normal car passage of the station area. Note that the case reflects the special situation when cars in transition are not propelled but only pushed through by the cars still on straight part of the track

Case 2 (Figure 12): shows the same car configuration as for case 1. Here cars are slowed down from 5 to 0 m/sec. This case is special as well as cars in transition are braked but slowing down of the cars on straight part of the track. Note pulsing character of forces and torques implying that control is required.

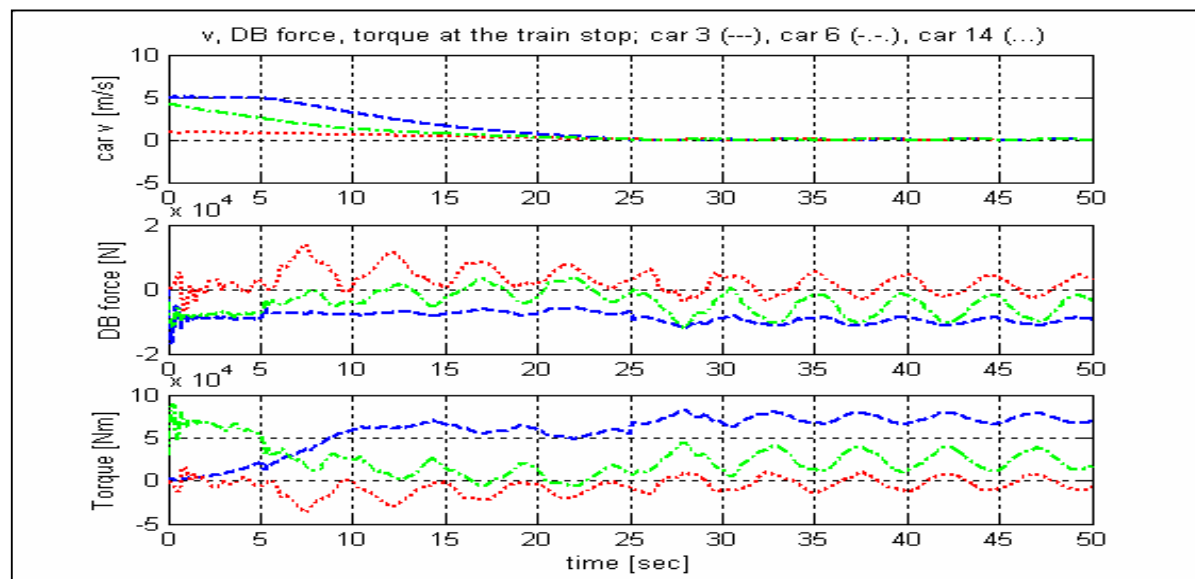


Figure 12: Cars in CRM during train stop from 5 to 0 m/sec

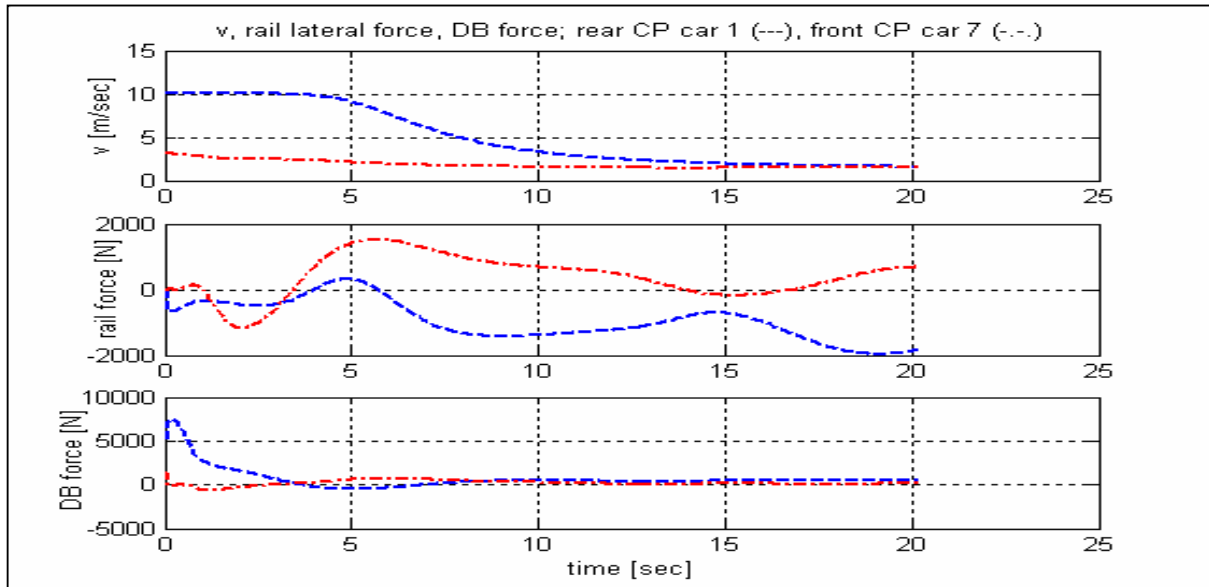


Figure 13: Cars of FRM during normal operation in station area

Case 1 (Figure 13): shows, velocities of cars, lateral forces on the rail and forces in distance beam acting on the selected cars of the train during normal passage of the station area.

6 Summary of results and conclusions

The study presented here had the main purpose to show for investors deep knowledge of the proposed NowaitTransit® concept. The study resulted accordingly in the models which allow ‘driving’ Nowait train along various track geometry. The study had the ambition to identify the process of driving long trains along complex track geometry, which implies that the modeling effort shall be continued. We can tell today that no distinct ‘show stoppers’ were identified, but at the same time we see difficulties to be met. There are clear tendencies to longitudinal car oscillations. Main effort of the studies was to generate track geometry reducing those oscillations to the minimum, which showed up feasible. The coming modeling stages should concentrate on design and verification of the train propulsion system, on finding optimal algorithms for train control (done partially in ref [1]) and on verification of the auxiliary systems for train start-up and brake-down. We are pretty advanced in further model development allowing study of the complete loops of the interconnected cars (Figure 14). Modelica models and package of Matlab functions are already powerful tools but must be developed further to ensure that train design and design verification will be effective and trustworthy.

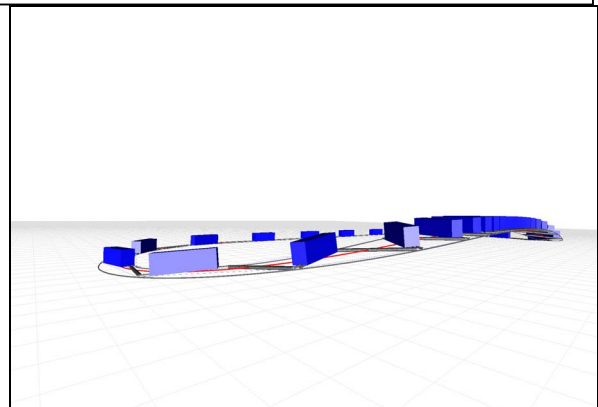


Figure 14: Animation of the complete closed loop of Nowait train model

7 References

- [1] Dynamic Simulation of the Train Concept Nowait Transit. Master Thesis, Daria Madjidian, Arash Majedi. 2005. Department of Automatic Control. LTH

Session 3a

Thermodynamic Systems for Energy Storage and Conversion

Analysis of steam storage systems using Modelica

J. Buschle, W.D. Steinmann, R. Tamme
 German Aerospace Center (DLR), Institute of Technical Thermodynamics
 Pfaffenwaldring 38-40, 70569 Stuttgart, Germany
 jochen.buschle@dlr.de

Abstract

Modelica is used for the analysis of steam accumulators used as energy storage systems in power plants and process industry. The analysis includes varying pressure accumulators and steam accumulators with latent heat technology. Physical models for the phase change material (PCM) and for the vertical discretisation of stacked volume elements are implemented in Modelica. Modelica is used for the analysis of new PCM enhanced steam accumulators which are not state of the art. The results of this analysis help to design these novel storage systems.

Keywords: thermal energy storage; steam accumulator; phase change material

1 Introduction

Industrial process heat applications have been identified as a promising new area of application for thermal energy storage systems. Storage systems improve the efficiency by the reuse of energy in cyclic processes. The bulk of process heat applications require steam at pressures between 1 and 20 bar with corresponding saturation temperatures between 100°C and 210°C. While the application of phase change materials (PCMs) is straightforward for isothermal energy storage, no commercial system is available in this temperature range today.

Modelica is used to analyse different kinds of steam storage systems for applications in power plants and process industry. The analysis includes varying pressure steam accumulators, which are state of the art, and a novel kind of steam accumulator with a phase change material (PCM). For the various storage systems physical models are implemented in Modelica.

2 Steam accumulator

Due to increasing costs for fossil energy, systems for thermal energy storage have become attractive for process heat applications. Especially in cyclic processes, energy storage systems offer an additional option for efficient energy usage. An example for such a cyclic process is the production of gas concrete. Here the produced stones are hardened under a steam atmosphere in an autoclave. During a hardening cycle, the pressure in the autoclave is raised to 15 bar for several hours. Between two production cycles, the steam is partially stored in varying-pressure accumulators (Figure 1), so called Ruths steam accumulators, which represent the current state-of-the-art technology in medium temperature thermal energy storage.

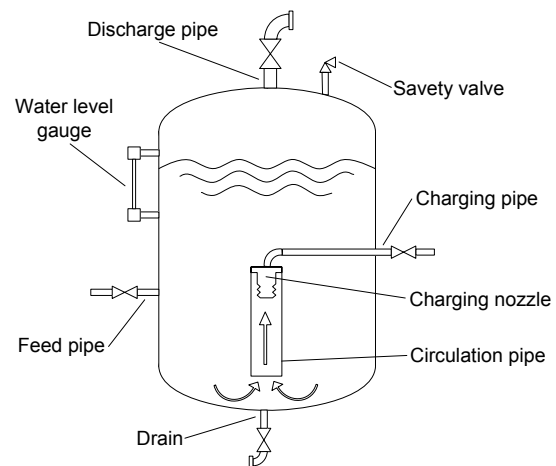


Figure 1: Varying pressure accumulator [1]

Varying-pressure accumulators use hot pressurised water as storage medium. Liquid and gas phases are in thermodynamic equilibrium. During the discharge process, the system provides saturated steam at decreasing pressure.

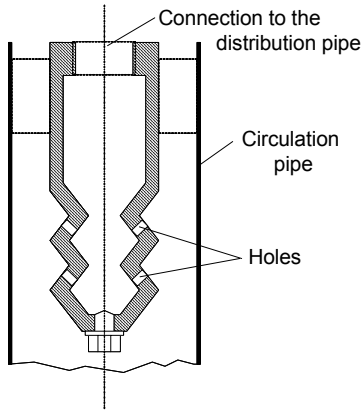


Figure 2: Charging nozzle with circulation pipe [1]

During the charging process, steam is blown into the liquid contained in the accumulator. The incoming steam bubbles condense in the liquid or pass into the steam space, depending on the thermodynamic equilibrium in the vessel. The bubbles which rise to the steam space increase the pressure and lead to a higher saturation temperature, so that the next bubble might condense. To use the entire storage content, the charging process requires circulation. Ruths invented a method that consists of nozzles (Figure 2) which turn the flow of steam upwards. The nozzles are surrounded by a circulation pipe, wherein the water flows upwards. The minimum temperature loss is composed of the difference between the steam space and the uppermost liquid layer as well as the difference between the saturation temperatures due to the additional pressure of the water at lower depths. Depending on the accumulator pressure and the steam intake, there is a certain depth for the nozzles which minimizes the overall temperature loss. [1]

3 MultiPhase Media Library

The **Modelica.Media** package provides a standardized interface to fluid media models and specific media models based on this interface. For the handling of multiphase applications, a general media package was developed based on the **Modelica.Media** package. An additional properties record called **MultiPhasePropertiesRecord** is used in the **BaseProperties** model to provide the thermodynamic properties of the different phases to the models using the **Modelica.Media** package.

```

replaceable record MultiPhasePropertiesRecord
  extends Modelica.Icons.Record;
  MassFraction x[nPhase];
  SpecificVolume v[nPhase];
  MassFraction[nPhase, nX] X;
  MassFraction[nPhase, nXi] Xi;
  SpecificEnthalpy h[nPhase];
  SpecificInternalEnergy u[nPhase];
  SpecificHeatCapacity R[nPhase];
  MolarMass MM[nPhase];
end MultiPhasePropertiesRecord;

```

In this record, the thermodynamic properties of the different phases of the medium are included. Compared to the **BasePropertiesRecord**, as defined in the **PartialMedium** package of the **Modelica.Media** library, the mass fraction of the different phases is added and the specific volume is used instead of the density. The **PartialMultiPhaseMedium** package extends the **PartialMedium** package of the **Modelica.Media** library. The **BaseProperties** model in the **PartialMultiPhaseMedium** package is extended in the following way.

```

redeclare replaceable partial model extends BaseProperties
  "Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a medium"

  MultiPhasePropertiesRecord MultiPhase;

equation
  /*
  d = 1/(MultiPhase.v*MultiPhase.x);
  for i in 1:nX loop
    X[i] = MultiPhase.X[:, i]*MultiPhase.x;
  end for;
  for i in 1:nXi loop
    Xi[i] = MultiPhase.Xi[:, i]*MultiPhase.x;
  end for;
  h = MultiPhase.h*MultiPhase.x;
  u = MultiPhase.u*MultiPhase.x;
  R = MultiPhase.R*MultiPhase.x;
  MM = MultiPhase.MM*MultiPhase.x;
  */
end BaseProperties;

```

It is possible to use the **MultiPhase** medium models with single phase models. The required base properties are calculated as a mixture of the different phases.

4 Downflow model

For the analysis of steam accumulators, a vertical discretisation of the storage vessel is elemental. Based on the **Modelica.Fluid** library, models for the simulation of multiphase flows are developed. The drainage and flooding of vessels can be reproduced as well as the vertical stacking of control volumes.

Between two volume elements stacked one above the other, two kinds of mass exchange mechanisms are identified. One type of mass exchange results from the pressure difference between these two elements and the other mass exchange occurs due to down-flowing water and up-flowing steam caused by the density difference of these two phases. The mass flow of the downflowing water is calculated in the following way:

```

if noEvent(y_liq > y_min and y_vap > y_min) then
  m_flow= V_upper*V_lower*f_flow*
    (y_liq - y_min)*(y_vap - y_min);
else
  m_flow = 0.0;
end if;

```

The mass flow of the upflowing steam is not calculated separately. This mass transport mechanism is covered by the mass transport due to the pressure difference. The required volume fraction, y , can be calculated from the **MultiPhase Modelica.Media** models in the following way:

```

y = medium.MultiPhase.v[nPhase] *
  medium_u.MultiPhase.x[nPhase] *
  medium_u.d;

```

5 Simulation steam accumulator

In Figure 3 the simulation models for a Ruths accumulator and a single control volume are shown. They are connected to an ambience model by a control valve, an enthalpy flow rate sensor and a MultiPhase flange which ensures that only vapour is leaving the volumes. The enthalpy flow rate is integrated over time. The steam accumulator is discretised in the vertical direction into control volumes. The water vapour exchange between the volume elements is calculated with the downflow model as presented in section 4.

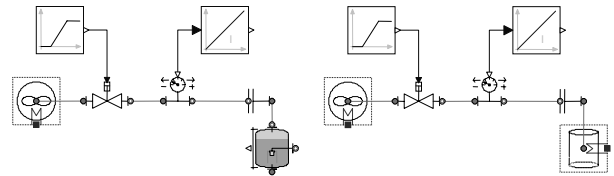


Figure 3: simulation models for Ruths accumulator and single control volume

The ambience model is initialized with a pressure of 3 bar. The volumes of the pressure vessel are initialized with a pressure of 4 bar and an enthalpy of 610 kJ / kg. The height of the pressure vessel is 3 m and the diameter is 1 m. The pressure vessel is filled up to 50 percent with water. The single control volume and steam accumulator model have the same dimensions. In the beginning, the control valve is closed. After 1000 seconds it is opened.

In Figure 4 the simulation results are presented. In the beginning the water is flowing down in the stacked volumes of the Ruths accumulator.

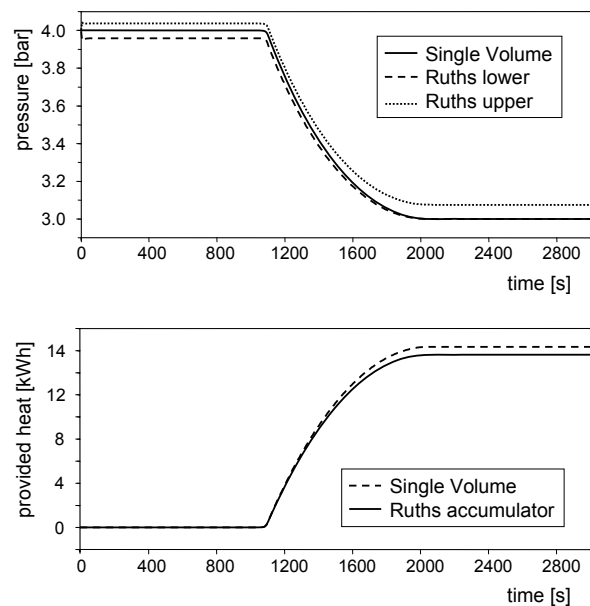


Figure 4: simulation results of pressure and provided heat for Ruths accumulator and single control volume

Because of the static pressure of the water column, the pressure in the lower volume element is increasing and the pressure in upper volume element is decreasing. After 1000 seconds, the control valve is opened and the pressure vessels are discharged to the ambience model. The pressure in the single control volume and in the upper volume of the Ruths accumulator is lowered to 3 bar. The pressure in the lower volume element of the Ruths accumulator re-

mains above 3 bar due to the static pressure. More heat is provided by the single control volume compared to the stacked volumes. The reason for this is the higher pressure in the lower volume elements of the Ruths accumulator which matches the real behavior of the varying-pressure accumulator.

6 Phase change materials (PCM)

Isothermal energy storage increases the storage efficiency. This can be achieved by using a PCM as storage material. Due to the latent heat of melting thermal energy can be stored in a very small temperature range.

6.1 Materials

For the temperature range of interest, various materials can be used as PCMs. Examples of inorganic materials are technical salts and eutectic mixtures of these salts such as: lithium nitrate (LiNO₃), lithium chloride (LiCl), potassium nitrate (KNO₃), potassium nitrite (KNO₂), sodium nitrate (NaNO₃), sodium nitrite (NaNO₂) and calcium nitrate (Ca(NO₃)₂). In the group of organic PCMs an interesting material is Pentaerythritol. It undergoes a solid-solid transition at 189°C with an enthalpy of 269 kJ/kg [Benson, 1985]. The melting point of this plastic crystal is about 50 Kelvin above the solid-solid transition temperature.

6.2 Calculation of enthalpy

Three ways to calculate the enthalpy in the **Modelica.Media** models are compared. The first method is the linear interpolation using If-, Elseif- and Else-clauses, the second method is the usage of the arc tangent function and the third method is the usage of the error function. Using the arc tangent method no If-clauses are needed. The arc tangent function can be calculated by series expansion as provided by Euler in the following way:

$$\arctan x = \frac{x}{1+x^2} + \frac{2}{3} \cdot \frac{x^3}{(1+x^2)^2} + \frac{2 \cdot 4}{3 \cdot 5} \cdot \frac{x^5}{(1+x^2)^3} + \dots$$

The error function is calculated according to W J Cody [3] as an approximation. In this implementation different approximations are combined by If-clauses.

Linear interpolation method

The heat capacity in the liquid and in the solid state is assumed to be constant and increases from the onset temperature of phase change to a maximum at the transition temperature followed by a decrease until the constant value of the liquid state is reached at the final phase change temperature. In Figure 5 heat capacity and enthalpy are plotted against the temperature.

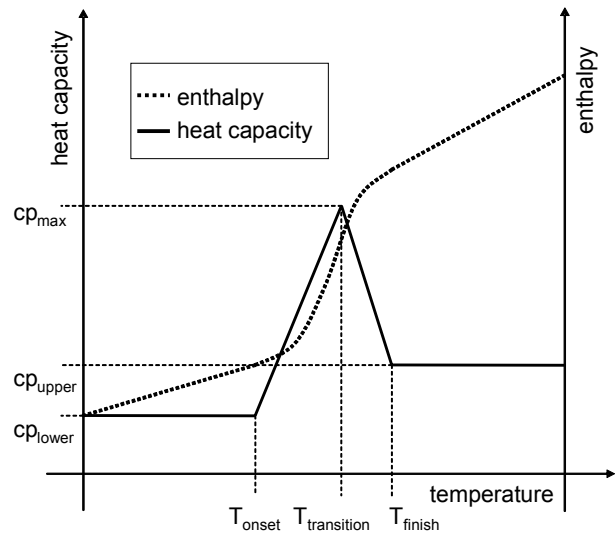


Figure 5: heat capacity and enthalpy plotted against temperature for the linear interpolation method

The latent heat of melting can be calculated in the following way:

$$\begin{aligned} h_{\text{trans}} = & c_{p,\text{low}} \cdot (T_{\text{trans}} - T_{\text{ons}}) + c_{p,\text{up}} \cdot (T_{\text{fin}} - T_{\text{trans}}) \\ & + 0.5 \cdot (T_{\text{trans}} - T_{\text{ons}}) \cdot (c_{p,\text{max}} - c_{p,\text{low}}) \\ & + 0.5 \cdot (T_{\text{fin}} - T_{\text{trans}}) \cdot (c_{p,\text{max}} - c_{p,\text{up}}) \\ & - c_{p,\text{low}} \cdot (T_{\text{fin}} - T_{\text{ons}}) - 0.5 \cdot (c_{p,\text{up}} - c_{p,\text{low}}) \cdot (T_{\text{fin}} - T_{\text{ons}}) \end{aligned}$$

If the transition temperature, the onset temperature of phase change, the final temperature of phase change, the heat capacity in the liquid state, the heat capacity in the solid state and the latent heat of melting are known, the maximum heat capacity can be calculated from the latent heat of melting.

$$\begin{aligned} c_{p,\text{max}} = & \frac{2 \cdot h_{\text{trans}} - c_{p,\text{low}} \cdot (T_{\text{trans}} - T_{\text{ons}}) - c_{p,\text{up}} \cdot (T_{\text{fin}} - T_{\text{trans}})}{T_{\text{fin}} - T_{\text{ons}}} \\ & + c_{p,\text{low}} + c_{p,\text{up}} \end{aligned}$$

The calculation of the enthalpy in the **Modelica.Media** package is implemented with if, elseif and else clauses in the following way:


```

if noEvent(T < Tonset) then
    h = cp,lower * (T - T0);
elseif noEvent(T > Tfinish) then
    h = cp,lower * (Tonset - T0) + htransition + cp,upper * (T - Tfinish);
else
    if noEvent(T < Ttransition) then
        h = cp,lower * (Tonset - T0) + cp,lower *
            (T - Tonset) + 0.5 * (cp,max - cp,lower) *
            (T - Tonset)2 / (Ttransition - Tonset);
    else
        h = cp,lower * (Tonset - T0) + cp,lower *
            (Ttransition - Tonset) + 0.5 * (cp,max - cp,lower) *
            (Ttransition - Tonset) + cp,max * (T - Ttransition) - 0.5 *
            (cp,max - cp,upper) * (T - Ttransition)2 /
            (Tfinish - Ttransition);
    end if;
end if;
    
```

Arc tangent method

The arc tangent function is used to reproduce the behavior of the PCM's enthalpy at the melting point.

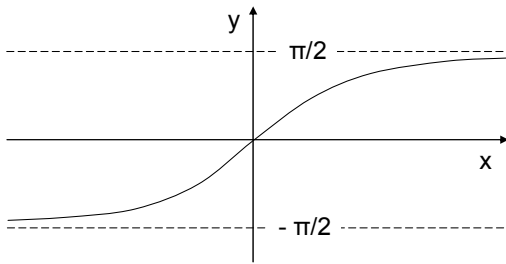


Figure 6: arc tangent curve

For the calculation of the enthalpy curve the arc tangent function is standardized to one and moved in y direction by 0.5 and in the x direction by the melting temperature, $T_{\text{transition}}$. The resulting curve is multiplied with the latent heat of melting, $h_{\text{transition}}$. The sensible heat, c_p^* , is added in the second term. The melting range coefficient, C_{MR} , is used to adjust the width of the melting range.

$$h = h_{\text{transition}} \cdot \left[\frac{\arctan\left(\frac{(T - T_{\text{transition}}) \cdot C_{\text{MR}}}{\pi}\right) + 0,5}{1} \right] + c_p^* \cdot (T - T_0)$$

The effective heat capacity is calculated by differentiating the enthalpy function.

$$c_p = \frac{h_{\text{transition}}}{\pi \cdot \left(\left(\frac{(T - T_{\text{transition}}) \cdot C_{\text{MR}}}{\pi} \right)^2 + 1 \right)} + c_p^*$$

In Figure 7 enthalpy and heat capacity calculated with the arc tangent method are plotted against temperature. The thermodynamic properties of an eutectic mixture of potassium nitrate, sodium nitrate and sodium nitrite are used. The latent heat of melting, $h_{\text{transition}}$, is 83 kJ/kg, the melting temperature, $T_{\text{transition}}$, is 415 Kelvin, the heat capacity, c_p , is 1.5 kJ/(kg K) and the reference temperature, T_0 , is 273 Kelvin. The melting range coefficient, C_{MR} , is 2.

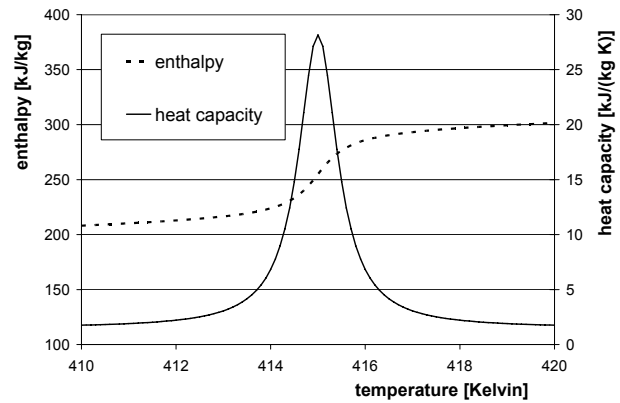


Figure 7: enthalpy and heat capacity calculated with the arc tangent method plotted against temperature

Error function method

The Gauss error distribution curve is used to reproduce the behavior of the heat capacity of the PCM at the melting point.

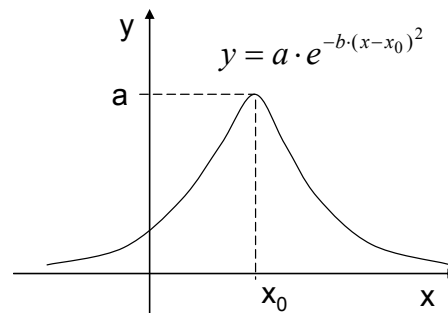


Figure 8: Gauss error distribution curve

For the calculation of the heat capacity curve the coefficients of the Gauss error distribution curve are adapted. The coefficient a is the maximum heat capacity, $c_{p,\text{max}}$, at the melting temperature, x_0 is the melting temperature $T_{\text{transition}}$ and b is the melting range coefficient, C_{MR} , which is used to adjust the width of the melting range. The sensible heat, c_p^* , is added in the second term.

$$c_p = c_{p,\text{max}} \cdot \exp\left[-C_{\text{MR}} \cdot (T - T_{\text{transition}})\right] + c_p^*$$

The enthalpy is calculated by integrating the heat capacity function.

$$h = h_{\text{transition}} \cdot \frac{\text{erf}\left[\sqrt{C_{\text{MR}}} \cdot (T - T_{\text{transition}})\right] + 1}{2} + c_p \cdot (T - T_0)$$

The error function is implemented according to W J Cody [3]. The heat capacity $c_{p,\text{max}}$ in the melting point can be found with the following equation.

$$c_{p,\text{max}} = h_{\text{transition}} \cdot \sqrt{\frac{C_{\text{MR}}}{\pi}}$$

In Figure 8 the enthalpy and the heat capacity calculated with the error function method are plotted against temperature. The thermodynamic properties of the eutectic salt mixture, as presented in the arc tangent method, are used. The melting range coefficient C_{MR} is 1.

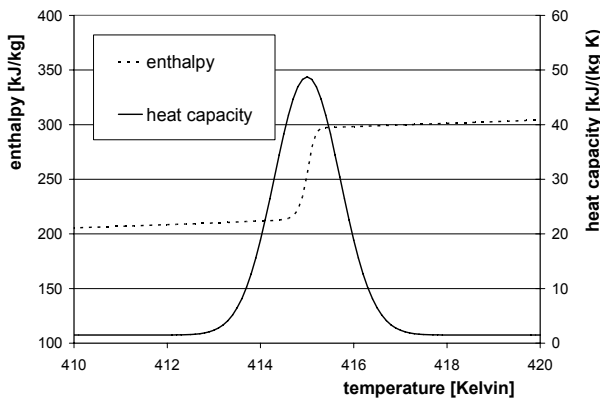


Figure 8: enthalpy and heat capacity calculated with the error function method plotted against temperature

Comparison enthalpy methods

For the comparison of the three suggested methods the thermal behaviour of a cylinder of PCM is simulated. The cylinder is discretised into 100 radial layers. The inner diameter is 1cm and the outer diameter is 2cm. The prescribed temperature at the inner side of the cylinder is switched between 440 and 400 Kelvin every 1000 seconds. Again the thermodynamic properties of the eutectic salt mixture, as presented in the arc tangent method, are used. The resulting temperature profile is plotted in Figure 9. Volume 1 is the PCM layer at the inner side of the tube and Volume 100 is the PCM layer at the outer side of the tube.

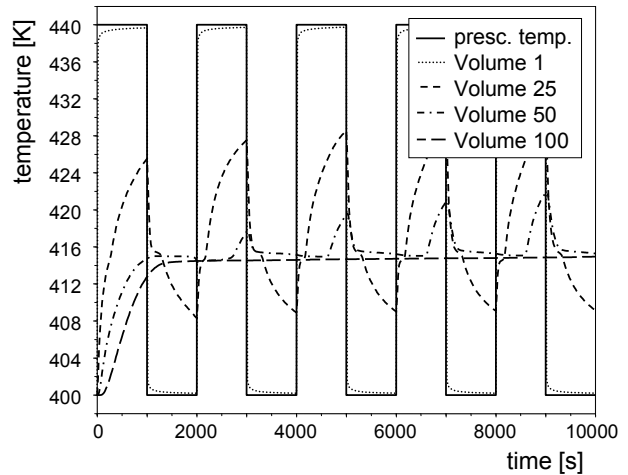


Figure 9: temperature profile of radial discretised cylinder

The results of simulation time of the three methods to calculate the enthalpy of the PCM are presented in the following table:

	Arc tangent	Linear interpol.	Error function
CPU-time for integration [s]	40,7	179	1250
CPU-time for one GRID interval [ms]	81,4	358	2500
Number of (successful) steps	197444	28846	25471

The arc tangent method is 4.5 times faster than the linear interpolation method and more than 30 times faster than the error function method.

7 Simulation PCM enhanced steam accumulator

A concept to increase the energy efficiency of an existing varying-pressure steam accumulator is to connect a tube register with externally arranged PCM to the pressure vessel (Figure 10). A pump assures the circulation between the tube register and pressure vessel. The charging and discharging operations are the same as for a varying pressure accumulator, as presented in section 2. With the PCM the capacity of the storage system can be increased. The same amount of thermal energy can now be stored in a smaller temperature range. The heat transport inside of the PCM is a limiting factor to the power of the storage system.

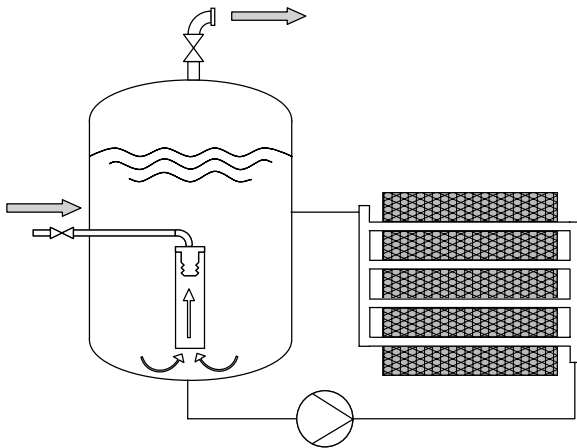


Figure 10: PCM enhanced steam accumulator

In Figure 11 the model of the PCM enhanced steam accumulator is shown as it is implemented in Modelica. The ambience model is initialized with a pressure of 3 bar. The volumes of the pressure vessel are initialized with a pressure of 4 bar and an enthalpy of 610 kJ / kg. The height of the pressure vessel is 3 m and the diameter is 1 m. The pressure vessel is filled up to 50 percent with water. The inner diameter of the 10 tubes connected to the pressure vessel is 20 mm and the outer diameter is 25 mm. The outer diameter of the PCM layer is 100 mm. The tubes have a length of 5 meter. The thermodynamic properties of the eutectic salt mixture, as presented in section 6.2, are used. The temperature of the PCM is initialized to 416 Kelvin. This means that the PCM is completely molten.

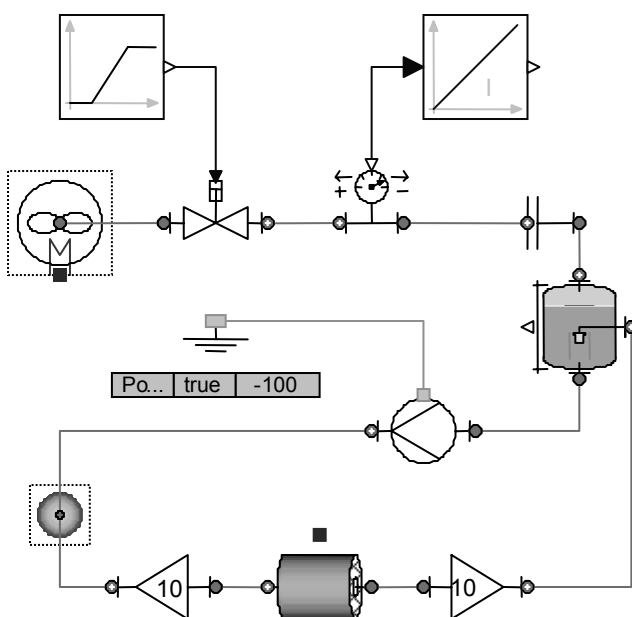


Figure 11: simulation model PCM enhanced steam accumulator

In the beginning, the control valve is closed. After 1000 seconds it is opened and the storage vessel is discharged. The pump is switched on all the time.

In Figure 12 simulation results of pressure and provided heat are plotted against time. The pressure in the upper part of the storage vessel is decreasing after the control valve is opened to 3 bar. The provided heat is higher compared to the simulation results of the varying-pressure accumulator in section 5.

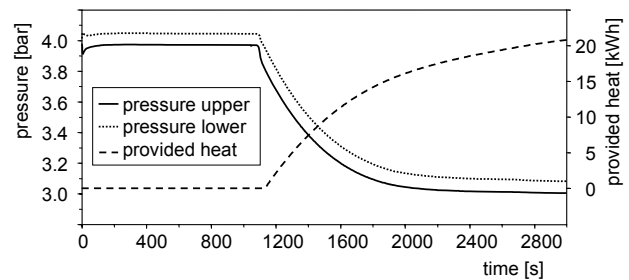


Figure 12: simulation results of pressure and provided heat for the PCM enhanced steam accumulator

In Figure 13 the temperature in the lower part of the storage vessel and in the PCM layer around the tubes is plotted against time. Only parts of the PCM are solidified after 2000 seconds of discharging. The melting temperature is 415 Kelvin.

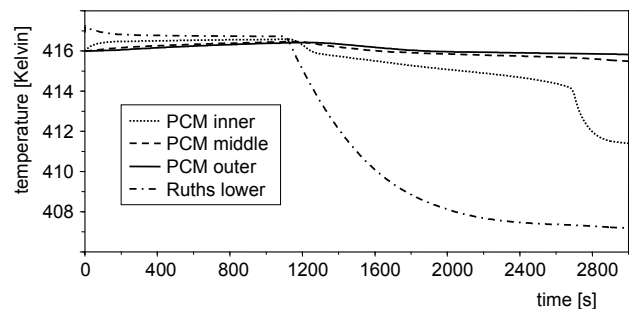


Figure 13: simulation results of the temperature in the lower control volume of the pressure vessel and the temperatures in the PCM cylinders around the tube register

8 Conclusions

With the simulation models presented in this work, PCM steam storage concepts can be analyzed. Simulation results of a PCM steam accumulator with a tube register and externally arranged PCM are presented. In the next step, different simulation models will be developed for comparison. These simulation models will focus on using macro-encapsulation of

the storage material, composite materials and the integration of layers made of materials showing a high thermal conductivity to increase the power of the storage system. Further validation of the models will be done using laboratory scale experiments. Using a tube register with externally arranged PCM, the number of tubes and their distance has to be defined, depending on the steam process the accumulator is integrated into. The aim of the work is to provide a dimensioning tool that enables to design the thermal behavior of the steam accumulator.

References

- [1] Goldstern, Walter Steam storage installations. Springer-Verlag OHG, Berling / Göttingen / Heidelberg, 1970.
- [2] Steinmann, W.D., Buschle, J. Analysis of thermal storage systems using Modelica. Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, pp 331-337, 2005.
- [3] Cody, W. J. Rational Chebyshev approximations for the error function, Mathematics of Computation, pages 631-638, 1969.
- [4] Benson, D.K., Burrows, R.W., Webb, J.D. Solid state phase transitions in Pentaerythritol and related polyhydric alcohols. Solar Energy Materials 13, pp 133-152, 1985.

An Enhanced Discretisation Method for Storage Tank Models within Energy Systems

Dr.-Ing. Stefan Wischhusen
XRG Simulation GmbH
Kasernenstraße 12, 21073 Hamburg, Germany
wischhusen@xrg-simulation.de

Abstract

This article presents a new discretisation function that can be applied to flow models using Finite-Volume-Method. The function is required since the commonly applied UPWIND discretisation yields a low accuracy when convection is small with regard to volume size, e. g. for storage tank models. The new approach is compared to measurement data and it shows a much higher accuracy incorporating the same number of control volumes so that the user may decrease the problem size considerably.

Keywords: discretisation method; UPWIND; tank model; cogeneration; energy system

1 Introduction

By means of simulation tools like HKSIm [1] it is possible to model complex energy supply system layouts in order to find improvement potential in the development or optimisation phase of a project. HKSIm integrates Dymola/Modelica for modelling and simulation. It is used by Imtech Deutschland GmbH & Co. KG and is currently extended by XRG Simulation GmbH for performing simulation of steam and compressed air systems.

Storage tanks are a necessary part of each cogeneration and regenerative power supply system. Those energy systems like shown in Fig. 1 have become very popular for large but also small applications. From the energetic point of view it is of big interest to model the transient behaviour of such tanks in order to determine temperatures in feed and return lines precisely. Since control algorithms for such plants rely on actual temperatures measured at different tank levels a correct temperature prediction influences the result of a simulation for power supply, switch-on times, etc., strongly.

A hot water storage tank fulfils two functions at the same time. First, hot water can be stored when demand and supply is incoherent (e. g., in cogeneration and regenerative applications). Second, a degree of freedom is introduced to the hydraulic layout so that pumps may operate independently for consumers and heat suppliers. For example, the storage tank in Fig. 1 is charged with hot water when the mass flow rate of water through the CHP is greater than that in the return line (boilers are considered to be off) – this happens when heat demand is low and power is required. On the other hand, a discharge is automatically initiated when the CHP is switched off and consumers are still fed. Due to the positive gradient of the specific volume w.r.t. temperature (above 4°C) a very good separation of water with different temperatures can be achieved when the tank is fed with hot medium at top and colder medium at bottom level. For that reason this kind of storage tank always shows an aspect ratio which is greater in vertical direction.

In general, a more or less sophisticated system control (cascade connexion) is implemented which switches CHP and boiler with regard to the actual heat demand and tank temperatures. At low heating demand the tank outlet temperature will rise after a period of time and therefore the CHP has to be shut-down to prevent an overheat of the engine. Of course a switch-on is possible again when heat demand rises but the shutdown interval and the number of power-up sequences influences the durability of the engine (note, that the CHP's are usually adapted from vehicle engines which have an average lifetime of a few thousands of hours). It is therefore of big interest to decrease the number of (cold-)starts at a maximum power output. An optimum performance can be achieved by also defining the "right" tank size. This can be done by means of the simulation tool HKSIm.

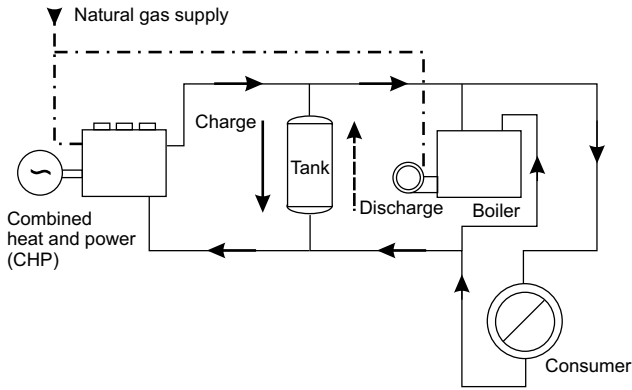


Fig. 1: Simplified schematic of a cogeneration plant with integrated hot water storage tank

2 Discretisation methods for thermo-hydraulic system modelling

Physical systems are always described by partial differential equations which consist of derivatives w.r.t. time and space. Applying simulation tools like Dymola/Modelica it is possible to model and solve the dynamic part of those equations – but all derivatives w.r.t. space have to be simplified using finite methods (e.g., Finite-Volume-Method or Finite-Element-Method [2]).

The balance equations for an incompressible medium with constant density (e.g. liquid water) can be derived from volume integrals [3]. The mass balance is rather simple due to constant density and constant volume size.

$$\frac{dm}{dt} = \dot{m}_{in} - \dot{m}_{out} = 0 \quad (1)$$

For the energy balance follows assuming low kinetic energy (total energy \approx internal energy U):

$$\frac{dU}{dt} = \underbrace{\dot{m}_{in} \cdot h_{in} - \dot{m}_{out} \cdot h_{out}}_{\text{convection}} + \underbrace{\dot{Q}_s}_{\text{heat source/sink}} \quad (2)$$

For incompressible media the specific internal energy $u = U/m$ is a function of temperature as well as the specific enthalpy h (since the density is considered to be constant). Both variables are computed by the specific heat capacity c which is equal to the derivative of specific internal energy u w.r.t. tempera-

ture T . The heat capacity may be a function of temperature or can be set constant which is a good approach for pure water in the usual range of operating conditions.

$$U = m \cdot c(T) \cdot T \quad (3)$$

Therefore one may simplify Eq. (2) to the following term:

$$m \cdot c \cdot \frac{dT}{dt} = \underbrace{\dot{m}_{in} \cdot c \cdot T_{in}}_{\dot{H}_{in}} - \underbrace{\dot{m}_{out} \cdot c \cdot T_{out}}_{\dot{H}_{out}} + \dot{Q}_s \quad (4)$$

Usually, mass and energy balance equations are coupled by a momentum balance but in the incompressible case the pressure is not a parameter for medium properties.

If Eq. (4) has to be solved in order to receive an explicit differential equation one must determine the outflow temperature (or outflow enthalpy) of the volume applying known states calculated for the center of each volume. That is the reason why a discretisation method has to be applied.

Usually, for thermo-hydraulic models like pipes, pumps and heat exchangers an UPWIND discretisation scheme is chosen since it is numerically robust and easy to implement at the same time. Validation reveals that such a discretisation is appropriate for plant components which show a large mixing behaviour [4] (e.g., stirrer tanks). Pipes are modelled by a number of serial control volumes which are connected by the UPWIND discretisation (Fig. 2). The equation for calculating state variables (like specific enthalpy h , temperature T or density d) on **downstream** volume boundaries is simple:

$$\Theta_{down} = \Theta, \quad \Theta = h, T, d \quad (4)$$

But the numerical mixing behaviour of this method shows a large diffusion (refer to temperature slope in 30th and 31st of Aug. in Fig. 3) of heat for tanks which are designed to store hot water in vertical layers. In order to prevent this one has to model each tank with a large number of control volumes (usually more than 40 volumes) if UPWIND is applied. Therefore, the tank model generates a high computa-

tional effort when it comes to plant simulations for one year simulation time.

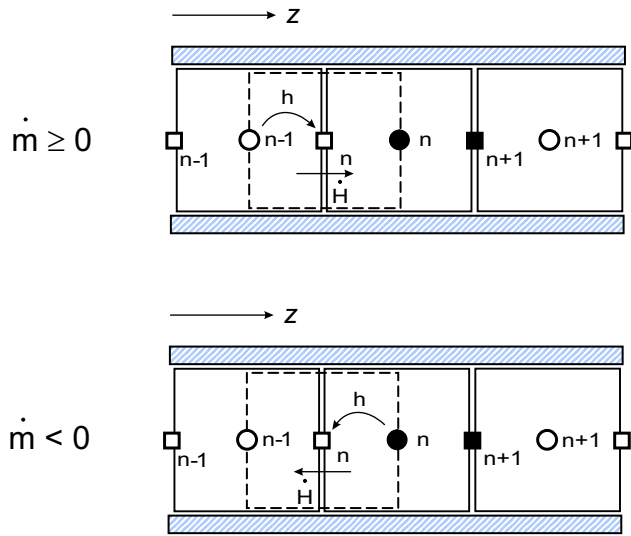


Fig. 2: UPWIND-discretisation for an one-dimensional flow and different directions of mass flow

Another very common discretisation method is the linear interpolation of states between two volumes. Unfortunately, this method results in an unstable or even false solution when it comes to transient simulations. Therefore, UPWIND is still widely used even when storage tanks have to be modelled.

A typical outcome of such a simulation is shown in Fig. 3. One can easily see, that the temperatures during discharge (31st of Aug.) are not predicted very well by the model which consists of 40 discrete volumes. In fact, there is a strong deviation with regard to time. In addition, the bottom temperature is rising quicker than measured during charging periods (28th of Aug.). Since the volume of the tank is known precisely, and the mass flow rate is available from measurements the discretisation method remains as one significant failure potential.

Another indication is following from control theory: Calculating the transfer function of Eq. (4) and furthermore carrying out a Fast Fourier Transformation for a step response the following equation is yielded.

$$T_{out} - T_{out,t=0} = \Delta T_{step,in} \cdot \left(1 - e^{-\frac{tm}{m}} \right) \quad (5)$$

The result of such a step response is visualised in Fig. 4. It clearly shows that the state (e. g. temperature) of a finite volume immediately rises due to a sudden change of enthalpy flow at the upstream boundary. Therefore, the outgoing flow changes its temperature **at the same time** the step or any change occurs at the inlet boundary.

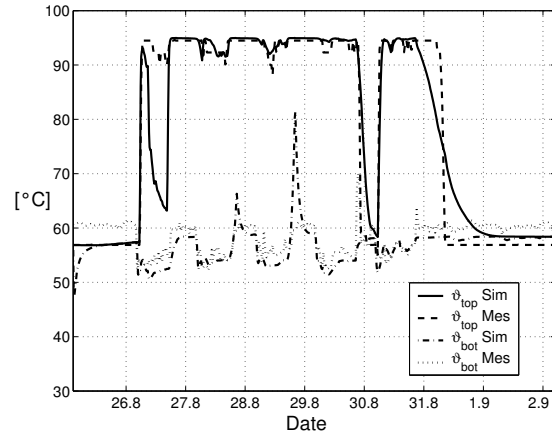


Fig. 3: Comparison of simulation (Sim) and measurement (Mes) of hot water temperatures at bottom and top level of a storage tank – Simulation is carried out by using UPWIND method

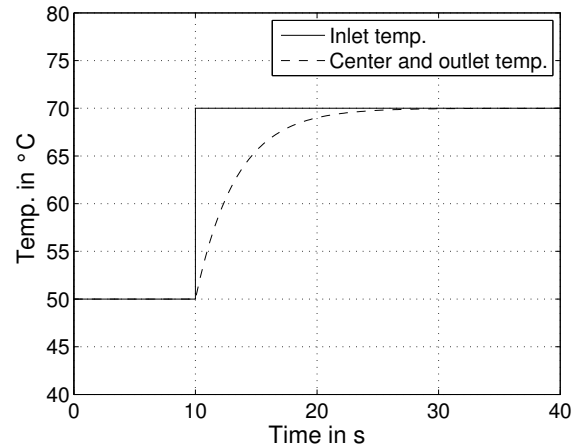


Fig. 4: Step response (Eq. 5) with a weight factor $\dot{m}/m = 0.3$

3 An enhanced discretisation approach

The requirements for a new discretisation method for storage tank models are listed below:

1. Most important, the approach shall deliver a better prediction for enthalpy flows over control volume boundaries w.r.t. time.

- Standard interfaces which are used for UPWIND models shall be compatible.

The main idea of the new discretisation method is to achieve a gradient-dependent outflow state. Therefore, the gradient between the adjacent upstream control volume and the center state is taken as a criteria for the interpolation between downstream and center state. The interpolation function is chosen to be of exponential type:

$$\Theta_{out} = (\Theta_{center} - \Theta_{down}) \cdot e^{-a|\Theta_{up} - \Theta_{center}|} + \Theta_{down} \cdot (6)$$

This function is valid for both possible flow directions in one-dimensional flow modelling. While the upstream gradient is not close to zero the outlet boundary state is almost equal to the downstream state. But when the upstream gradient is small the exponential function turns zero and therefore the corresponding volume is considered to be “charged”. From that point the outflow state will be equal to the center state. By increasing the tuning parameter a it is possible to decrease the numerical “diffusion” of that method (see Fig. 5).

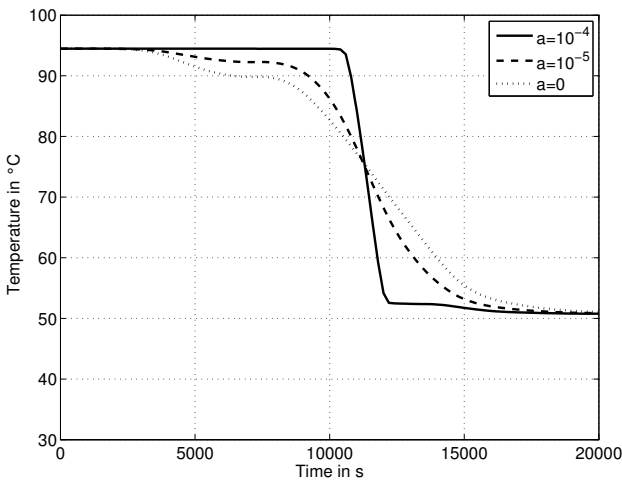


Fig. 5: Discharge temperature of the storage tank model shown in Fig. 6 for different tuning factors a

It has to be mentioned that the tank model also displays buoyancy effects (which are an important feature) and therefore the unstable case of different gradient signs w.r.t. upstream and downstream directions is not resulting into problems. If this function should be applied for modelling of other flow problems provisions in Eq. (6) should be made in order to encounter those possible problems.

4 Validation of tank model

For the validation of the tank model measurement data is available. It shows the flow rate through a storage tank as well as the temperatures for top and bottom duct. The tank model looks like displayed in Fig. 6.

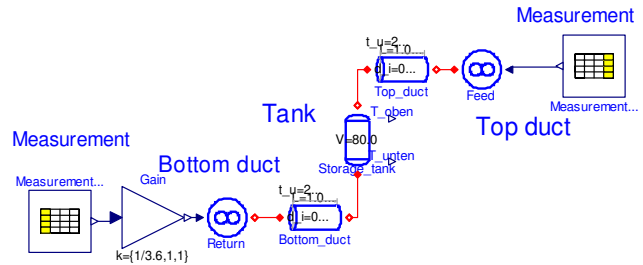


Fig. 6: Validation model for the storage tank model

The storage tank has a liquid volume of 80 m³ and the thermal insulation can be considered to be ideal (adiabatic conditions). Buoyancy effects in flow direction (gravity vector is parallel to flow vector) are taken into account. Since the mass flow rate is provided in terms of a volume flow rate a gain block divides this input by 3.6 (conversion from m³/h to kg/s for water). The measurement data was obtained from the beginning of a heating period in September and was recorded from a large cogeneration plant with a time step interval of 15 min. The tank model applies the new discretisation approach ($a=10^{-4}$) for $n - 4$ volumes. This means that the first two and last two volumes refer to UPWIND-scheme.

An operation of one week is investigated.

Number of volumes n	10 new	10 UPW	20 UPW	40 UPW
Non-dimensional simulation time	2.60	1.00	3.92	11.82
σ_{sim} [-]				

Tab. 1: Effect of both discretisation methods and number of volumes on non-dimensional simulation time

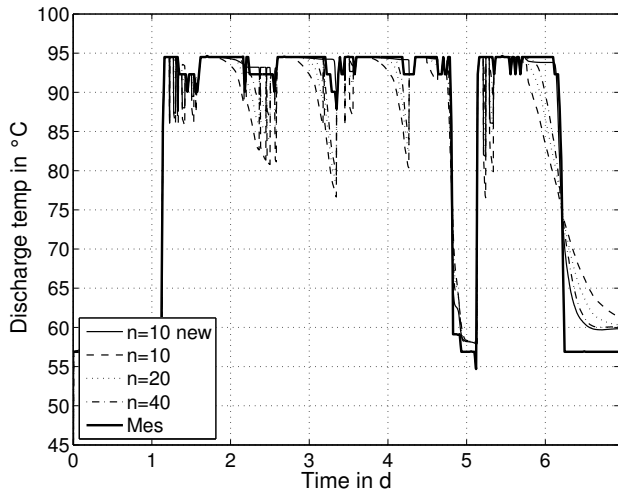


Fig. 7: Comparison of discharge temperature with measurement data

First, the effect of a different number n of discrete volumes is evaluated. The result is shown in Fig. 7. One can see that the accuracy of the solution with regard to the measurement of the top duct temperature is rising when the number of volumes is increased. But with the number of control volumes also the simulation time increases as Tab. 1 reveals. Deviations for the top temperatures are high when convection is small w.r.t. volume size (see day 6 to 7). But also the bottom temperature shows higher deviations when the number of control volumes is decreased (refer to peaks in Fig. 8 on 2nd, 3rd and 4th day).

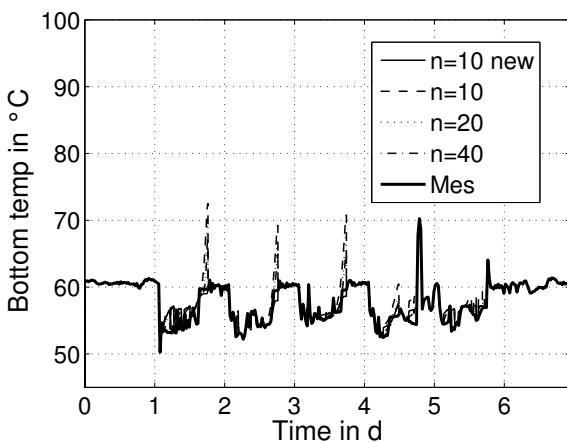


Fig. 8: Comparison of bottom temperature with measurement data

Using the new discretisation it is possible to achieve a much higher accuracy. Comparing Fig. 7 and Fig. 9 reveals that the top temperature is following the measurement much better although just 6 volumes

(due to interface compatibility, note remark on previous page) are applying the new discretisation scheme. This fact explains a visible deviation on 7th day in Fig. 9 which is due to the two uppermost volumes using UPWIND discretisation. Adapting hydraulic interfaces in order to enable a calculation of the upstream gradient will lead to a better anticipation of the measurement.

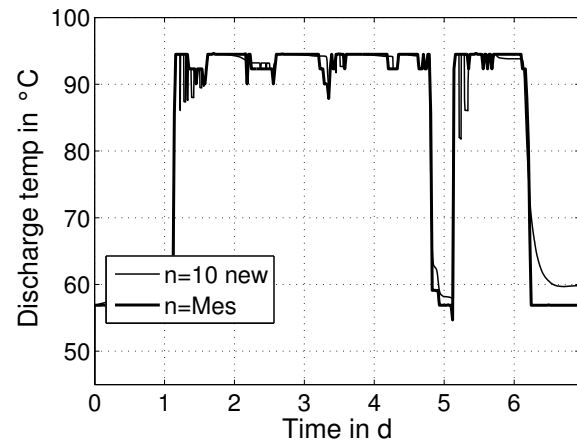


Fig. 9: Comparison of new discretisation approach with measurement data

In addition, also the bottom temperature is predicted more precisely so that the temperature peaks obtained with a discretisation of 10 or 20 volumes vanish (compare Fig. 8 and Fig. 10).

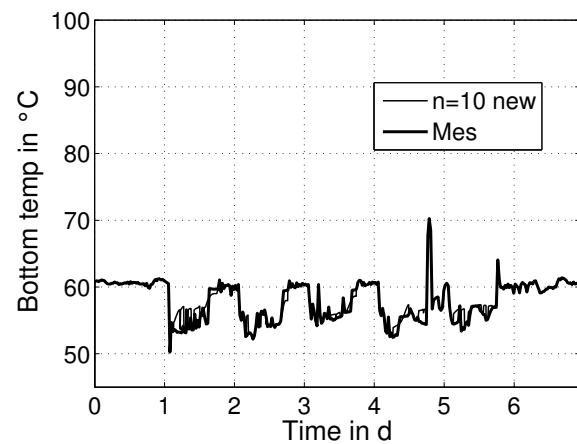


Fig. 10: Comparison of new discretisation approach with measurement data

5 Evaluation for cogeneration plant

What are possible consequences of the discretisation's implementation within a transient simulation model of a typical cogeneration plant? As mentioned before the purpose of a hot water storage tank is to decouple power and heat supply. In case of a high power requirement the tank will be charged until the maximum capacity is reached. At this point CHP's heat production must be cut down or even completely shut down. Thus, possible power peaks can not be decreased resulting in high power charges (actually, it is not the energy rate which dominates in such a case). So, from the energetic point of view one could tend to integrate larger tanks in such plants. But from the financial point of view this strategy is obviously restricted to defined limits. Of course, it is possible to integrate heat exchangers for removing "waste" heat to ambience but this option is not desirable from the energetic point of view.

In Fig. 11 a simulation model of a typical industrial cogeneration plant is shown. Two CHP's are installed for providing power (power priority control) while the boilers are used in order to control the feed temperature and backup heat production in case CHPs are off. The heat consumers are represented by a simple consumer model on the right side of that figure. The profile of the heat consumption may change dynamically and almost any signal source from the Modelica.Blocks library may be used.

In this evaluation a simple constant power and heat demand was applied while the power demand (1200 kW_{el}) is larger than the heat demand (600 kW_{th}) forcing the tank (volume $V = 30 \text{ m}^3$) to be charged when CHPs are on. A very ordinary flip-flop control is used to prevent overheating at CHP's hot water inlet. Basically, both engines are switched off when a temperature of $87 \text{ }^\circ\text{C}$ at the bottom level **and** $100 \text{ }^\circ\text{C}$ at the top is exceeded. A restart is possible again when top temperature drops below $90 \text{ }^\circ\text{C}$. The tuning parameter a_{dis} was set to either 0 for UPWIND or $1e-4$.

Results reveal that with the same tank volume but different discretisation schemes the operational time at the same power output is different. Actually, the CHP models can be operated 15 to 20 % longer than with UPWIND discretisation (see Fig. 12 and Fig. 13). The difference is depending on the supplied CHP control logic. So, the chosen discretisation may result in considerably smaller tank sizes when a certain operation time interval has to be guaranteed – another evaluation reveals that the same mean continuous operation interval (time between switch on and switch off) is reached with new discretisation when tanks are approx. 25 % smaller in volume. Also, it is important knowing the number of power-up sequences during a certain period of time which has to be lower than the requirement from the CHP manufacturer when warranty conditions shall be respected.

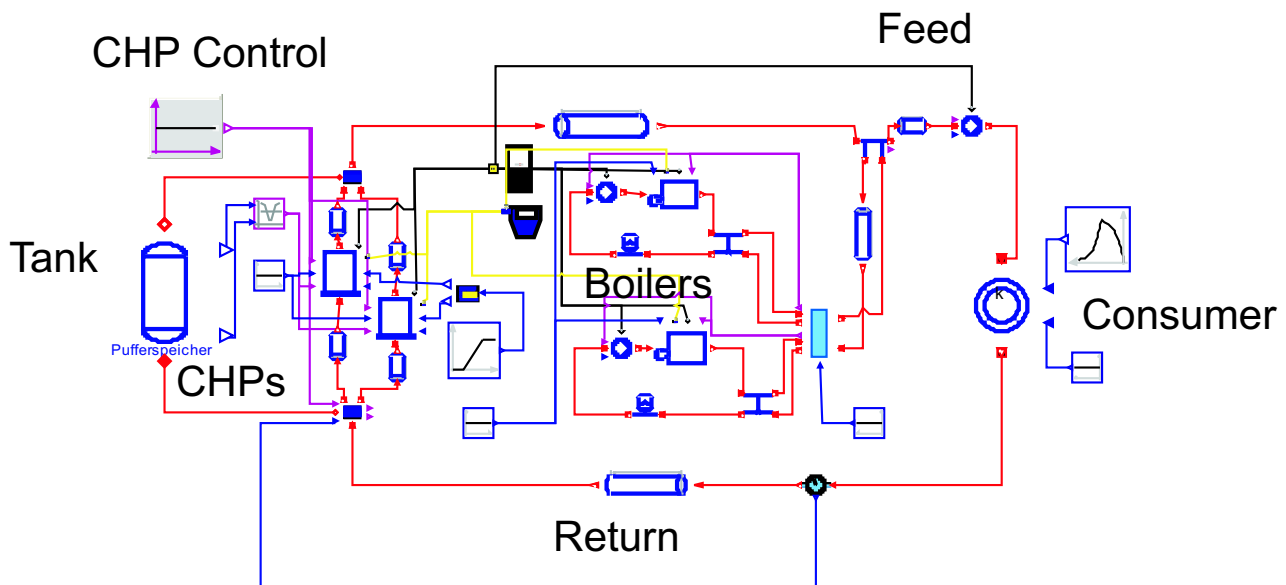


Fig. 11: Simulation model of a cogeneration plant with temperature dependent CHP control

For the simulation of one week the total number of engine starts (for a single engine) was determined with 37 for UPWIND and 31 for new discretisation approach. For both simulations it must be pointed out that the total power and heat production of all moduls was basically equal indicating that the energy balance of the system was conservative (this statement applies also for a simulation of one year under the same boundary conditions).

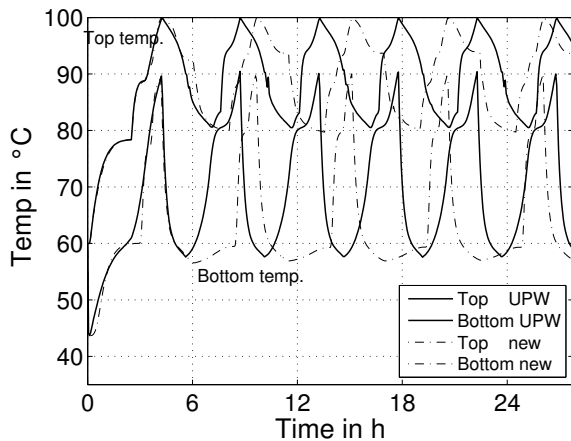


Fig. 12: Tank temperatures for both discretisation approaches

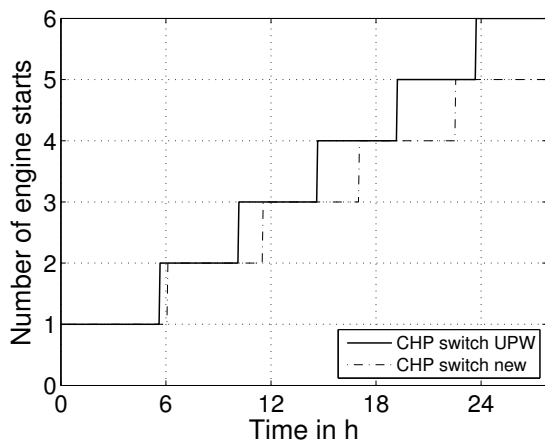


Fig. 13: Number of CHP engine starts

6 Conclusions

The new discretisation scheme enables a more accurate modelling of storage tank models although far less control volumes are required than for a conventional UPWIND discretisation that yields almost same results. Thus, computation times are reduced considerably. Validation shows that temperatures at standard positions for temperature measurements are predicted with a very good agreement. Especially,

temperatures during slow discharge are displaying a much sharper gradient than with UPWIND method. It is possible that the new method may also be used for other flow problems which show the same behaviour: convective flow and very low dissipation. (sharp border front flow) like for example in some kind of evaporators in cooling plants. The influence of dissipation which is displayed in the gradient of a step change is adjusted by a single parameter. This tuning parameter can be changed so that mathematically the UPWIND method is applied. It must also be stressed that the method is compatible with control volume models which apply UPWIND. If this method should be applied for every component model the interfaces must be changed in order to access downstream and upstream states for first and last control volume in any component model.

With the new discretisation method it is possible to obtain smaller tank sizes while assuming identical boundary conditions. The result's difference could reduce investment costs when large plants are planned by means of simulation tools. For example the savings in acquiring a 30 m³ instead of a 40 m³ tank could be approx. 8.000 € [5].

References

- [1] Lüdemann, B.; Wischhusen, S.; Engel, O.; Schmitz, G.: Optimierte Energiesysteme, BWK, Bd. 55, No. 9, Springer VDI-Verlag, Düsseldorf, Germany, 2003.
- [2] Casella F. and Schiavo F.: Modelling and Simulation of Heat Exchangers in Modelica with Finite Element Methods. In Proceedings of 3rd Modelica conference, Linköping, Sweden, pp. 343-352.
- [3] Wischhusen, S.: Dynamische Simulation zur wirtschaftlichen Bewertung komplexer Energiesysteme. Cuvillier Verlag, Göttingen, Germany: PhD thesis, Department of Thermodynamics, Hamburg University of Technology, 2005.
- [4] Mühlthaler, G. Anwendung objektorientierter Simulationssprachen zur Modellierung von Kraftwerkskomponenten, VDI Verlag, Düsseldorf, Germany: PhD thesis, Department of Thermodynamics, Hamburg University of Technology, 2001.
- [5] KFServer – Online-Server for cost functions of energy supply system components: <http://kfserver.kaiserstadt.de>.

HydroPlant – a Modelica Library for Dynamic Simulation of Hydro Power Plants

Kristian Tuszynski, Jan Tuszyński, Karl Slättorp
Modelon AB, Datavoice HB, Tactel AB

kristian.tuszynski@modelon.se, jantuz@tele2.se, karl.slattorp@tactel.se,

Abstract

This paper presents a library for simulation of hydro power plants. The library is designed to be an effective tool for commissioning, testing of new control strategies and verifying complete hydro plants or selected plant systems only.

Keywords: Hydro Power; Simulation; Turbine; Penstock; Reservoir

1 Introduction

The hydro plant process is on first sight easy to understand and well documented. Development of new control strategies could be accordingly based on that knowledge verified through trial and error during the commissioning. This traditional approach shows to be time consuming and expensive. Plant models available are often obsolete as they are simplified for the narrow linear range of the working area and not directly executable. Intuitive knowledge of the original developers is practically not available. Real plants are run under stringent economical demands and thus not available for testing.

The library was initially designed to test control strategies and make commissioning more efficient, and it proved to be useful for evaluation of complete plants (Figure 1).

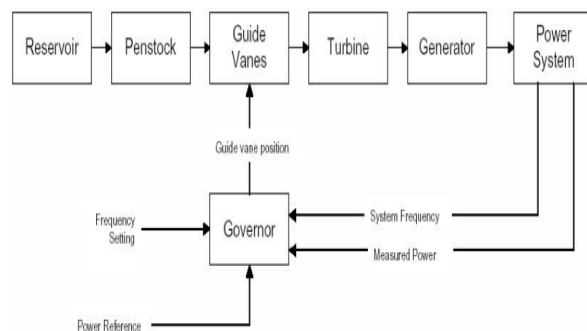


Figure 1 Component Overview

The library structure builds on four main groups of models: hydro components, electrical power system, mechanical machinery and control components. This paper covers mainly hydro components and hydro turbines exemplifying mechanical machinery. Models of the electrical power system with generator and grid address active power only and are more or less conventional but still very useful for study of the complete plant system.

2 Flow Transfer

2.1 Theory

The simulation of thermo hydraulic systems is based on three equations of mass (1), energy (2) and momentum (3) conservation in a control volume:

$$\frac{dM}{dt} = m_{\dot{i}} - m_{\dot{o}} = \sum m_{\dot{}} \quad (1)$$

$$\frac{dU}{dt} = m_{\dot{i}}h_i - m_{\dot{o}}h_o + q + W_s - p \frac{dV}{dt} \quad (2)$$

$$\frac{dG}{dt} = m_{\dot{i}}v_i - m_{\dot{o}}v_o + (A_i p_i - A_o p_o - F_f) + A \rho g (z_i - z_o) \quad (3)$$

where; $m_{\dot{}}$ - mass flow, h - specific enthalpy, q - heat flow from (+) the environment, W_s - external mechanical energy flow, p - the media pressure in the control volume, G - the media momentum, v - media velocity, A - intake/outrake area, F_f - the friction force, z - elevation of the intake/outrake. Indices i, o symbolize entering or exiting flows to/from the control enclosure.

Conservation principles were expressed initially as partial differential equations for an infinitely small control volume (local form) and then converted into the above global formulation of differential equations of the physical enclosure modeled (Leibniz rule).

Pressure p , and temperature T , are selected as state variables of the media in a control volume, allowing calculation of all other media properties from tabulated Modelica water properties.

2.2 Basic Structure of the Flow Transfer

The basic element of the flow structure is built of two containers (control volumes) exchanging media through a single connecting module. The thermodynamic states, p and T , of control volumes are calculated from the mass and energy exchange through connecting module and from/to the environment.

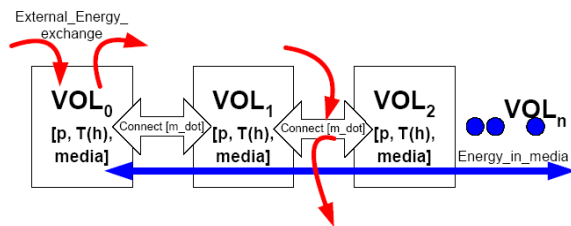


Figure 2. General concept of interconnected control volumes and connecting modules

Considering length and complexity of the plant water ways, modeling of the longitudinal pressure transfer in an elastic media was found essential.

A basic element of two volumes was accordingly expanded into a media transmission line developed initially by Heaviside as telegraph line formulation. Water conduits of the library are sliced into a number of interconnected volumes (Figure 2). Inductance and resistance are represented by water inertia and flow resistance (equation (3)), while capacitance is represented by media elasticity in the control volume.

2.3 Modeling challenges. Nonlinearities

The well established theory of the models would imply ‘green light’ for easy implementation in the hydro plant library. In spite of that several modeling challenges were met.

Vector of media flow velocity: Assuming control volumes part of the water conduit requires completion of the enthalpy with the kinetic energy factor $v^2/2$. Transfer of that energy to the next segment of the conduit depends on the direction of the velocity vector. The problem was addressed by inclusion of the ‘flow recovery’ parameter. An additional challenge of the flow velocity concerns mainly turbines, where flow vortex is an essential part of turbine design.

Nonlinearity of forces acting on the water flow:

The momentum equation includes highly nonlinear force components. These are mainly momentum forces, ‘ m_dot*v ’, and friction forces, F_f , depending on the flow direction and flow velocity (Reynolds number) in laminar and turbulent regions. Library models were based on forms presented in reference [1].

Flow calculation in open channels/reservoirs: Flow in open channels was modeled from the same momentum equation as for enclosed channels, subject to the following:

- Pressure drop, the main force moving the mass, is calculated from media level difference between the adjacent volumes. Level variations depend strongly on the volume geometry.
- The main flow caused by the mass inertia is completed by additional components as e.g. water ‘sliding’ along the slopes of the waves.
- Friction calculation depends strongly on the channel geometry, e.g. bottom and coast lines.

3 Basic Hydro Components

The library is built on two main types of hydro components: *containers*, basic models representing control volumes in open and closed containers, and *connecting modules*, models representing media conduits.

Closed volumes represent a single container or an enclosed water segment of a long media conduit.

Open volumes represent containers with free surface between the media and the gas above allowing media compressibility to be neglected.

Generally standard Modelica connectors are used, but with *FlowPort* and *MediaPort* added. *FlowPort* connects flows of mass and enthalpy, while *MediaPort* carries vector of media property. Input and output ports of both connectors are separated.

4 Hydro Subsystems

Hydro subsystems include a reservoir, penstock and surge tank. All of those models are built of interconnected segments of control volumes and connecting modules. The number of segments is parameterized allowing automatic segment interconnection into complete subsystem. By increasing the number of segments, higher flow/pressure frequencies can be

studied, but at the expense of simulation execution time.

4.1 Reservoir

The library allows simple parameterization of reservoirs fulfilling almost any desired geometry.

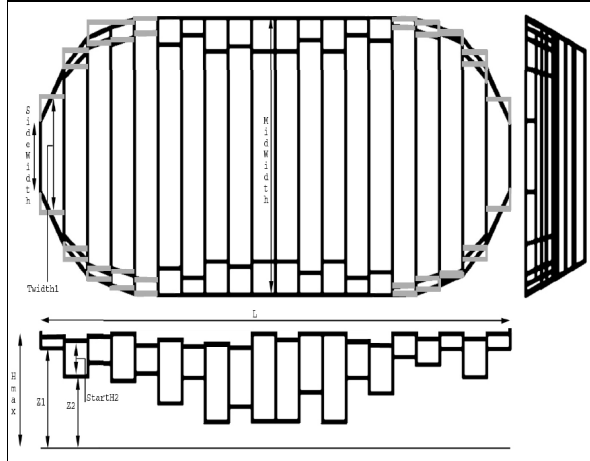


Figure 3. Parameters for dimensioning of the reservoir

Main parameters are: middle and side shore-to-shore width, height of each segment over a reference level, start water level and temperature, maximum container height and a contour factor that decides the shape of the reservoir “coast line”. The contour factor k (Figure 4), represents coast line as a generalized ellipse:

$$\left(\frac{x}{a}\right)^k + \left(\frac{y}{b}\right)^k = 1 \quad (4)$$

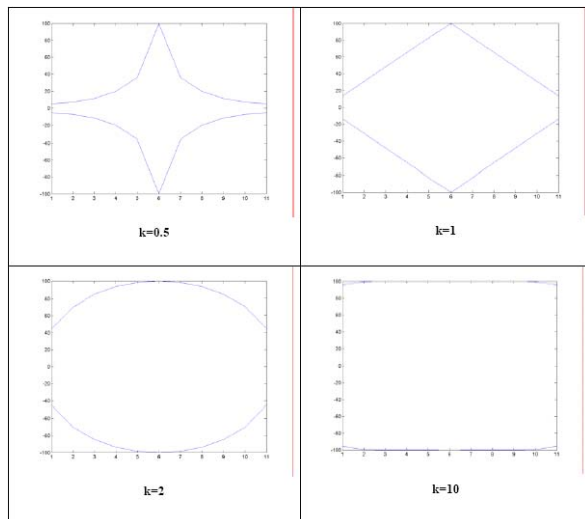


Figure 4. Some examples of reservoir coastlines depending on the contour factor k

Water inlets and outlets can be connected to any segment of the reservoir, allowing study of the hydro plants in cascade. Wave propagation through the reservoir caused by transients in the upstream or downstream plant can be studied. Coordination of plants can be tested to prevent the water level to exceed the maximum level allowed.

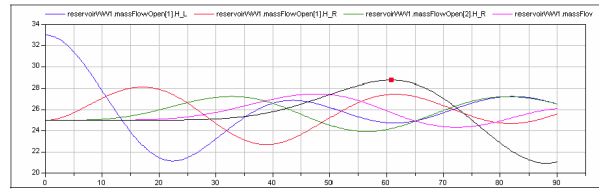


Figure 5. Wave propagation in reservoir of 5 segments and $k = 1$. The start level is 25m in all segments except 33m for the first one

4.2 Water ways. Penstocks and surge tanks

Water of the plant can be transported through enclosed or open conduits. The latter are basically identical to the reservoirs but of the suitable prolonged shapes. The main enclosed conduit is a penstock model. Penstocks are sliced in segments as described above and are treated by the system as flow connecting modules. Surge tank models represent vertical water columns with the upper end opened to the ambient environment.

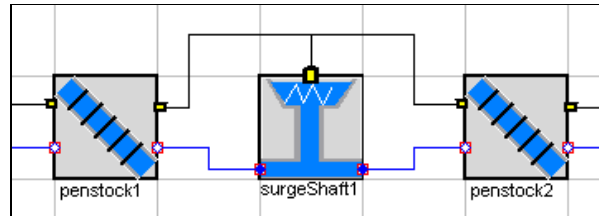


Figure 6. Example of the interconnected penstocks and surge tank

Penstocks and surge tanks can be connected directly (Figure 6), and different groups of penstocks can be coupled through interconnecting volumes. In that way almost any shape and configuration of the waterways on both the inlet and outlet side of the turbine can be modeled.

5 Electrical Power System

There are two modes of the generator modeling: unsynchronized/no-load turbine-generator and generator synchronized to the grid, in load operation. The first mode is basically non-electric, as the turbine-generator represents only a speed-controlled rotating mass. The second mode requires modeling of the

active power of the grid, where the local turbine governor reads whole grid frequency and power output of the local generator. The frequency signal represents dynamics of the whole grid, or the angular speed of all rotating units of the grid.

Figure 7 introduces the main components of the power grid. *Generator and Synchronizer* represents rotating masses of the turbine-generator driven by the power from the turbine shaft, and loaded/driven by the balance of the grid production – grid load. Synchronizing part of this model generates pulses for adoption of the turbine-generator speed to the grid frequency. The *Main Circuit Breaker (MCB)* models switch between no-load to load operation allowing simulation of load rejection and connection to the grid after a synchronization phase handled by the *synchronizer*.

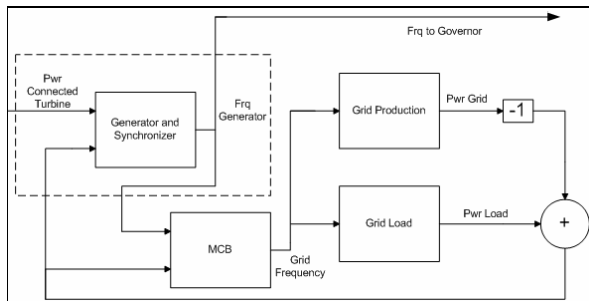


Figure 7. Overview of the power grid

The load model, *Grid Load*, represents the active power demand of the grid. The load of the grid is divided into three groups: resistive, frequency dependent and quadratic frequency dependent. The number of units within each group is decided through parameterization. The response to step changes of the load is simulated by using first order transfer functions.

As the grid load is constantly changing a normal distributed random number generator was added to the load of each group. In addition to this there is also the option to add a disturbance at any given time.

The *Grid Production* model is the representation of all other production units connected to the grid. The grid production units are divided into different groups depending on their response time. This enables simulation of different behavior depending on the types of power plants connected to the grid.

6 Mechanical machinery

Models of mechanical machinery cover main types of water turbines, Pelton, Francis and Kaplan. Tur-

bine models are complete with guide vanes and runner angle actuators.

6.1 Actuators and servo motors

The actuators are modeled as first order transfer functions with a time constant representing the actuator response time. Transfer functions are complemented by

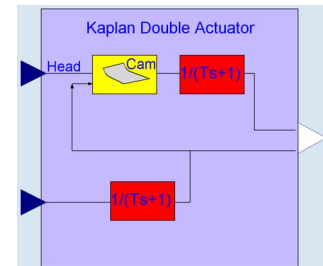


Figure 9: Kaplan servo

multiple speed limits to allow simulation of multi-step opening and closing of the guide vanes and runner angles. Adding play and hysteresis allows analysis of the influence of obsolete equipment. The model of the Kaplan servo motors includes a Kaplan Cam curve adjusted for the actual head of the water.

6.2 Models of water turbines

The simple approach to models of water turbines builds on the assumption that flow through the turbine can be estimated the same as for orifices, i.e. $Q_T = C_v \sqrt{\Delta p_T}$, where C_v is the guide vane opening factor. Power on the turbine shaft would be accordingly $P_T = Q_T \Delta p_T \eta$, where η is the total turbine efficiency. The pressure drop over the turbine, Δp_T , is available from the models of water ways. This simple approach could be satisfactory but only if there is tabulated data of C_v and η available. Both factors are functions of runner speed, water head, runner angle, etc. The problem will complicate further in case modeling interest is bound to phenomena of unusual runner situations, as at the turbine start-up, shut-down or load rejection. In reality such detailed information is not available or it would take a considerably long time and cost to get it.

HydroPlant library provides models according to this simple approach above or alternatively as detailed

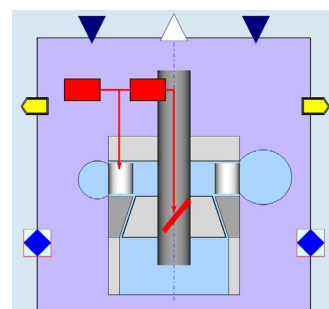


Figure 8. Icon of detailed Kaplan model

models calculating water velocity vectors on the inlet and outlet of the turbine runner.

The detailed model (shown schematically as Modelica icon in Figure 9) covers three essential volumes of the turbine; scroll case, guide vane and runner (GVR) volume and draft tube. Flows are calculated separately for guide vanes, for runner and for leakage through runner circumference from GVR-volume to draft tube.

Vectors of water flow velocities are shown in Figure 10 (from ref [3]); u is the velocity of the runner (1: inlet, 2: outlet), v is the velocity of the water flow in relation to the runner, c is the external velocity of the water entering (1) and leaving (2) the runner.

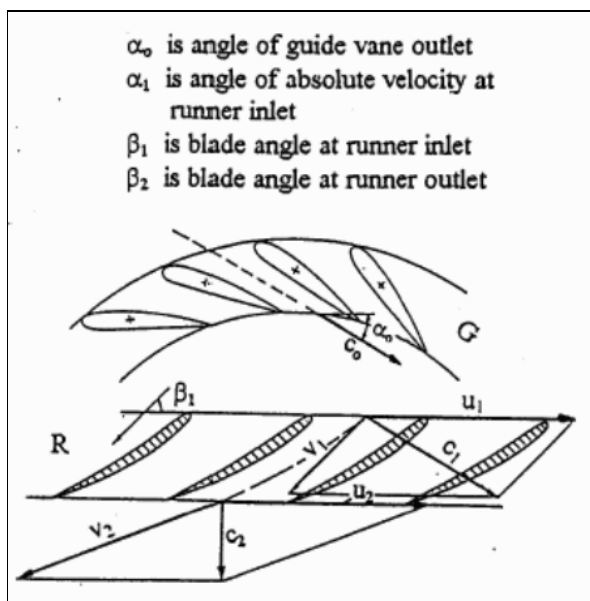


Figure 10: Angles of Kaplan water flow velocities

Power on the turbine shaft is now calculated from the change of the momentum on the runner:

$$P_T = m_{\text{dot}} \cdot \omega \cdot (r_1 \cdot c_{u1} - r_2 \cdot c_{u2}) \quad (5)$$

Vectors $c_u = c \cos(\alpha)$, for α angle between c and u vectors. One will ask naturally if information on all angles required here is easier to get than for factors C_v and η ? This information is calculated automatically by assuming the runner and guide vanes are well designed; it means mainly the following:

- Maximum efficiency is assumed at the nominal power at the nominal head and speed
- Vector v_1 is assumed at the nominal conditions to coincide with angle β_1 .
- Vector c_2 at the nominal conditions enters draft tube at $\alpha_2 = 90^\circ$

7 Control Challenges

The implemented controller is a standard PID turbine governor, complete with a feed forward for power change control. The error signal to the PI is:

$$e = \Delta f + ep \cdot \Delta P \quad (5)$$

where Δf is the frequency error, ΔP is the power error and ep is the speed regulation factor.

Setting of PID parameters depends on various control challenges.

Water level control: Each hydro power plant has an assigned maximum water level allowed in the reservoir for environmental reasons. If this level is exceeded the hydro plant is normally fined. Since the amount of water stored corresponds to stored energy, the power plants will try to be as close to this level as possible. Problems arise when the water reservoir level is close to the max allowed level and the power demand is low. In situations like this the plant will be forced to let a certain amount of water through the gates which is a waste of money for the plant and produces environmental costs of dumping large amount of water down the river. Tuning this kind of control system takes a long time due to the long time factor of a large reservoir which often results in settings of the controller for the worst case scenario. Using models to tune this kind of system would both save time and improve the performance of the controller.

Scheduling of turbine governor settings: Conventional turbine governors have normally two sets of PID parameters; one for no-load mode of operation and one for the generator synchronized to the grid. Digital governors allow a practically unlimited number of PID settings, but logic switching between those settings will become complex, error prone and difficult to verify. HydroPlant library was developed initially for testing different methods of adapting PID settings to the actual mode of operation, actual water way configuration and to the actual system load and dynamics. An adaptive approach requires on-line identification of the dynamics of both the power grid and the local plant.

Nonlinear plant behavior: Plants having complex waterways can be difficult to control. It will change dynamics depending on the power generated (nonlinearities). HydroPlant library will allow development of the simplified real-time models of the plants allowing continuous tuning of the PID settings.

Varying power grid: When tuning PID parameters or developing alternative control schemes, it is of im-

portance to know the dynamics of both the power grid and the local plant simulated. Identification of the grid is more complicated as the main grid information is provided in frequency signal only. Identification considers mainly grid size in relation to the size of the local generator. If that relation is large, the power plant will not affect the grid frequency noticeably (stiff grid) and the PID settings can be chosen more aggressively. But in the other case (soft grid), precautions need to be taken. In this case overshoots and oscillations in the local plant will affect the grid frequency and the PID settings should consider mainly grid stability. The problem gets more complicated as the grid changes, and the local generator works both on the stiff and soft grid. The HydroPlant library facilitates development of techniques for continuous, on-line, grid identification.

Schemes of joint control of plant units: The majority of plants run several turbine-generator units on the common water ways. There are serious control problems to be solved here to allow most efficient plant operation or to avoid certain power divisions causing vibrations, pressure oscillations or other forbidden working situations.

8 Simulation Results

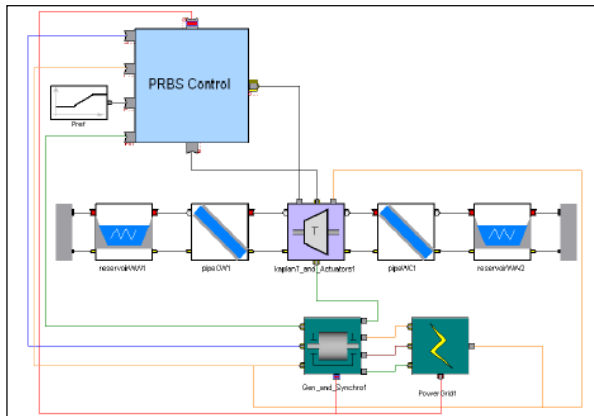


Figure 11. Model of a complete hydro power plant

Simulation results from the example of a simple hydro power plant (Figure 11) are briefly presented here. For a more in-depth result analysis please refer to [4], and to the HydroPlant manual [5].

8.1 No-load, synchronizing and loading

This example will illustrate a hydro power plant acting under no load and when connected to the power grid. Simulation starts when the grid load is increas-

ing and the grid frequency is falling below the nominal level (Figure 12).

The turbine starts and the governor (in no-load mode of operation) controls turbine speed to the generator's nominal frequency as it can be seen during the first 110s of simulation.

After 110s synchronization is initialized, by sending pulses to frequency set point in order to match the generator frequency to frequency of the power grid.

After 300s the frequency of the generator is synchronized to the grid frequency, the MCB is closed, the power reference is set to 50MW and new set of PID parameters is applied. Generator power output increases until new load balance is reached.

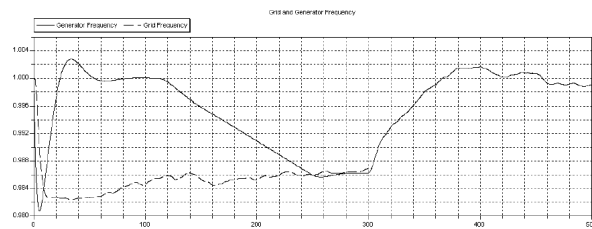


Figure 12. Frequency of the grid and local generator during start and synchronization

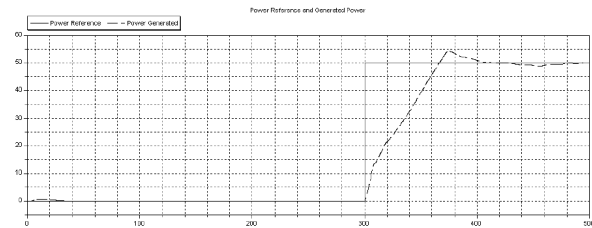


Figure 13: Loading of the local generator

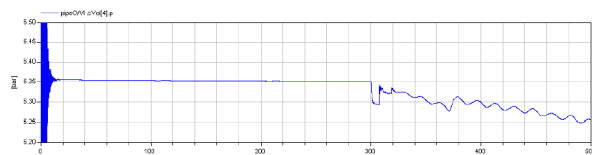


Figure 14: Pressure drops at the turbine runner due to the water acceleration in the penstock.

8.2 Load Rejection

The particular shape of the turbine velocity profile during load rejection is normally a guaranteed issue and the ultimate test of turbine governor quality and its settings.

Figure 15 presents opening of the MCB in 500s of simulation. The speed rises but after approximately 100 sec is controlled back to normal. Behavior of the guide vane and Kaplan angle servos can be studied on Figure 16. As the guide vane servo is faster than

angle servo, a large combination error can be seen. The combination error means a discrepancy exists between optimum angles leading water through the turbine, which affects the turbine efficiency drastically.

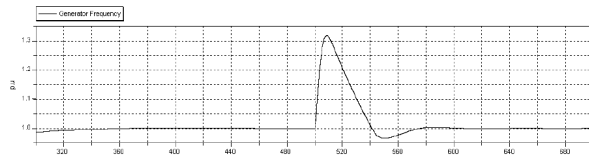


Figure 15: Generator frequency at the load rejection

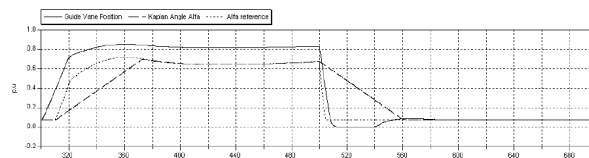


Figure 16: Guide vane position, Kaplan angle and angle reference

References

- [1] Elmqvist, H, Tummescheit, H and Otter, M (2003): “Object-Oriented Modeling of Thermo-Fluid Systems”. Dynasim Sweden, UTRC, USA and DLR, Germany.
- [2] Roberson, Cassidy, Chaudhry (1998): “Hydraulic Engineering. Second edition” John Wiley & Sons, New York, USA.
- [3] Kjølle, A.: Hydropower in Norway. Mechanical Equipment. A survey
- [4] Slättorp, K, Tuszynski, K (2005): “Model of a Hydro Power Plant – New Algorithm for Turbine Governors”, Master Thesis. Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden
- [5] HydroPlant Library. The Manual
(*may be distributed on request*)

Session 3b

Hardware in the Loop

Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation

Arno Ebner Anton Haumer Dragan Simic Franz Pirker
Arsenal Research
Giefinggasse 2, 1210 Vienna, Austria

phone: +43 50550 6663, fax: +43 50550 6595, e-mail: arno.ebner@arsenal.ac.at

Abstract

The paper presents a concept and an implementation of *Modelica* simulation interaction using the operating system inter-process communication method of the *Named Pipe*. The main aim of this presented work is to implement a hardware-in-the-loop simulation (HILS) environment based on *Dymola* which runs on a normal *Microsoft Windows* Personal Computer.

An energy storage test bench is connected by an analogue and digital data input/output card with the *Dymola* simulation computer. With this proposed system, particularly long-time simulations with sample rates up to 30 Hz can be executed very cost effective. Typical applications are simulations of drive cycles to test energy storage systems in electrified vehicles such as batteries or fuel cells. Other application examples are the verification of battery models, thermal management models or battery management system (*BMS*) models.

In this paper all methods used for implementation are described in detail. Especially the concept of inter-process communication and the concept for real-time and simulation time synchronization is discussed.

An application example which uses the provided concept is also shown at in this paper. In this example a longitudinal simulation of a vehicle is presented. The startup phase of the internal combusting engine model and a short drive cycle in combination with a connected real battery is shown.

1 Introduction

The traditional approach for simulating technical or physical systems is to describe real systems in mathematical models. These systems are described with discrete equations or with continuous algebraic and differential equations. The simulation environment has a solver algorithm which generates a solu-

tion for the system model considering the initial values.

In some cases the user or other applications have to interact with the simulation, for example to:

- Start or interrupt the simulation
- Change parameters or variables during the simulation
- Communicate with other applications, for example with another simulation environment
- Exchange data with an input/output-card or a peripheral communication interface
- Build up a hardware-in-the-loop simulation environment

Hardware-in-the-loop (HIL) is the integration of real components and system models in a common simulation environment [1]. This means that some parts of a system, which should be tested, are virtual and other parts are real. HIL simulations are an important method for the development of mechatronic systems. An important advantage of HIL is that it allows function tests of mechatronic systems or components of such systems under simulated real conditions. Therefore HIL helps to save cost and time compared to conventional test runs on a real prototype.

There are three important considerations for the implementation of a hardware-in-the-loop simulation:

- The simulation of the dynamic system, in other words the mathematical or physical models must be processed in real-time.
- There must be synchronization between the time in the real world (the so called real-time) and the digital simulation-time of the simulation tool.
- The simulation tool must be able to communicate e.g. with others applications or an I/O communication interface.

The *Monitoring, Energy and Drives* division at *Arsenal Research* does research and development on components for Hybrid Electric Vehicles (HEV's) and electrified auxiliaries for vehicles. For that they acquire know-how in the simulation of electric

drives, of energy storage systems and in vehicular simulations. A great deal of simulation models was built up in *Modelica* and simulated with *Dymola* [2], [3]. In order to verify and validate the implemented models and the developed system components, a connection to a hardware-in-the-loop simulation environment has to be implemented.

In that way in the provided paper an interaction of a *Modelica/Dymola* simulation with components outside of the simulation tool is shown. At this proposed HIL system the *Dymola* application runs on a standard Microsoft Windows Personal Computer. Actually two important processes run on the simulation PC: one process is the *Dymola* application and one process provides the input and output functionality for the peripheral card. Details on implementation for both tasks, first for the *Dymola* application using external functions and furthermore for the peripheral card application are given below.

This two processes communicate together using the inter-process communication (IPC) methods provided by the operating system. In the proposed case the tasks communicate via a so-called *Named Pipe* mechanism. Through the first-in first-out behavior of the *Named Pipe* communication method there is an implicit synchronization between the two processes. In this paper is described how to create and to open, how to write to and how to read from the *Named Pipe* and how to close it.

2 Inter-process Communication with Named Pipes

The IPC method that is used in this work is the so-called *Named Pipe* mechanism. This IPC method is available both on *UNIX* systems and on Microsoft Windows systems. The semantic and the function calls differ in the operating systems but the concepts are the same [4], [5]. Only the implementation on a *Microsoft Windows* operating system is shown in this paper.

Named Pipes are designed for the communication between the pipe server and one or more pipe clients. They have first-in first-out behavior and can be accessed like a file.

Stdio.h standard *C* library functions for file handling, such as *fopen()*, *fclose()*, *fread()* or *fwrite()* can be used to deal with *Named Pipes*. However, the usage of Microsoft Windows *Software Development Kit (SDK)* functions gives a more powerful access on the functionalities of *Named Pipes*. Specifically, using the *SDK* a pipe server calls the *CreateNamedPipe()* function to create an instance of a *Named Pipe*. The

client calls the *CreateFile()* or *CallNamedPipe()* function to connect to an instance of the *Named Pipe*. *ReadFile()* and *WriteFile()* functions allow reading and writing to a specified *Named Pipe*.

In the presented work the *Dymola* process corresponds to the server process, which generates the pipe and waits for a connection with a client (for instance the I/O process). This process then executes the simulation steps, puts data in the communication pipe and gets data from the client process.

3 Implementations in Modelica and Time Synchronisation

The described mechanism of inter-process communication is implemented as an external function written in *ANSI/ISO C*. At *Arsenal Research* two main *C* functions for the communication of *Modelica* with other processes using *Named Pipes* are developed.

The *AllocateResources()* function creates and connects a *Named Pipe*. The function *PipeIO()* computes a string from the simulation variables array calculated by *Dymola* and writes this string in the *Named Pipe*. Then the function gets the data string from the *Named Pipe* and computes an array of *Real* variables for the *Dymola* simulation solvers.

In Figure 1 the flow chart of the functionalities of the server process is shown. Specifically, the function *AllocateResources()* and the function *PipeIO()* are described. For these functions a static library is build. This library is linked to the model using the function definitions. In *Dymola* a wrapper function to convert the external *C* function to a *Modelica* function is defined.

For a working HILS it is important, that the simulation time in the simulation application is synchronized with the real time. This synchronisation in *Modelica* is achieved by using the when clause in an algorithm statement:

```
...
when (time >= SimuNext) then
  (SimuNext, input) := PipeIO(time, output);
end when;
```

In this short code fragment *time* means the simulation time in *Dymola* and *SimuNext* means the real-time. *input* means an array of *Real* variables taken from external *C* code to *Modelica* and *output* means an array of *Real* variables taken from *Modelica* to external *C* code.

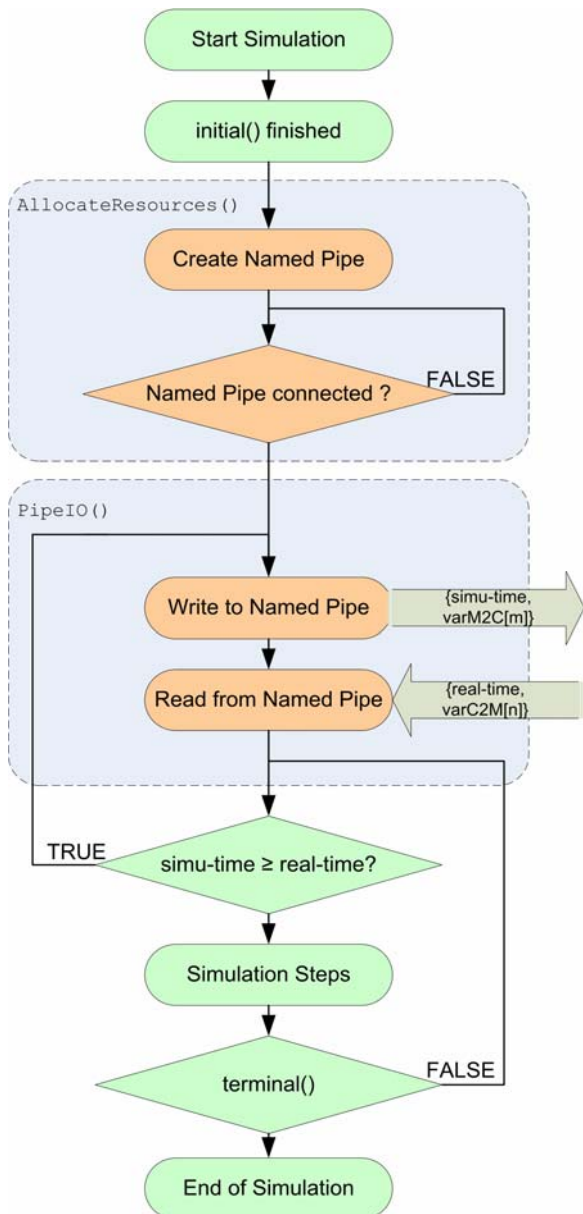


Figure 1: Flow chart of the server process, which runs as an external function call of Dymola.

The simulation process gets the real-time information from the client process via the *Named Pipe* IPC. As long as the real-time is greater than the simulation time in *Dymola* a new inter-process communication cycle get processed and the function *PipeIO()* gets called. As long as the real-time is smaller than the simulation time new simulation steps of the *Dymola* solver get executed.

In figure 2 the client process of *Named Pipe* IPC is shown. In this process the data in- and output functions and the calculation of real-time information for the simulation application using functions from the *ANSI/ISO C* library *time.h* are executed.

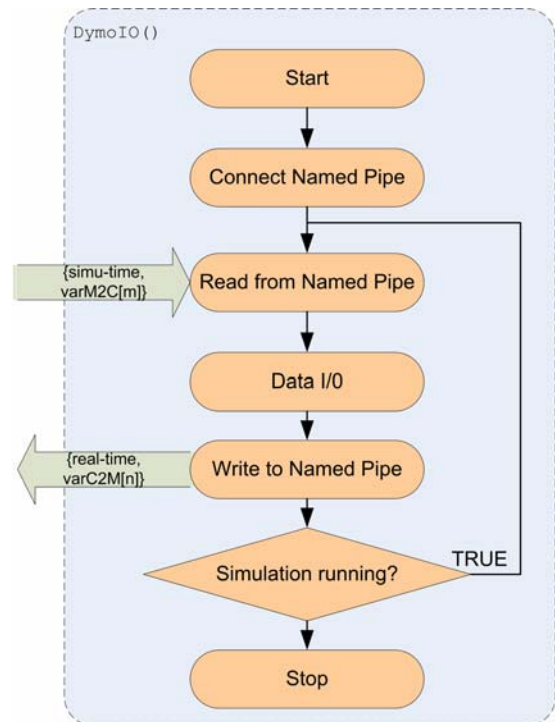


Figure 2: Flow chart of the client process.

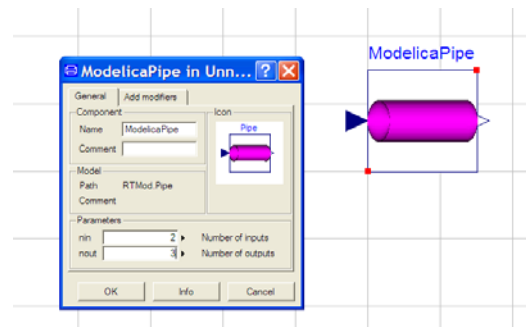


Figure 3: Icon and parameter settings of the Pipe block.

All inter-process and simulation time synchronization functionality are combined in the so called *Pipe* block. This block is shown in figure 3.

4 Application Example: Hardware-in-the-loop Energy Storage Test Bench

The HILS environment presented in this paper is a very cost efficient system which is based on a normal *Windows* PC platform.

The communication between the simulation PC and the energy storage test bench is realized by an analogue inputs and outputs card. Specifically, a commercial data acquisition (DAQ) Card from *National Instruments (NI)* was used for the implementation. By using the *DAQ* functions from *NI* the simulation

tool can communicate with the interface card [6]. The main components of the energy storage test bench are a stack of electronic power supplies and a stack of electronic loads.

The technical data of the energy storage test bench is:

- Voltage: 0V-600V
- Current: 600A charge and 750A discharge
- Peak Power: 96 kW

The target applications of this HIL-test bench are the development and the test of energy storage, thermal management and battery management models.

The concept diagram for this HILS-environment is shown in Figure 9. In this application example the vehicle model calculates the loads for the battery. During the simulation the instantaneous real battery conditions (Voltage, Current, and Temperatures) are considered by *Dymola*. The electric drives are modelled using the *SmartElectricDrives* library [7], [8].

The following simulation results show the first 30 seconds of the *New European Drive Cycle (NEDC)* of a conventional vehicle. After 2 second the internal combustion engine of the vehicle is started for 2 seconds, which is illustrated in Figure 4. Figure 5 shows the shaft speed of the ICE. It appears that the motor is running with idle speed after 4 seconds. During the starting process the starter motor has a defined current demand for the battery. In figure 6 this current demand is shown. At time $t=4s$ the ICE is running in idle speed and at time $t=10s$ the vehicle begins to drive following the NEDC. In these phases the alternator produces electrical power, which charges the battery of the vehicle.

With the proposed HIL interface the electronic load creates a real electric current drain for the real battery. The electronic power supply generates a real charging current for the real battery connected to the energy storage test bench. In figure 7 the measured current in the battery pins and in figure 8 the real voltage at the battery pins is shown.

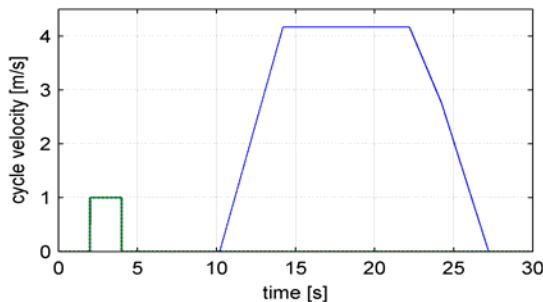


Figure 4: Starting the ICE after 2 seconds for 2 seconds and simulate first 30 seconds of the NEDC

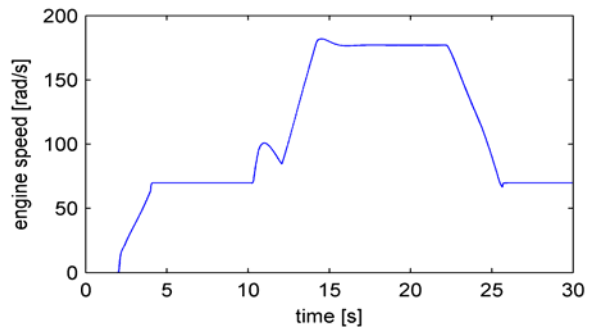


Figure 5: Simulation result, shaft speed of the ICE

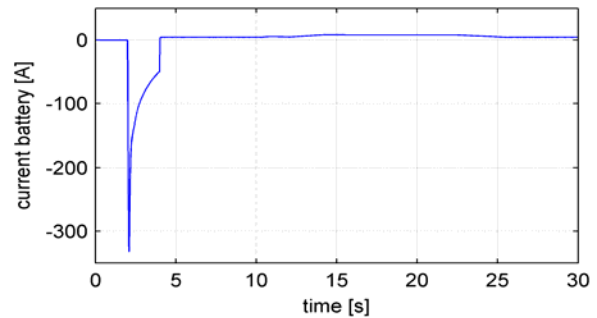


Figure 6: Reference current for battery – output of the HIL simulation

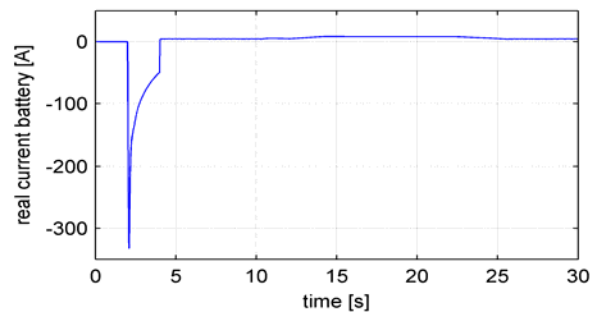


Figure 7: Real current in battery– input of the HIL simulation

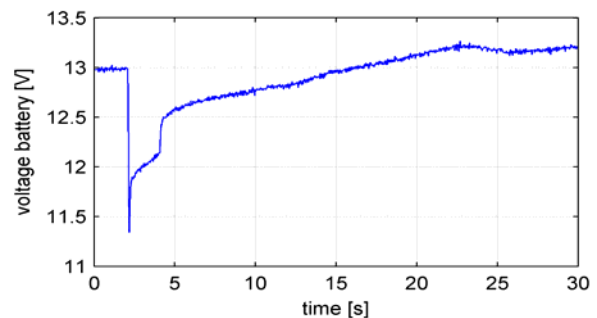


Figure 8: Real voltage at battery– input of the HIL simulation

This HIL simulation experiment was executed on a *MS Win32* PC with *Intel Pentium Mobile* processor with 1.8 GHz clock. In *Dymola* the *Dassl* integration method is used with a tolerance of 0.001. 25Hz Input/Output communication frequency was chosen.

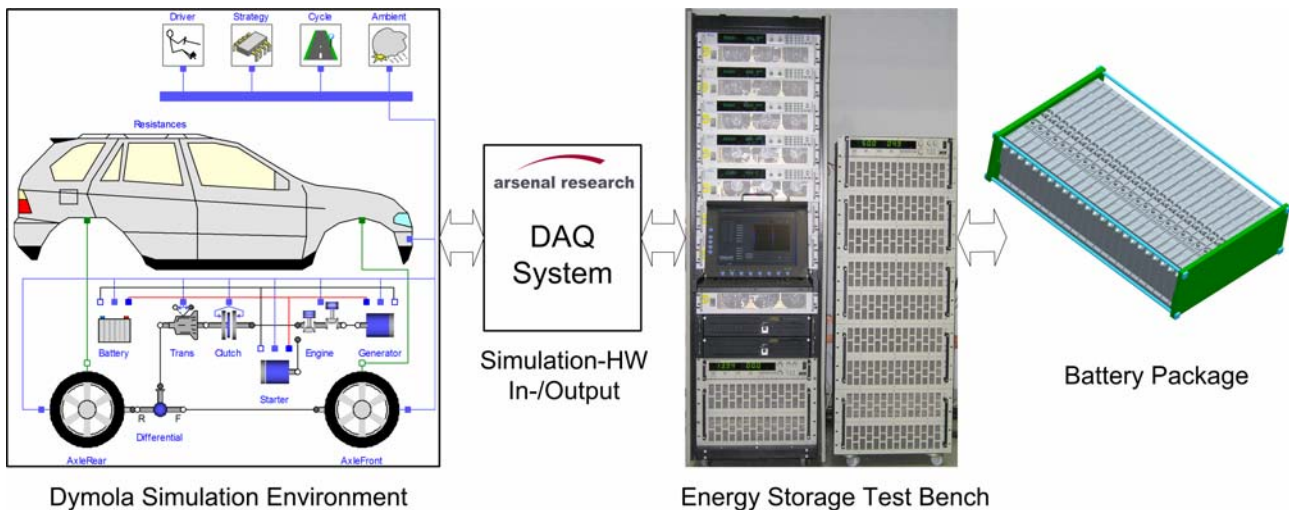


Figure 9: Application example, functional diagram of the HILS energy storage test bench

5 Conclusions

The *Modelica* hardware-in-the-loop simulation approach using Named Pipes inter-process communication methods was shown in this paper. This proposed system does not need an expensive “true” real-time platform. It bases on a normal *Windows* PC with a *Dymola* simulation environment and a communication interface card for inputs and outputs. Complex models can be simulated with an in-/output sample rate up to 30 Hz.

The HIL energy storage test bench which was proposed as application example in this paper is under operating conditions at *Arsenal Research* since the begin of 2006. With this HIL test platform particularly long-time simulations such as drive cycles test of energy storage systems or verifications of battery models, thermal and battery management models was done.

As future work *Arsenal Research* will implement the proposed inter-process communication algorithm also on a *Dymola* application which runs on a *Linux* PC.

Abbreviations

BMS	Battery Management System
DAQ	Data Acquisition
HEV	Hybrid Electric Vehicle
HIL(S)	Hardware-in-the-Loop (Simulation)
ICE	Internal Combustion Engine
I/O	Input/Output
IPC	Inter-process communication
NEDC	New European Drive Cycle

References

- [1] Verein Deutscher Ingenieure, VDI Richtlinie 2206, Entwicklungsmethodik für mechatronische Systeme – Design methodology for mechatronic systems, June 2004.
- [2] Dymola, Dynamic Modeling Laboratory, User’s Manual, <http://www.Dynasim.com>: Dynasim AB, 2004.
- [3] P. Fritzson, Principles of Object-Oriented Modelling and Simulation with Modelica 2.1. Piscataway, NJ: IEEE Press, 2004.
- [4] Microsoft MSDN Library, – Interprocess Communication – Named Pipes, http://msdn.microsoft.com/library/en-us/ipc/base/named_pipes.asp, 2006.
- [5] Elmenreich W., Systemnahes Programmieren – C Programmierung unter Unix und Linux, 2002.
- [6] National Instruments Document, DAQ - Traditional NI-DAQ User Manual, Version 7.0, April 2003.
- [7] D. Simic, H. Giuliani, C. Kral and F. Pirker, Simulation of Conventional and Hybrid Vehicle including Auxiliaries with Respect to Fuel Consumption and Exhaust Emissions, SAE World Congress, Detroit, 2006.
- [8] H. Giuliani, C. Kral, J.V. Gragger, F. Pirker, Modelica Simulation of Electric Drives for Vehicular Applications - The Smart Drives Library, ASIM, Erlangen, 2005.

Parametrization of *Modelica* Models on PC and Real time platforms

Matthias Kellner Martin Neumann Alexander Banerjee Pritesh Doshi

ZF Friedrichshafen AG

Graf-von-Soden-Platz 1, D-88046 Friedrichshafen, Germany

matthias.kellner@zf.com

martin.neumann@zf.com

alexander.banerjee@zf.com

Keywords: model based development, dynamic model parametrization, SW-Tests, ZBF-Parameter, Realtime

1 Introduction

Throughout the development process of control units for new transmission system, computer models are needed to perform different tasks, such as concept evaluation and the design and testing of controllers in MiL, HiL and SiL environments. These models will be used within different CAE-tools and different environments. To avoid redundancies and sources of errors the parametrization of these models using the same set of parameters is preferred, since these will change during the development process. Furthermore, the parameters will be kept within at one location. The only possibility to deal with this problem is to keep models and parameters separated, which means that models have to be parameterized using a set of files. Unfortunately, some environments do not allow file I/O operations. Even though no file I/O operations are available, for example on Real time platforms, there is still a strong need for flexible parametrization. Several approaches have been developed to overcome these challenges, especially for Dymola/Modelica models which will be presented within this paper.

In the following chapter the integrated use of vehicle models within ZF electronics development will be introduced. In chapters 3 and 4 the ZBF-parameter format will be discussed as well as the different parametrization approaches. With the help of an example the use of one of the approaches will be illustrated. Finally the results will be summarized and open questions will be addressed.

2 Integrated use of powertrain models within ZF electronics developments

Within this chapter the use of simulation models within the ZF electronics development will be illustrated, concentrating on passenger car applications. The focus will be an integrated use of these models from specification phase up to series application. The use of Dymola models within Hardware in the Loop (HiL)-simulation requires some adaptations within Dymola for parametrizing models on real time platforms.

Rising demands for better comfort, more power and lower fuel consumption as well as increased integration of different systems lead to a higher weighting on software development. Although the development period has to decrease, the quality of the software and the level of customer satisfaction have to be continuously improved. For validation and verification purposes almost 40% of the total budget for software development has to be invested [1].

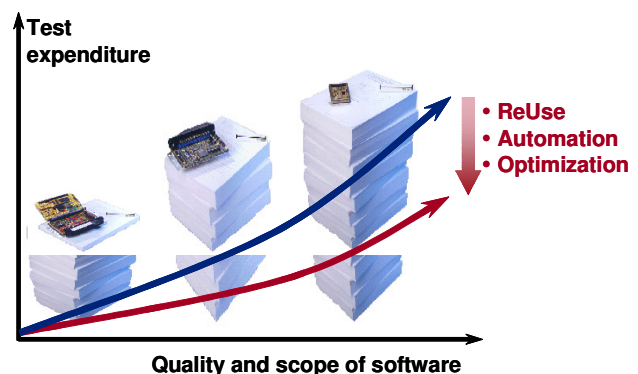


Figure 1: Test Expenditure of Transmission Software

As can be seen in Figure 1 the necessary testing expenditures increase disproportionately due to higher quality requirements. These ever increasing demands, which are closely linked to costs, can only be taken care of by improving efficiency. The application of ReUse, automation and optimization of testing processes can reduce the testing expenditures up to about 30% [2].

In order to reduce the time used for developing a new product the parallelization and decoupling of mechanic, electronic and software development is needed. By reusing models within different testing environments the service expenditures can be reduced. Testing at an early stage in development will also lead to a decrease in development expenditures.

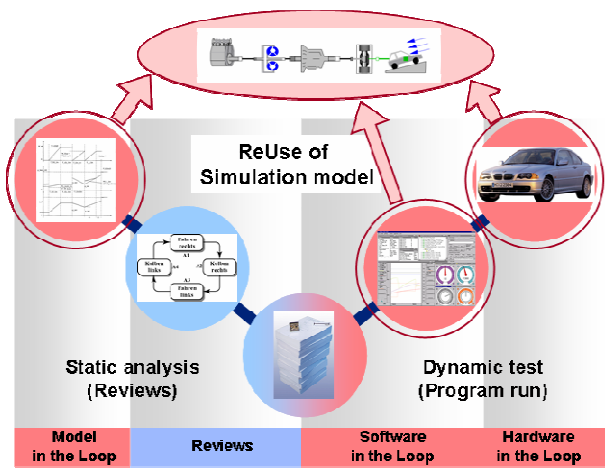


Figure 2: Testing as part of the software development process

To ensure a development of software in parallel and independently, appropriate environments for development and testing are needed. These should also include simulation capabilities. In order to service the software development process, different simulation and development tools are in use, which allow the simulation of complex full vehicle models as well as for testing control units on HiL test rigs. All tools have to be suitable for an integrated model based development process orientated along the V-model approach and also must provide thorough analysis possibilities which can facilitate the finding of errors at an early stage. For the modelling of powertrains the software tool Dymola is thought to be the standard.

The electronic development simulation models are primarily in use for testing the functionality and suitability for the series application of control unit software [4]. These tests are an essential part of the software development process.

Dynamic models are needed to test software independently without involving the mechanic and electronic hardware. With the help of different testing environments the software can be tested at each level of maturity.

Figure 2 shows the development phases and the adjoined testing environment and methods. The testing environments Model-in-the-Loop (MIL), Software-in-the-Loop (SIL) and HiL cover the whole process of software development. Hence models are not only needed in the conceptual phase, the left part of the V-model, but rather in the testing phase, presented on the right side of the V-model. In order to minimize service expenditures a unitary use of simulation models within all testing environments must be stipulated. The consequent application of the ReUse-concept does not only reduce the service expenditures for about 50%, but also reduces expenditures for integrating these models in the development process while heavily simplifying the version management.

Since in reality the integration of ZF products strongly varies among the different customer applications, a vast amount of different models are needed. One major objective is then to generate a universal model that can be configured for the appropriate customer application by solely changing model parameters.

One major disadvantage of Modelica is the inability to easily parametrize models for different development platforms. Therefore, an approach has been developed in ZF that has the ability to separate models from the parameters. The model parameters are stored within standardized ZF-ASCII-files, which will then be loaded at initialization [3]. In order to ensure a parametrization of models with ZBF-data on platforms without file I/O, e.g. dSPACE, modifications and adaptations within Dymola have been done. These procedures will be described in the following chapters.

3 ZBF-Format and Parametrization of models on platforms With File I/O

The ZBF-Format will be discussed in chapter 3.1, with an emphasis on its advantages in comparison to other parameter formats. In the following paragraphs an approach will be discussed which enables a parametrization of models by using ZBF-data within environments with File I/O.

3.1 ZBF-Parameter

As stated in the previous chapter, there is a strong need to separate models from parameters. A detailed description of the ZBF-format can be found in [3]. For the sake of completeness an example of the ZBF-format is included in Figure 3.

```
J1 [kgm^2] 0.1
; scalar parameter
InU [-] 0 1 2
OutY [-] 0 1 2
; two vectorial parameters
Test_Table2D[
[-] U1 [-] 0 1 2
2 Y [-] -2 -1 0
1 Y [-] -1 0 1
0 Y [-] 0 1 2
Test_Table2D]
; Two-Dimensional-Table
```

Figure 3: Parameterization with File I/O

ZBF originated from the strong need for exchange of formatted data between different Excel-programs. Thereinafter a broad use has been promoted for C and C++ calculation programs. It has been finally declared as a standard within ZF.

A big advantage of the ZBF-format is that it allows the provision of parameters which do not comply with SI-units, something usual for transmission design (e.g. [rev/min] instead of [1/s]). New approaches such as XML are not in use, since a large amount of programs are already able to read ZBF-data files. Furthermore, it is quite simple to transfer ZBF-data-files to Excel and edit these data files by using simple test editors.

With the help of an easy example, the differences of these different formats can be illustrated. A scalar parameter in ASCII such as the moment of inertia of the engine can be given as:

ZBF:
JMot [kgm/s^2] 1.5

XML:
<Identifizierer>
<name>JMot</name>
<einheit>kgm/s^2</einheit>
<wert>1.5</wert>
</Identifizierer>

NetCDF:

```
netcdf motor{
dimensions:
One = 1;
variables:
float JMot(One);
JMot:long_name = "Motor-
trägheitsmoment";
JMot:units = "kgm/s^2";
data:
JMot = 1.5;
}
```

3.2 Parametrization of models on platforms with File I/O

For the development of control function, models are used by CAE-tools running on PC-platforms with a file I/O operating system. A description of the approach on how to parameterize models on platforms with file I/O has been given in [3]. A short summary of the approaches in the forthcoming chapter, together with the necessary terms, will be presented next.

The parameters which are usually scalars, vectors and matrices are stored separately from the model at a central location (Figure 4). In order to read these data files, an appropriate parser will be linked to the model at the time of compilation. With the help of the parser, the model then reads all the necessary data at initialization and all parameters will be stored within a special data structure.



Figure 4: Parameterization with File I/O

The use of self developed C-Functions which will be linked to the model during compiling help to find the parameter and assign it to the component. Hence the model can be implemented within different CAE-applications, which run on an operating system with file I/O. The big advantage is that there is no need to modify the parametrization process based on what is required for the present application.

4 Parameterization of models on platforms Without File I/O

For controller testing, Dymola models have to be implemented in environments, such as SiL and HiL, with programs that do not allow for file I/O operations, such as dSPACE. Therefore another method has to be used. Moreover, the strong need for rapid prototyping calls for flexible parameterization within these environments. In the following chapter two realizations will be discussed. The first one will be referred to as “static parameterization” and the second “dynamic parameterization”.

4.1 Static Parameterization

Within environments that do not allow file I/O operations one straight forward approach is to attach the parameter files to the existing Code. This will be done by converting the ZBF-files into C-code files and storing the parameters in a single character-string. Afterwards, these files will be linked to the model including the Parser throughout compilation (Figure 5).

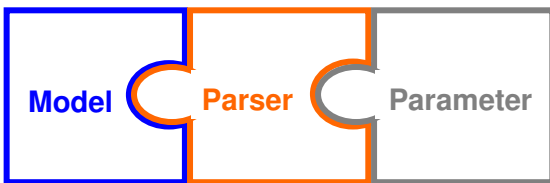


Figure 5: Static Parameterization

A slight extension of the existing Parser algorithm allows for proper parsing of the string and hence parameterization of the model at initialization. This approach is very useful in situations where parameters do not change very often or the model has to be exported as a single binary source.

4.2 Dynamic Parameterization

In order to test the robustness of a controller for various model settings, the static approach can be extended. This is done by changing the parameters directly within the code. Therefore a method is used

which has been applied in dSPACE for easy re-parameterization of models on their hardware.

For this purposes the Dymola model will be imported into Matlab/Simulink as an S-Function. An extra parameter will be added to the S-Function during its generation by modifying the SimStruct to Dymola interface file *ss2dym.c*. Using this additional parameter the new set of ZBF parameter files can be passed on to the model in the form of an array of double values. An array is generated by Matlab from the default set of parameter in order to locate the parameter memory which will be needed later for re-parameterization (Figure 6).

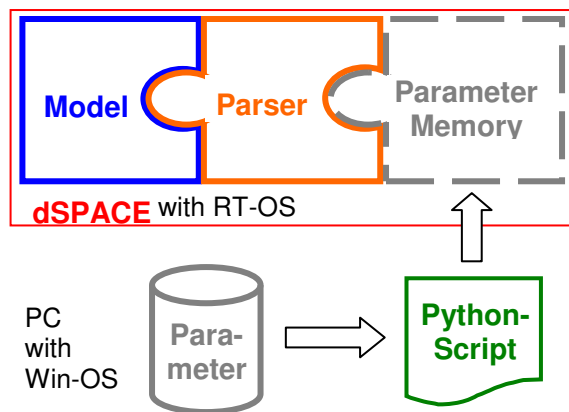


Figure 6: Dynamic Parameterization

The model with the additional S-function parameter (Figure 7) is exported into the dSPACE Real-Time platform using the Matlab RTI workshop. While exporting the model into dSPACE, the double array is converted into C-Code and subsequently linked to the model. This guarantees that the appropriate memory space can be accessed for dynamic re-parameterization. An SDF-file for dSPACE simulator is generated as well.

Finally the parameters can be transferred from the PC to the computer with the RT-OS with the help of Control-Desk. Fortunately, the re-parameterization can be done outside of Matlab. With the help of a python script the model parameter files will be re-parsed on the PC-platform with a regular file I/O, where the parameters will be converted into a double array of the same structure as the one for exporting purpose.

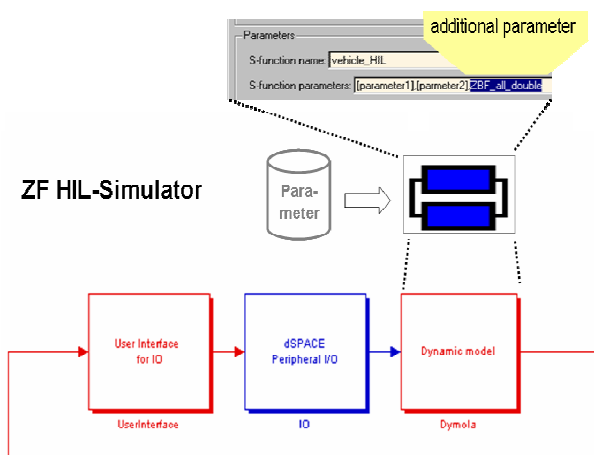


Figure 7: Additional S-Function Parameter

In order to “reload” the new parameters, the default parameters within the Real-Time model are accessed by another python script using the read/write routines from *rtpLib* and ControlDesk. The length of the array is matched with that of the default array and the default parameter in the model is overwritten with new one. Finally, the initialization flag is activated and the model is re-initialized.

5 Dynamic ZBF-Parametrization of a passenger car model on dSPACE-HIL-Simulator

For Dymola vehicle models at ZF, all relevant mechanical, electrical and hydraulic modules needed for software development and HiL testing have been modelled.

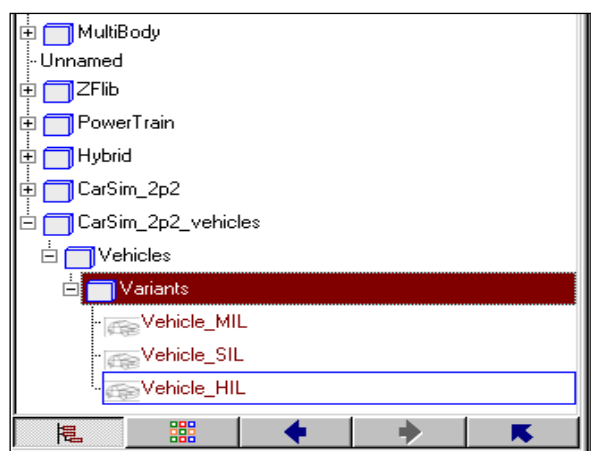


Figure 8: Modelica Libraries for HIL Tests

With the help of commercialized model packages, for example PowerTrain and MultiBody, as well as ZF-specific packages, as ZFLib, Hybrid and CarSim, powertrain models can be developed for different testing purposes (vgl. Figure 8).

In Figure 9 a vehicle model is shown as it is used for HIL-Simulation at ZF. It consists of the following sub-modules: engine, alternator, torque converter with torque converter clutch, ZF automatic transmission, rear axis, simple vehicle model with brakes, Control units (electrical/hydraulic), signal bus and I/O-interfaces.

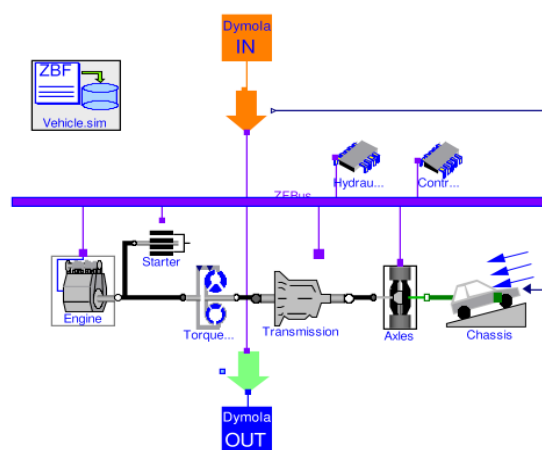


Figure 9: Dymola-Vehicle used in MIL, SIL and HIL environments

The degree of detail in the powertrain modules has to be adapted according to the field of application. Models which are used for testing purposes require a thorough consideration of internal interactions. For example the interaction of a set of clutches while doing a change in ratio requires a detailed application of system hydraulics modelling. Whereas the vehicle module has been simplified to a minimum and the engine has been represented by look-up tables in order to guarantee real time capability.

Due to the fact that ZF products will be implemented in different vehicle settings there will be a vast variety of models. The only possibility for dealing with this situation is following a modular approach, storing modules in libraries and separating models from data. The models can be parametrized by using data sets which relate to a specific version of vehicle set up. The basis for the parameter format is the ZBF-format, which has been described in chapter 3.1. The parametrization process of a vehicle model which is used for HiL-testing of a control unit on a dSPACE is explained in the following section.

The parameters have to be stored in ZBF-data files as well as an appropriate allocation within the model has to be done. Afterwards the model will be in-

cluded in a Simulink block. For the first time generating the model S-function the parameters will be read from file and stored in a double array within Matlab. At the same time the S-function will be supplemented by an additional parameter, which most often is referred to as the third parameter. This parameter is the essential link to the double array. Whenever the model is transferred to dSPACE by applying the RTI-Workshop the double array will be converted into a C-File and afterwards linked to the model. All described steps will be done automatically.

With the help of the described dynamic parametrization approach (see chapter 4.2) model parameters can be changed on the dSPACE-simulator. This is done by applying appropriate Python-scripts which allow for an easy change of parameters without starting the implementation process once again. This approach is quite essential, since not all dSPACE-HiL-simulators at ZF provide a Matlab development environment.

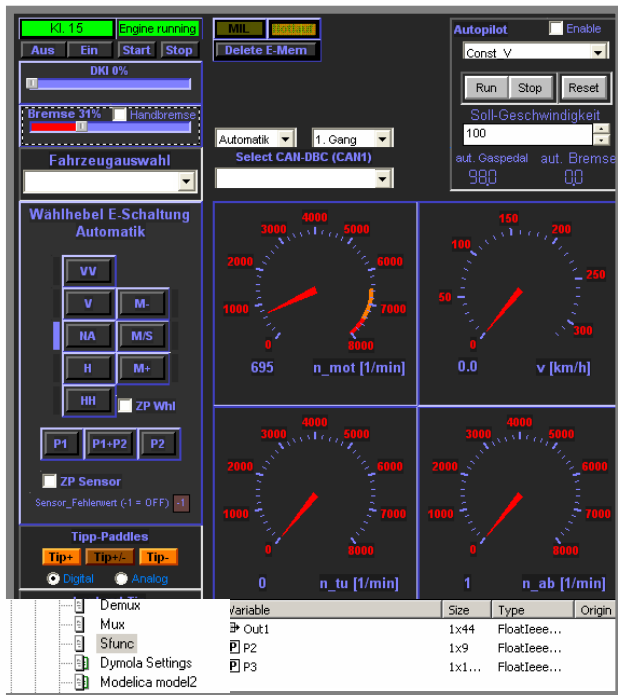


Bild 10: ControlDesk Interface

Applying ControlDesk a graphical I/O user interface is set up, which interacts with the model on the dSPACE board. All necessary inputs and outputs of the testing environment can easily be monitored or changed. Typically the hardware configuration, e.g. the control unit variant and CAN-Bus-system can be selected. All control inputs of the Dymola-model such as ignition, selection of gear ratio, throttle and brake pedal position can be changed manually or automatically.

Finally by applying the mentioned Python-scripts within the windows-OS the appropriate set of ZBF-parameters of a vehicle variant will be read from file. By activating the third parameter within the model-SDF-file, parameters will be mapped into the allocated memory space of the model which is implemented on the dSPACE platform. Hence parameters can easily be changed while the model is operating at running time.

6 Summary and Outlook

In order to use models within different tools and environments models and parameters have to be kept separate from each other. Within ZF these parameter files are set up according to a standardized description referred to as ZBF. Approaches have been developed which enable a uniform and unanimous use of these files on all simulation platforms independent of whether they provide file I/O routines or not. For environments with file I/O, typically PC platforms, an approach based on linking a Parser algorithm to the model has been outlined in [3]. For environments without file I/O two realizations referred to as static and dynamic parameterization have been developed, where the latter allows for flexible parameterization. The static method generates a single source from model, parser and parameters. The dynamic method utilizes the method used by dSPACE. With the help of a Python script, the parameters will be read from ZBF files and directly mapped into the parameter memory of the model, hence facilitating the modification of parameters on Real-Time platforms.

Future work includes the issue of overruns occurring at initialization, presenting opportunities for improvements, especially for some specific environments which do not allow for overruns even at initialization. The optimization of code can also be possible by applying the parameter evaluation feature within Dymola, which changes parameters into numbers and hence simplifies the code. Presently, this is not possible whenever parameter-files are in use. An extension which allows for optimization even when parameter files are in use can be very helpful if a more efficient Dymola code is to be developed.

7 References

- [1] G. Bauer, M. Gromus, M. Neumann and C. Tapia. Model-based software development in production applications with a closed-loop controlled lockup clutch in a ZF 6-speed transmission, Fisita 2004
- [2] H. Deiss, B. Aumann, T. Schober Time to Market in der Softwareentwicklung - Reuse und Standardisierung bei Getriebesteuerungen - Elektronik im Kraftfahrzeug, Baden-Baden 2000, Germany
- [3] J. Köhler and A. Banerjee Usage of *Modelica* for transmission simulation in ZF, pp. 587-592, Gerhard Schmitz, Editor, Proceedings of the 4th International Modelica Conference, Hamburg March 7-8, 2005, Germany
- [4] R. Gonzelez-Ramos, M. Neumann, A. Banerjee and J. Köhler Standard drive train models for increased Testing Efficiency, pp. 243, Proceedings of the 4th IAV Symposium, Berlin Juli 9-10, 2005, Germany

Synchronising a Modelica® Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System

Dietmar Winkler Clemens Gühmann

Technische Universität Berlin

Department of Electronic Measurement and Diagnostic Technology

Sekr. EN13, Einsteinufer 17, 10587 Berlin

{Dietmar.Winkler,Clemens.Guehmann}@TU-Berlin.de

Abstract

The modeling language Modelica® is widely used by the automotive industry. In connection with Hardware-in-the-Loop (HiL) testing it can accelerate the development process enormously. This paper presents the application of Modelica® models for a Hardware-in-the-Loop simulation using a highly dynamic engine test-bench system. Certain steps have to be taken to be finally able to connect the real-time Modelica® model to the test-bench system. One of the most important issues when connecting a simulation model to a hardware device is the synchronisation process between them. This includes the determination of interface signals, the adaptation of the models according to existing interfaces, and the actually online test of the new real-time adjusted model. All these parts shall be explained in this paper.

Keywords: Hardware-in-the-Loop simulation, real-time, RT-LAB, engine test-bench system

1 Introduction

Nowadays the car manufacturers try to reduce the development times of new cars in order to cut the costs and therefore stay competitive. At the same time the manufacturer is interested in the potentials of new engine developments in terms of fuel efficiency and exhaust-gas emissions. This results in engine tests being carried out on so-called engine test-benches instead of using roller test-benches or expensive test drives. The advantage of an engine test-bench is that one does not need a prototype car into which the engine has to be mounted. A detailed model of the cars drive-train is sufficient to yield fuel saving and exhaust-gas emissions measurements from the engine test-bench. Engine calibrations can be carried out at

a very early development phase using an engine test-bench for example.

Because of a cooperation of our Department of Electronic Measurement and Diagnostic Technology of the Technische Universität Berlin with the IAV GmbH Berlin, our department has access to a highly dynamic engine test-bench system. This test-bench system is used in connection with the model based calibration of electronic control units (ECU) of engines and transmissions [1]. As a next step object-oriented Modelica® models¹ will now be used to simulate the behaviour of all parts of the vehicle except the engine.

2 Hardware-in-the-Loop system

This section provides some more details about the applied HiL system.

The HiL system consists of a highly dynamic engine test-bench and a *HiL simulator*. The principle setup is depicted in Fig. 1.

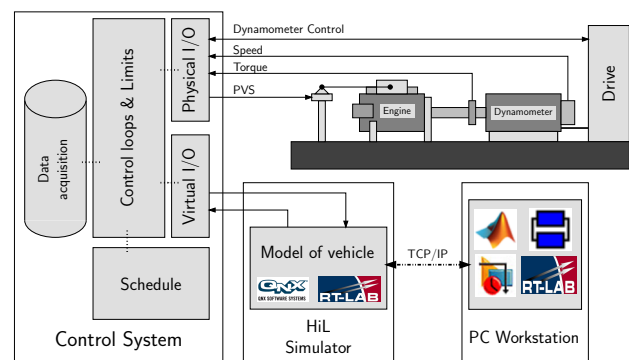


Figure 1: Principle setup of the highly dynamic engine test-bench system – “Test-Bench of the Future”

¹Modelica® is a free modelling language developed by the Modelica Association → www.modelica.org

2.1 Test-bench system

The highly dynamic engine test-bench system consists of a combustion engine which is directly coupled with an electric *Dynamometer* which in turn is controlled by a power electronic converter (*Drive*). In addition to that a *Control System* is needed to supply the needed control signals (e.g. Pedal Value Source α for the engine, *Dynamometer Control* for the *Drive*) and to acquire the measurement data (e.g. speed and torque signals of the shaft). All this is done by the physical input/output cards (i.e. Analogue/Digital and Digital/Analogue cards).

2.2 Real-time system

The real-time system consists of a standard PC hardware running the real-time operating system *QNX*² and the real-time software *RT-LAB*³. This *HiL Simulator* is connected with the *Control System* via an ether-net (UDP/IP) connection. To guarantee loss-less communication a watchdog is implemented in the simulation model.

2.3 PC Workstation

The *PC Workstation* is also a standard PC. On this PC the Modelica[®] simulation models are created with *Dymola*⁴. These models are not suitable for real-time yet.

In order to adjust them for real-time the *Dymola*[®] model has to be included in a *MATLAB*[®]/*Simulink*[®] model⁵ (this is done by using the *DymolaBlock*). The real-time software *RT-LAB* then automatically translates the resulting model with the *RealTimeWorkshop*^{®5}, transfers the C-code to the *HiL Simulator* via FTP and starts the compilation process. For another example of how to use *Dymola*[®] in connection with *RT-LAB* see [3]. Once that is finished the compiled model can be loaded and executed via the *RT-LAB* main control panel on the PC Workstation.

3 Simulation models

To demonstrate the synchronisation of a Modelica[®] model, we choose a standard 6-gear automatic

²QNX[®] Software Systems → www.qnx.com

³RT-LAB is a real-time software of Opal-RT → www.opal-rt.com

⁴Dymola[®] is a dynamic modelling software of Dynasim AB → www.dynasim.se

⁵MATLAB[®]/*Simulink*[®]/*RealTimeWorkshop*[®] is a simulation package of The Math Works, Inc. → www.mathworks.com

transmission drive line model of the Power Train Library [4] (see Fig. 2).

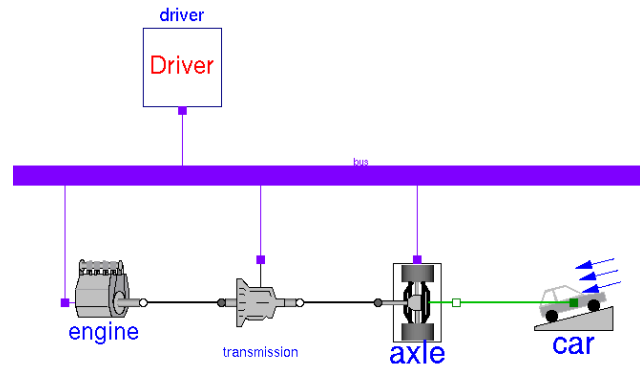


Figure 2: Graphical presentation of the drive line model

As driving cycle an excerpt of the New European Driving Cycle (NEDC) was used (The only reason for not using the whole NEDC was to speed up the simulation work, since for this work no additional information could be gained by using the complete NEDC.). Figure 3 shows the velocity in km/h over time of the NEDC excerpt when simulated offline with *Dymola*[®]. The standard Modelica[®] model has no input and output connectors so far. These connectors are needed when integrating the model into *Simulink*[®]. In a next step we have to define which signals the test-bench system needs and which signals the Modelica[®] model needs in order to function correctly as a unit. Also the exact mode of interaction of the simulation model and the test-bench system has to be defined.

3.1 Requirements for test-bench application

In our application we came across four main issues to clarify:

1. Which parts are simulated and which exist as “hardware”?
2. What control strategy is used?
3. How to synchronise the model and the test-bench?
4. What interface signals are needed depending on the control strategy?

3.1.1 Used hardware

The answer to the first question is quite simple. We want to simulate a drive line model in connection with

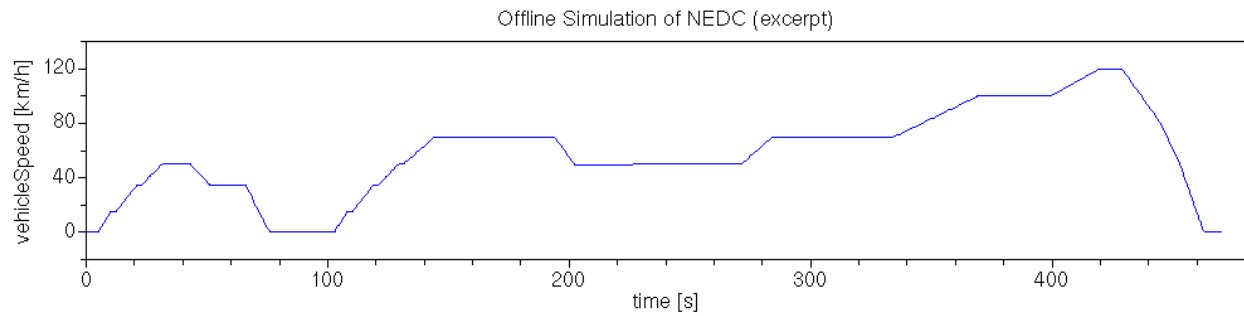


Figure 3: Excerpt of NEDC

engine test-bench system. Therefore our “hardware” device is the *engine*. So the parts which will be simulated in real-time on the *HiL simulator* will be:

- Driver (modified)
- *fake* Engine (modified)
- Transmission
- Axle (modified)
- Car

Some more details on the kind of modification applied will be given later in Section 3.2.

3.1.2 The “right” control strategy

The second question is bit harder to answer. Depending on the used test-bench system there might different control strategies available. A control strategy defines the kind of signals with which the test-bench system is controlled. In our case we have the choice of controlling the dynamometer with either a torque signal (M for moment of torque) or a speed signal (n). The engine is always controlled via the pedal value source (α). These control strategies are therefore called $M - \alpha$ - or $n - \alpha$ -control. Depending on the operation mode one is more preferable to the other.

In our case the drive line simulation model includes an automatic gear transmission with a torque converter. This hydraulic torque converter connects the engine outlet (pump) with the transmission inlet (turbine). A speed difference of the pump and the turbine of the converter yields a transfer torque and vice-versa. So whenever there is a torque on the load side (transmission) the inlet speed of the torque converter can be computed. So we can use the speed signal to control the test-bench system (the dynamometer to be more precise) at all times of simulation.

If we would use a manual shift gearbox a speed signal for controlling the dynamometer of the test-bench can not be obtained so easily for certain modes of operation. One of these modes being the shifting of the gear. When a gear is shifted the clutch (which is located between the engine and the transmission) is opened. When the clutch is opened it is difficult to determine the speed of the “free spinning” engine side of the clutch. To do this correctly some parameters such as the internal friction and the inertia of the engine have to be known in detail. Obtaining these parameters is not very easy (if not sometimes impossible) and comes with a lot of measuring effort. On the other hand the applied load torque of the engine side of the clutch during gear shift is known to be zero (this is true when neglecting the inertia of the clutch for now). So by using the $M - \alpha$ -control strategy whilst shifting the gear we can overcome the uncertainties of not knowing the correct control speed (as needed by $n - \alpha$ -control). Now the reader may ask why not using $M - \alpha$ -control during all states of operation?

One reason for this is the safety issue that it is always good to know and to check the speed set-point rather than handing over a torque set-point from which an acceleration or deceleration will result. A too big torque request and hence a too big acceleration might lead to speed too high for the test-bench system even before the system itself can react. Obviously things like this can be avoided by an appropriate design of the test-bench system. But in most cases this leads to a very complicated control structure. So in practice one uses the $M - \alpha$ control strategy for modes where $n - \alpha$ -control is not suitable. See [1] for more on this topic.

3.1.3 How to synchronise

Section 2.2 described in short the coupling between the test-bench system and the real-time computer on which the drive line model is running. We will give a

short example to get a better understanding of how the whole synchronisation process between the test-bench system and the drive line model works.

At first the the test-bench engine is started and put into idle mode (this can differ if someone is interested in measurement results of a “cold-start and run” cycle). In our example the engine is now running in idle speed and controlled by the control software of the test-bench system. In the meantime the drive line simulation model was loaded into the HiL-simulator (see again Fig. 2). For now both the test-bench system and the HiL-simulator are already communicating with each other via UDP. But until now they are just “listening” to each other. The next step would be to connect the simulation model to the test-bench system so that the drive line model controls the pedal value source α of the real engine and the set-point of the torque or the speed (depending on the control strategy) of the dynamometer. But before this connection can be made the difference of the control signals (i.e. α, n, M) between the measured values of the test-bench system and the calculated values of the simulation model may not exceed a certain boundary. To ensure this we use the Boolean signal `syncTBS`⁶ which when activated forces the drive line model to accept the engine torque and engine speed as input set-point for the torque converter (or clutch in case of a manual shift transmission). This way the drive line model is running at the same speed like the engine regardless of the measured input torque from the engine.

Once the engine and the drive line model are running at the same speed the drive line model can take over the control of the engine. This is done by a software switch of the test-bench system. After setting the `syncTBS` signal to “false” the *HiL simulator* controls now the engine via the pedal value source α and the speed signal n or the torque signal M , depending on the control strategy.

3.1.4 Interface signals

Up to now we should have identified all necessary interface signals. These depend also on the used control strategy. To keep a certain degree of freedom in terms of which control strategy is used, we choose to supply interface signals suitable for both, $M - \alpha$ - and $n - \alpha$ -control. We also need some extra signals to control the Dymola[®] model embedded in the real-time environment. These extra signals are the signal to start the driving cycle, the signal to switch between the two

control strategies and the signal to switch the drive line model into synchronisation mode.

It follows a list of the the inputs used (extra signals are marked with ‘*’).

Inputs to the simulation model:

- speed of the test-bench engine [*rpm*]
- torque of the test-bench engine [*Nm*]
- start of driving cycle [*boolean*]*
- $n - \alpha$ -control active [*boolean*]*
- synchronize model to test-bench [*boolean*]*

As outputs again we needed some additional signals to display the current state of the simulation model (again marked with ‘*’). These are optional and not required by the test-bench system.

Outputs to the test-bench system:

- drive line speed [*rpm*]
- drive line torque [*Nm*]
- pedal value source α [p.u.]
- vehicle speed [*km/h*]*
- cycle speed set-point [*km/h*]*
- selected gear*

3.2 Modification of simulation models for HiL

The previous section showed the interface signals necessary to couple the *HiL simulator*(drive line model) to the test-bench system. Now this section will give some more more details in which way the drive line model had to be adapted in order to be able to provide the interface signals or to react accordingly to them.

The `Driver`, the `Engine` and the `Axle` models had to be modified:

3.2.1 Modifications to the driver model

The Boolean signal `startCycle` had to be added in order to start the driving cycle at a given time. This means that the original `CombiTimeTable` block from the Modelica Standard Library (MSL 2.2.1) had to be replaced by the two blocks `Timer` and `CombiTable1Ds` (see Fig. 4).

⁶= `synchroniseTestBenchSystem`

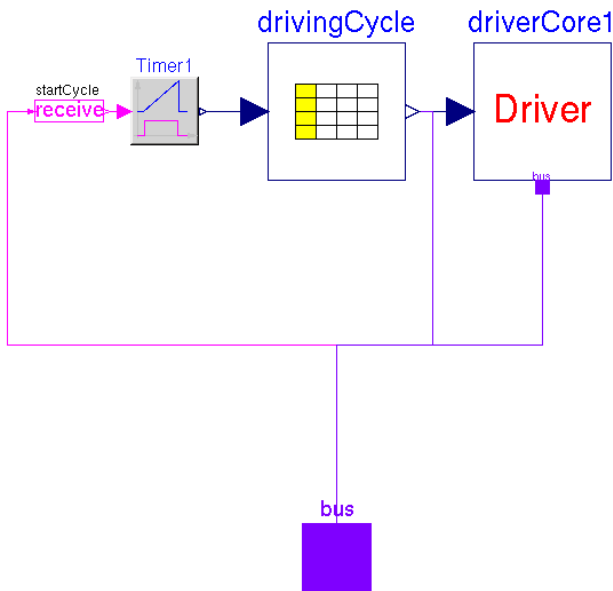


Figure 4: Picture of TriggerdDriver model

Finally the driving cycle table had to be adjusted as well because CombiTable1Ds only accepts monotonically increasing data vectors. The Timer block on the other hand accepts two different function values at one time instant.

3.2.2 Modifications to the engine model

Since the engine exists as hardware part within our HiL simulation we only need some kind of “fake” engine. This “fake” engine will be used as interface to the torque or speed signal coming from the test-bench system. We did not want to break with the drive line architecture of the Power Train Library. Therefore a FakeEngine model was created based on the PowerTrain.Engine frame (see Fig. 5).

The governor had to be removed since the test-bench system includes an idle-speed control of its own. Also the torque measurement of the drive line torque is done in this model.

The FakeEngine model includes the subcomponent TBSsyncEngineMalphaN instead of the original BaseEngine subcomponent (see Fig. 6).

In the model TBSsyncEngineMalphaN all the synchronisation and control strategy functions are implemented.

When the signal syncTBS is active both coupling clutches Malpha and Nalpha are closed making sure that the drive line synchronises to the test-bench system. When the synchronisation process is complete the syncTBS signal can be deactivated. Depending

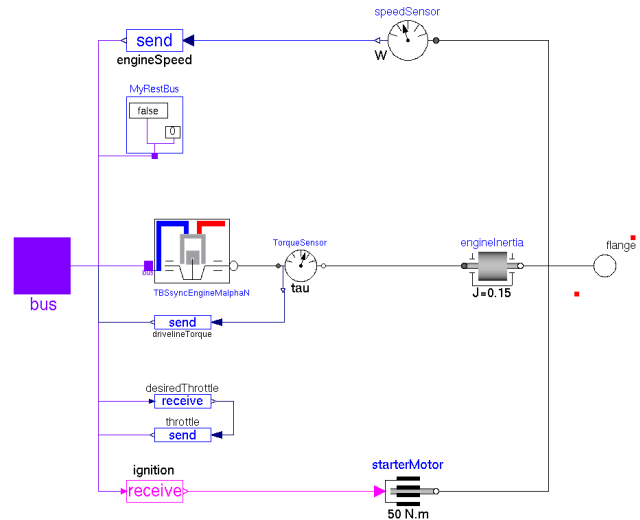


Figure 5: Picture of the FakeEngine model

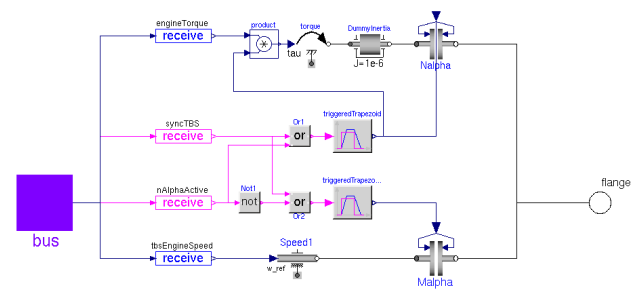


Figure 6: Picture of the TBSsyncEngineMalphaN model

on the state of the signal nAlphaActive either the Malpha or the Nalpha clutch stays closed after deactivating syncTBS. To avoid sudden changes (which in turn can cause instabilities) the closing and opening process of the clutches is done via triggered ramp signals.

The engineTorque signal is multiplied with zero as long as the $M - \alpha$ control strategy is activated. This is to avoid a constant speed-up of the left hand side of the then open $n - \alpha$ coupling clutch.

3.2.3 Modifications to the axle model

When simulating the drive line model with the CarResistance2 component of the Power Train Library included we noticed a phenomenon. At the beginning of the simulation the CarResistance2 model calculates a small erroneous torque which is applied at the axle flange. This in turn causes a small deceleration of the vehicle model (i.e. the car rolls back-

wards). This backward rolling behaviour causes the real-time HiL model to become unstable. A quick solution to avoid this was to activate the brakes as long as the driver is not starting to drive (see Fig. 7).

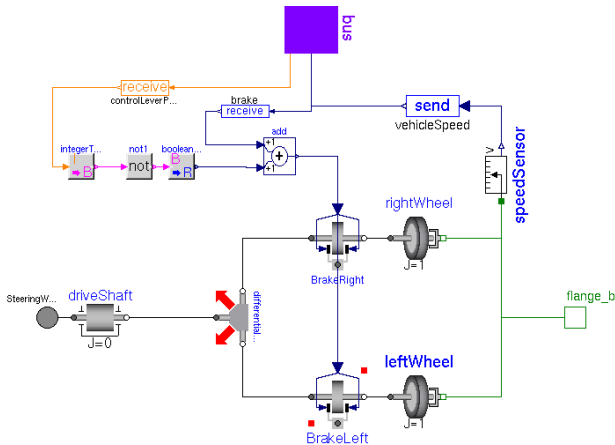


Figure 7: Picture of the ModifiedAxle model

4 HiL simulation results

Finally after all necessary interface signals have been defined and the modification to the drive line model has been done it is time to test the model with the test-bench system. In order to generate real-time capable code the Dymola[®] model is included in a Simulink[®] model as a wrapper using the *DymolaBlock*. The resulting Simulink[®] model is then arranged to fit the real-time structure of RT-LAB[®]. More on one how to do a real-time simulation using Dymola[®] in connection with RT-LAB[®] can be found in [5].

Figure 8 shows the velocity in km/h and the engine speed in rpm over time of the NEDC excerpt when simulated online. By online we mean the Dymola[®] model runs in real-time on the *HiL simulator* and communicates with the test-bench system via the UDP/IP interface. Since we are using a automatic gear transmission the $n - \alpha$ control strategy was used during the simulation (i.e. `nAlphaActive=true`).

4.1 Synchronisation

To demonstrate the synchronisation process a plot of roughly the first 30 seconds is displayed in Fig. 9. Prior coupling the *HiL simulator* and the test-bench system together we need the drive line speed of the simulation model and the engine speed of the test-bench to be the same. We can divide process in Fig. 9 into three phases.

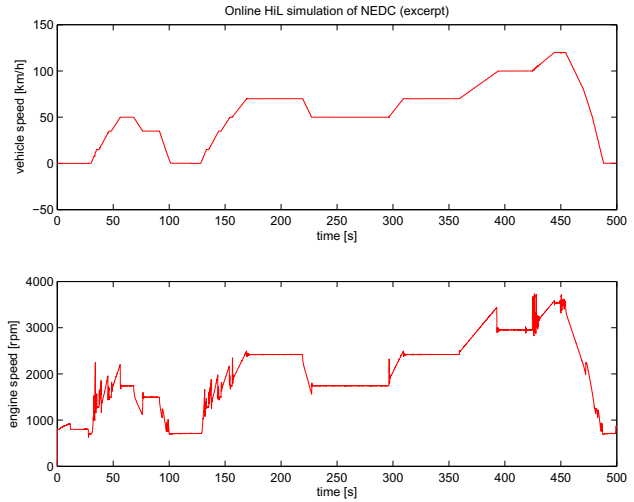


Figure 8: Online simulation of NEDC cycle (excerpt)

Phase 1: At the start the `syncTBS` signal is false. This means that the drive line model receives the torque signal of the test-bench engine. As mentioned before the test-bench engine is idling at the beginning. The torque is something near zero Nm . This small torque causes now the free-spinning torque converter (control lever position is still “Neutral”) to accelerate slightly (see *driveLineSpeed* in Fig. 9).

Phase 2: After about 10 seconds the `syncTBS` signal is activated. Now the drive line model is forced to the speed of the test-bench engine. Within this phase the test-bench engineer also activates a switch on the test-bench system to hand over the control to the *HiL simulator* (i.e. the coupling process is completed and n and α are now provided by the drive line model).

Phase 3: We can now deactivate the `syncTBS` signal once the two systems are coupled. Everything is now ready to start the driving cycle (i.e. `startCycle=true`). The synchronisation process is therefore complete.

5 Conclusions

In this paper we have demonstrated the process of synchronising a standard Modelica[®] model to a highly dynamic engine test-bench system. Different interface signals had to be added to the original simulation model so it can communicate with the test-bench system. This led to an adaptation of the simulation models

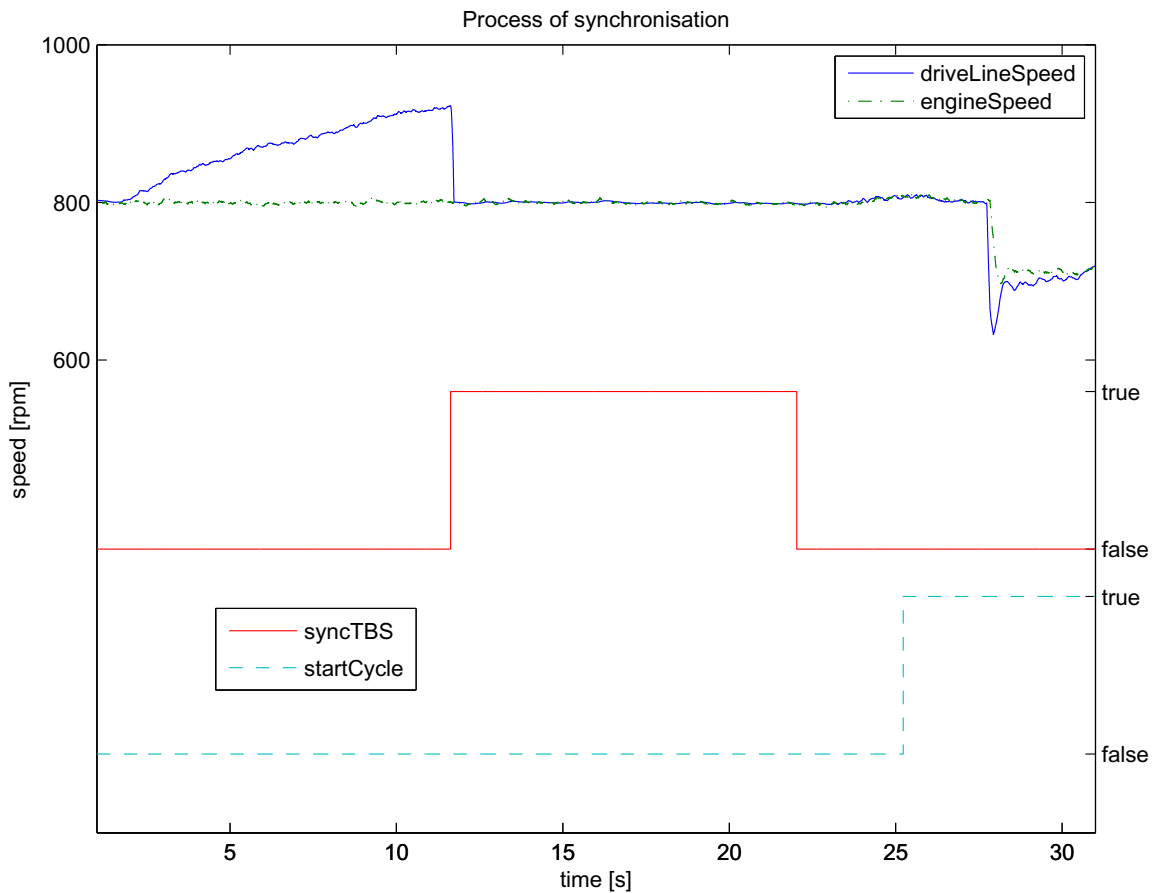


Figure 9: Synchronisation process in detail

which was presented in detail. In the end the simulation results of a working synchronisation process were shown.

The test-bench system can be used to simulate a variety of drive line models. Starting from conventional drive lines up to drive lines with a double-clutch transmission and even a hybrid electric vehicle (HEV) drive line. At all times the switching between the two different control modes (i.e. $n - \alpha$ and $M - \alpha$) is a very delicate issue especially when done “online”.

Our Department will continue to investigate the challenges of HiL simulation in connection with the engine-test bench. The focus lies hereby on the simulation of hybrid electric vehicles power trains. The test-bench system gives here the opportunity to gain measurement data of fuel consumption and exhaust-gas emissions for different HEV applications.

References

[1] D. Winkler, C. Gühmann, B. Barzantny, and M. Lindemann, “Model Based Calibration of

ECUs Using a Highly Dynamic HiL Test Bench System,” in *Design of Experiments (DoE) in Engine Development II* (K. Röpke, ed.), vol. 49 of *Haus der Technik Fachbuch*, (Berlin), pp. 268–277, Haus der Technik Essen, expert verlag, June 2005.

- [2] Modelica Association, “Modelica is a free modelling language.”
- [3] H. Elmqvist, S. Mattsson, H. Olsson, J. Andreasson, M. Otter, C. Schweiger, and D. Brück, “Real-time Simulation of Detailed Automotive Models,” in *Proceedings of the 3rd International Modelica Conference*, pp. 29 – 38, 2003.
- [4] M. Otter, C. Schweiger, and M. Dempsey, *PowerTrain Library 1.0*. German Aerospace Center (DLR), Oberpfaffenhofen, 1.0 ed., 2002.
- [5] Dynasim AB, *Dymola 5 - User Manual*. Lund, 2004.

Session 3c

Language, Tools and Algorithms 3

A Numeric Library for Use in Modelica Simulations with Lapack, SuperLU, Interpolation and MatrixIO

Anders Sandholm^{‡,‡,*} Peter Bunus^{‡,†} Peter Fritzson^{‡,‡}

[‡] PELAB - Programming Environment Lab
Dept. Computer Science, Linköping University
S-581 83 Linköping, Sweden

[‡] eHealth Institute
Dept. Health and Behavioural Sciences, University of Kalmar
Kalmar, Sweden

Abstract

This paper introduces a numerical Modelica library that provides access to some of the most well-known powerful libraries for numerical methods. Our approach has been to develop wrappers that allow Modelica users easy access as functions both from textual and graphical Modelica environments [9], [10]. This library also includes additional external functions with corresponding Modelica wrappers to interpolate data and to read/write matrix data from/to files.

Keywords: Matrix, Lapack, SuperLU, Matrix Market File Format, Harwell-Boeing Matrix Format, Interpolation

1 Introduction

One important area of research is developing and implementing fast numerical methods that can be used to simulate physical phenomena. Researchers who are working with simulation usually do not want to spend time and resources implementing, debugging, and maintaining new numerical libraries. Instead they want to use existing libraries that are recognized as stable and efficient.

Numerical methods can be divided into different areas such as: optimization, solution of ordinary and partial differential equations, mesh generation, numerical integration, solution of nonlinear equations, solution of

linear equations, eigenvalue problems, curve and surface fitting, interpolation, etc. Finite element methods is a well-known group of methods for solving PDE problems, which typically are rather computation intensive.

This paper introduces a new wrapper library called Numeric intended for Modelica users who want to use standard common numeric libraries as well as methods and routines for saving and loading matrixes to/from files.

1.1 Small Example of Using the Library

Assume that the user wants to calculate the eigenvalues for an N-by-N real nonsymmetric matrix stored in the Matrix Market file format. The first task would be to load the matrix file, here called matrix.mtx. This is done by using the functions **getMatrixSize** and **getMatrixFile** where the first one returns the size of the matrix and the other one returns the matrix data, both taking the file name as a string argument. Functions for loading and saving matrices in Matrix Market is located in package **Numeric.MatrixIO.MatrixMarket** along with other Matrix Market functions.

Below Modelica pseudo code is shown for loading the matrix.

```
Integer n = getMatrixSize("matrix.mtx");  
Real A[n,n];  
A=getMatrix("matrix.mtx");
```

More information about loading and saving data can be found in the MatrixIO section. For the calculation of eigenvalues Lapack [2] contains a function **dgeev**

*andsa@ida.liu.se

†petbu@ida.liu.se

‡petfr@ida.liu.se

that calculates the eigenvalues along with the left and right eigenvectors of a general matrix. The `dgeev` routine uses double precision but the Lapack library also contains a corresponding function for single precision calculations, named `sgeev`.

In the library outlined in this paper all Modelica wrapper functions for Lapack are stored in subpackages. The wrapper for the `dgeev` function is located in **Numeric.Lapack.SimpleDriver**, for further detail see the section dealing with the structure of the library.

Below Modelica pseudo code is shown that outlines the call to the `calcEigenValGeneralMatrix_dgeev` which uses the Lapack `dgeev` function for the calculations of the eigenvalues.

```
Real eigenvReal[size(A, 1)];
Real eigenvImag[size(A, 1)];
Real eigenVectors[n,n];
(eigenvReal, eigenvImag,
eigenVectors) =
calcEigenValGeneralMatrix_dgeev(A1);
```

The Modelica wrapper function `calcEigenValGeneralMatrix_dgeev` allows the user to specify more input data and receive more information from Lapack than is shown here, which is further outlined in the Lapack section.

2 Structure of the Numeric Library

The design of this library focuses on two major issues:

- It should be easy to locate libraries and functions
- The package should be easy to maintain with all the external library dependencies
- The package structure should allow easy addition of new external libraries and native Modelica functions

This library contains both functions that are implemented natively in Modelica and functions that act as wrappers to C and FORTRAN 77 functions [9],[1]. The top level structure of the Numeric library can be seen in Figure 1 with the subpackages Lapack, SuperLU, MatrixIO, and Interpolation

2.1 The Structure of the Numeric Package

The subpackages Lapack and SuperLU contain Modelica wrapper functions that call corresponding external functions in each external library. The MatrixIO subpackage is further divided into subpackages that implement different matrix file formats for saving and

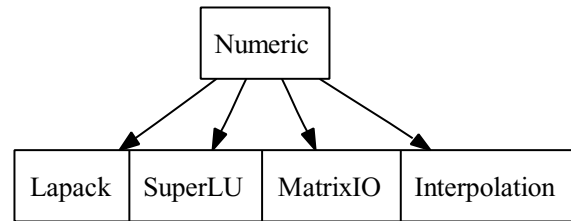


Figure 1: Structure of numeric package

loading matrix data. The Interpolation subpackage contains subpackages with methods both developed natively in Modelica code but also Modelica wrapper functions to interpolation library routines.

2.2 Structure of the Lapack Subpackage

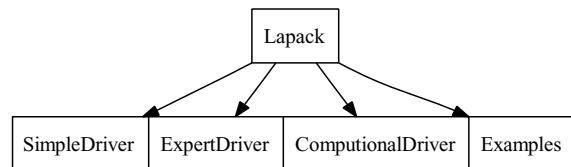


Figure 2: Structure of the Lapack subpackage and its subpackages

The Lapack subpackage can be seen in Figure 2. This package contains four subpackages, SimpleDriver, ExpertDriver, ComputationalDriver and Examples. For more information about SimpleDriver, ExpertDriver and ComputationalDriver see the Lapack section. In the Examples library different examples have been implemented which explain how the Lapack subpackage can be used in Modelica code. These examples are mostly constructed for users who know the Modelica language but are new to the Lapack library.

2.3 Structure of SuperLU package

The SuperLU subpackage has been divided into library subpackages, Driver, Computation, Utility as well as a section called Examples that has been added. The packaged structure can be view in Figure 3. For detailed information about the Driver, Computation and Utility subpackages see the SuperLU subpackage. In the Example subpackage to SuperLU different Modelica examples have been implemented that show how the SuperLU library can be used

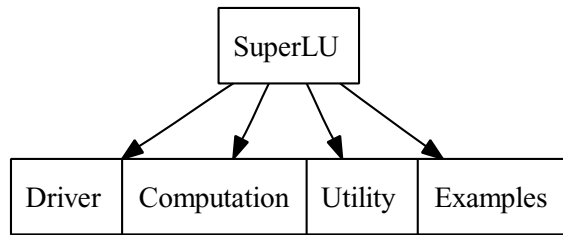


Figure 3: Structure of Lapack package with its sub-packages

2.4 Structure of MatrixIO package

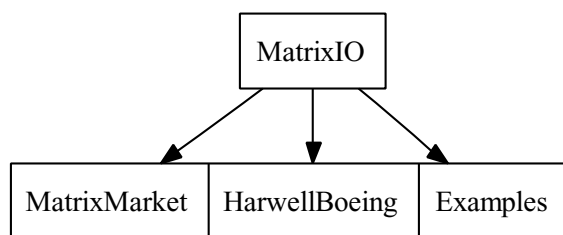


Figure 4: Structure of MatrixIO package

The MatrixIO packages implement support for different matrix file formats. Currently the Matrix-Market and the Harwell-Boeing subpackages are supported with functions for saving and loading dense and sparse matrix data. An overview of the MatrixIO package can be viewed in Figure 4. For more detailed information about the Matrix Market and the Harwell-Boeing see corresponding sections. Examples that show how matrix data can be loaded and saved are implemented in the Examples subpackage.

2.5 Structure of the Interpolation subpackage

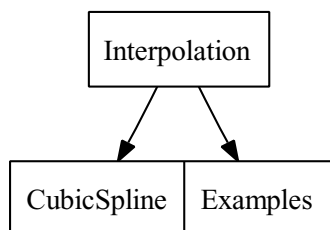


Figure 5: Structure of the Interpolation subpackage

The Interpolation subpackage is designed with the same idea as the other packages. Currently the sub-

package is divided into two subpackages, CubicSpline and Examples, see Figure 5. The CubicSpline subpackage contains both native Modelica function implementations and Modelica wrapper functions for use of external cubic spline function implemented in C code. The Examples subpackage contains easily understandable examples that show both how the Modelica implemented versions and the external version can be called from Modelica code.

For further details about cubic spline see the Interpolation subpackage section.

3 Library Design Issues

As already mentioned, the main idea is to create a Modelica package where different numerical methods, format handling functions, and solvers can be readily available for use from Modelica. Several design issues have been addressed on how to handle documentation from the external libraries and variable naming in the external functions. Without the library documentation the package would be hard to use and a user who is familiar with the corresponding non-Modelica package will be confused if the input/output variable has changed name in the Modelica wrapper function.

3.1 Naming Conventions

The Modelica Numeric library uses function and variable names from the original package as a postfix part of the name along with a more explanatory Java-style name comprising the beginning of the name. This will give new users more understanding of functions and variables, without reading the detailed documentation for each variable. Users who are familiar with the corresponding non-Modelica libraries will recognize functions and variables due to the postfix part of the name.

An example is the Modelica wrapper function `calcEigenValGeneralMatrix_dgeev` which is introduced in the Introduction part of this paper. The first part of the function name tells the user that it calculates the eigenvalues for a general matrix and the postfix part specifies that the `dgeev` function is used. The same naming convention is used for variables. The `dgeev` function has a variable named `JOBVL` that specifies if the left eigenvalues should be calculated or not. In the Modelica wrapper function this variable is named `calcLeftEigenV_JOBVL` which are a more self explanatory Java-style name along with the Lapack

variable name as a postfix part of the name.

3.2 Documentation

The issue about documentation has been addressed by including the external function documentation into the Modelica wrapper function documentation node. Below the first part of the documentation for the Modelica wrapper function `calcEigenValGeneralMatrix_dgeev` is shown.

First in the documentation comes a specification of the difference between the native function call and the Modelica wrapper function call. In the Modelica wrapper function the LDA, LDVL and LDVR variables are not needed, and therefore have been removed from the Modelica interface. After the library annotation the Fortran function declaration follows along with version and argument documentation. Further down comes the purpose and argument documentation. In this example only four arguments are shown.

annotation(Documentation(info="Lapack

```
#### Numerical Library annotation ####
Variables that has been excluded
in Numerical Library
```

```
LDA    = size(A,1);
LDVL   = size(A,1);
LDVR   = size(A,1);
```

```
#####
```

```
SUBROUTINE DGEEV( JOBVL, JOBVR, N, A,
LDA, WR, WI, VL, LDVL, VR,
LDVR, WORK, LWORK, INFO )
```

```
-- LAPACK driver routine (version 3.0)
Univ. of Tennessee, Univ.
of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National
Lab, and Rice University
December 8, 1999
```

```
.. Scalar Arguments ..
CHARACTER JOBVL, JOBVR
INTEGER   INFO, LDA, LDVL, LDVR,
LWORK, N
..
.. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ),
VL( LDVL, * ), VR( LDVR, * ),
WI( * ), WORK( * ), WR( * )
..
```

Purpose

=====

DGEEV computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies
 $A * v(j) = \lambda(j) * v(j)$
 where $\lambda(j)$ is its eigenvalue.
 The left eigenvector u(j) of A satisfies
 $u(j)**H * A = \lambda(j) * u(j)**H$
 where u(j)**H denotes the conjugate of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Arguments
 =====

JOBVL (input) CHARACTER*1
 = 'N': left eigenvectors of A are not computed;
 = 'V': left eigenvectors of A are computed.

JOBVR (input) CHARACTER*1
 = 'N': right eigenvectors of A are not computed;
 = 'V': right eigenvectors of A are computed.

N (input) INTEGER
 The order of the matrix A. N >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the N-by-N matrix A.
 On exit, A has been overwritten.

4 Lapack

Lapack is one of the most widely used libraries for solving many common numerical problems in linear algebra. The library includes routines for solving systems of simultaneous linear equations, finding least square solutions of overdetermined systems of equations, solving eigenvalue problems, and solving singular value problems [2]. The Modelica **Numeric.Lapack** sublibrary is divided into three different parts: Basic Routines, Advanced Routines and Computational Routines.

- Basic Routines solves a specified problem with a few options. Examples of functionality in basic routines are finding the eigenvalues of a matrix or solving a set of linear equations.
- Advanced Routines allows the user to control the calculations more by taking more options and returning more information than the simple driver routines. An example can be calculation of error bounds or normalizing matrices to improve accuracy.
- Computational Routines shall more be seen as routines designed to perform a specific task, such as a LU factorization or reduction of a real system matrix to tridiagonal form. Usually these functions are used to construct more advanced functions in the Basic and Advanced routines libraries. The routines are categorized in systems of linear equations, eigenvalue problems, orthogonal factorization, and singular value decomposition.

4.1 Example

An example of the simple driver routines is the `dgeev` function that calculates right and left eigenvalues and eigenvectors for an N-by-N real nonsymmetric matrix. This calculation can be described as finding the eigenvalues λ and corresponding eigenvectors $z \neq 0$ as equation (1) and (2) describe.

$$Az = \lambda z \quad (1)$$

$$A = A^T \text{ where } A \text{ is real} \quad (2)$$

When all eigenvalues and eigenvectors have been calculated equation (3) is solved.

$$A = Z\Lambda Z^T \quad (3)$$

Where Λ is a diagonal matrix whose diagonal elements are the eigenvalues, Z is an orthogonal matrix whose columns are the eigenvectors [3].

As described previously the Modelica wrapper function for `dgeev` is called `calcEigenValGeneralMatrix_dgeev` and is shown below, where the documentation part has been removed in this example.

```
function calcEigenValGeneralMatrix_dgeev
input Real A[:, size(A, 1)];
input String calcLeftEigenV_JOBVL = "N"
```

```
"Left eigenvectors of A
are not computed";
input String calcRighEigenV_JOBVR = "V"
"Right eigenvectors of A
are computed";
output Real eigenReal_WR[size(A, 1)]
"Real part of eigenvalues";
output Real eigenImag_WI[size(A, 1)]
"Imaginary part of eigenvalues";
output Real leftEigenVectors_VL
[size(A, 1), size(A, 1)]
"Left Eigenvectors";
output Real reightEigenVectors_VR
[size(A,1), size(A,1)]
"Right Eigenvectors";
output Integer INFO
"=0 successful computation";

protected
Integer N=size(A, 1)
"The order of the matrix";
Integer LWORK=10*N
"MAX size if JOBVL = V or
JOBVR = V LWORK >= 4*N";
Real WORK[LWORK];
```

```
external "Fortran 77" dgeev(
calcLeftEigenV_JOBVL, calcRighEigenV_JOBVR,
N, A, N, eigenReal_WR, eigenImag_WI,
leftEigenVectors_VL, N,
reightEigenVectors_VR, N,
WORK, LWORK, INFO)
annotation (Library="lapack");

end calcEigenValGeneralMatrix_dgeev;
```

The first argument is the Matrix A which the eigenvalues and eigenvectors are to be calculated for. The following two arguments, `calcLeftEigenV_JOBVL` and `calcRighEigenV_JOBVR`, determine if the right or/and left eigenvalues/eigenvectors are to be calculated. In the default setting only the right eigenvalues are calculated.

In the output section the eigenvalues variable comes first then the left and right eigenvectors and last an information flag that tells if the calculation could be performed.

Variables that don't add to the functionality of the Modelica wrapper function but are needed for the Lapack implementation have been placed in the protected section. For the function outlined above the working variables `LWORK` and `WORK` have been placed here, along with the variable `N` that specifies the order of the matrix.

5 SuperLU

For solving large, sparse, nonsymmetric systems of linear equations the SuperLU library is commonly used [11]. The SuperLU library is available either in C or in Fortran code. Here our Modelica implementation uses the Fortran interface for maximum performance. The SuperLU library starts by performing an LU decomposition [15] with partial pivoting and triangular systems solved through forward and backward substitution.

The LU decomposition can handle non-square matrices, but it is only for square matrices the triangular solver is used. For improving backward stability interactive refinement subroutines are used. The library also contains routines provided to equilibrate the system, estimate the condition number, calculate the relative backward error and estimate error bounds for re-fined solutions.

The SuperLU subpackage is divided into three parts: Driver, Computation, and Utility. In the Driver subpackage functions for solving systems of linear equation are provided. In the Computation subpackage specified computational routines are provided instead of a complete driver as in the Driver package. Using this pack-age the user can develop a new computation driver in the Modelica environments. The last package is the Utility subpackage that supplies the user with routines for creating and destroy SuperLU matrices.

5.1 Examples

Take the function `dgstrf` as an example in the **Numeric.SuperLU.Computational** sublibrary. It performs a LU factorization of a general sparse m -by- n matrix, A , using partial pivoting with row interchanges. Factorization has the form of equation (4)

$$Pr * A = L * U \quad (4)$$

where Pr is a row permutation matrix, L is lower triangular with unit diagonal elements and U is upper triangular. The documentation for the function `call dgstrf` can be found in the SuperLU documentation [11], [12].

6 Interpolation

In many engineering and science areas data is gathered either from sampling real observations or by sim-

ulations where data is created at certain time intervals. Interpolation is a technique which uses the sequence of known values to estimate the value of an unknown point [14]. Given a sequence of known sample points, x_k , and the corresponding values, y_k , the interpolation tries to fit a function, f , that which when given an value in x_k , returns the corresponding value in y_k , shown in equation (5).

$$f(x_k) = y_k \quad \text{where } k = 1, 2, 3, \dots, n \quad (5)$$

This method of trying to find f is commonly known as curve fitting and the function f is then called the interpolant.

When calculating a value for an unknown data point, α , a control has to be made that α , lies inside the sequence of known values, se equation (6).

$$\min(x_k) \leq \alpha \leq \max(x_k) \quad (6)$$

No interpolation can be performed if the data point is lying outside the sequence x_k . To calculate the interpolated value the point is inserted in the interpolation function, $f(\alpha)$ and the function is evaluated. In the Numeric package a cubic spline interpolation scheme has been implemented both in native Modelica code and by using external library. The external library can be reached through a Modelica function that acts as a wrapper.

6.1 Cubic Spline

A cubic spline is a function that is defined as a piecewise third-order polynomial function which passes through a set of points. To create a solvable system a boundary condition is commonly placed on the second derivate of each polynomial end point. If the boundary condition is that the second derivative is equal to zero the spline is commonly called a natural cubic spline which gives a tridiagonal system that easily can be solved. Different boundary conditions can be used for creating other spline interpolation scheme [4] [7]. Suppose that the function f is to be interpolated, given by the data (x_i, f_i) , $i = 0, \dots, N$ where $f_i = f(z_i)$ and z_i form an order of sequence such as $a = x_0 < x_1 < \dots < x_N = b$. From this the cubic interpolation function $S \in C^2[a, b]$ can be described for each interval $[x_i, x_{i+1}]$ as equations (7) and (8) along with the fact that the polynomials are smoothly adjusted (10) and that the interpolation condition (13) is satisfied [13].

$$S(x) \equiv S_i(x) \tag{7}$$

$$S_i(x) = a_{i,0} + a_{i,1}(x - x_i) + a_{i,2}(x - x_i)^2 + a_{i,3}(x - x_i)^3 \tag{8}$$

$$\text{for } x \in [x_i, x_{i+1}], i = 0, \dots, N - 1 \tag{9}$$

$$S_{i-1}^r(x_i - 0) = S_i^r(x_i + 0) \tag{10}$$

$$i = 1, \dots, N - 1 \tag{11}$$

$$r = 0, 1, 2 \tag{12}$$

$$S_{i-1}^r(x_i - 0) = S_i^r(x_i + 0) \tag{13}$$

$$i = 1, \dots, N - 1 \tag{14}$$

$$r = 0, 1, 2 \tag{15}$$

7 MatrixIO

While working with numerical applications the ability to save and load matrix data in an efficient file format is often needed. Here we decided not to create our own file format but rather to build in support for the most common formats. This gives the user the ability to work with existing data and to easier exchange data with other users. We have chosen to support the Matrix Market [6] [5] and Harwell-Boeing [8] formats.

7.1 Harwell-Boeing Matrix Format

The Harwell-Boeing format is today one of the most popular text-file exchange formats for sparse matrixes. The file format starts with a header block where the first line contains the title and an identifier. The second line contain the number of lines for each of the data blocks and the total number of lines in the file, excluding the header. The third line contains a three character string denoting the matrix type and the number of rows and column entries. The fourth line contains the variable Fortran format for the following data block and the fifth line is only present if there is a right hand side of the matrix. The data is stored in an 80-column, fixed length format where each matrix begins with a multiple line header block, which is followed by two, three or four data blocks.

Using this information the correct storage can be allocated before the actual matrix data is accessed [8].

7.2 Matrix Market Format

The Matrix Market format provides a powerful and simple file format for storing and exchanging matrix data. The format is based on an ASCII file format that is based on a collection of affiliated formats which share certain design elements. So far, we have focused on supplying routines for accessing two of these design elements, general sparse matrices and general dense matrices.

In the general sparse matrices version only the non-zero entries are stored, and for each value the corresponding matrix coordinates is stored. For general dense matrices the array format is the most efficient, and the data is provided in a column-oriented order.

In both of the formats an arithmetic field is defined that specifies the matrix entries, i.e, real, complex, integer, pattern. The format also specifies the symmetry structure such as general, symmetric, skew-symmetric or Hermitian [6].

7.3 Examples

The easiest way to read a Matrix Market file is using the functions **getMatrixSize** and **getMatrixFile**.

getMatrixSize takes the file name as argument and reads the size of the matrix so that the a matrix with the correct size can be allocated. The function **getMatrixFile** also takes the filename as argument and reads the matrix data and store it in the corresponding data structure. A Modelica pseudo code example can be seen below where a matrix is loaded from a file called matrix.mtx.

```
Integer n = getMatrixSize("matrix.mtx");
Real A[n,n];
A=getMatrix("matrix.mtx");
```

During the process of reading the file and storing it in the MatrixMarket format messages are provided through the **ModelicaMessage()** function.

Acknowledgements

This work was supported by Kalmar eHälsöinstitut, by Vinnova in the GRIDModelica project, and SSF in the VISIMOD project.

References

- [1] The Modelica Language specification version 2.2. The Modelica Association, March 2005. <http://www.modelica.org>.

- [2] E Anderson, Z Bai, C Bischof, J Demmel, J Dongarra, J Du Croz, A Greenbaum, S Hammarling, A McKenney, S Ostrouchov, and D Sorensen. *LAPACK Users' Guide*. 1995.
- [3] George B. Arfken, Hans J. Weber, and Hans-Jurgen Weber. *Mathematical Methods for Physicists*. Academic Press, 1985.
- [4] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1995.
- [5] Ronald F. Boisvert, editor. *The Matrix Market: A Web Resource for Test Matrix Collections*, London, 1997. Chapman & Hall.
- [6] Ronald F. Boisvert, Roldan Pozo, and Karin Remington. The matrix market exchange formats: Initial design. Technical Report NISTIR 5935, National Institute of Standards and Technology, Gaithersburg, MD, USA, December 1996.
- [7] Carl De Boor. *A Practical Guide to Splines*. Springer, 2002.
- [8] Iain Duff, Roger G. Grimes, and John G. Lewis. Users' guide for the harwell-boeing sparse matrix collection (Release I). Technical Report TR/PA/92/86, CERFACS, October 1992.
- [9] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004. ISBN 0-471-471631.
- [10] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldami, and David Broman. The Openmodelica modeling, simulation, and software development environment. *Simulation News Europe*, 44/45, 2005.
- [11] John Gilbert James W. Demmel and Xiaoye S. Li. Superlu users' guide. Technical Report UCB/CSD-97-944, EECS Department, University of California, Berkeley, 1997.
- [12] John R. Gilbert Xiaoye S. Li James W. Demmel, Stanley C. Eisenstat and Joseph W.H. Liu. A supernodal approach to sparse partial pivoting. Technical Report UCB/CSD-95-883, EECS Department, University of California, Berkeley, 1995.
- [13] Boris I Kvasov. *Methods of Shape-Preserving Spline Approximation*. World Scientific, 2000.
- [14] Erik Meijering. Chronology of interpolation: From ancient astronomy to modern signal and image processing. volume 90, pages 319–342. IEEE, March 2002.
- [15] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in FORTRAN (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.

Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA

Rüdiger Franke
ABB AG, Power Technology Systems
Kallstadter Str. 1
68309 Mannheim, Germany

Jens Doppelhamer
ABB Corporate Research
Wallstadter Str. 59
68526 Ladenburg, Germany

Abstract

The Modelica technology and the increasing availability of model libraries allow an efficient modeling of complex dynamic processes. Having a good process model at hand one might want to apply the model online to improve the operation of the real process. These online applications range from the generation of high-level information like performance indices from process measurements over the estimation of unmeasured quantities in a so called soft sensor up to model based control and online optimization.

This paper discusses the online application of Modelica models in an industrial control system. The models are developed and tested using a standard Modelica tool. Afterwards they are imported into the control system. Here the model variables can be associated with process signals. This way a model can be initialized with current process values. A numerical solver performs simulation, estimation or optimization activities. Solution results can either be used for diagnostics or they can be fed back to the process as manipulated variables.

The Dynamic Optimization system extension has been developed for the Industrial IT System 800xA. Exploiting Aspect Object technology, the required functionality for model-based applications can be integrated seamlessly with the control system. Model based applications can be set up in a modularly structured way.

The Dynamic Optimization system extension has been used to deploy different model-based applications. A Nonlinear Model-based Predictive Controller (NMPC) for the start-up of steam power plants is discussed as an example. The overall NMPC application consists of several model-based activities, including preprocessing of process values, estimation of model states, prediction of optimal operations, and post-processing of optimization results. A scheduler periodically triggers

these activities online.

Keywords: Modelica, 800xA, Industrial IT, control system, online optimization, NMPC

1 Introduction

The Modelica technology clearly separates between model specification and model solution. This way not only existing models can be used with different tools, but also different kinds model based activities can be performed for one and the same model. Such activities include, besides the solution of initial-value simulation problems, also the estimation of model parameters, the optimization of the design of a modeled process and model-based control.

Also the increasing availability of libraries for fluid processes is making Modelica more and more suitable for process applications, see also [5, 8, 4].

Additional things that have to be treated in a real model based application include the signal exchange with the process, concepts for security and redundancy as well as real-time scheduling of model based activities and the operator user interface.

This paper discusses the integration of dynamic optimization with the Industrial IT System 800xA by ABB, allowing a rapid online deployment of model based applications once appropriate models and model based activities have been set up offline.

2 System 800xA overview

The architectural framework for the Industrial IT System 800xA is built upon ABB's Aspect Object technology [1]. Aspect Objects relate plant data and functions – the aspects, to specific plant assets – the objects. Aspect objects represent real objects, such as process units, devices and controllers. Aspects are informational items, such as I/O definitions, engineering

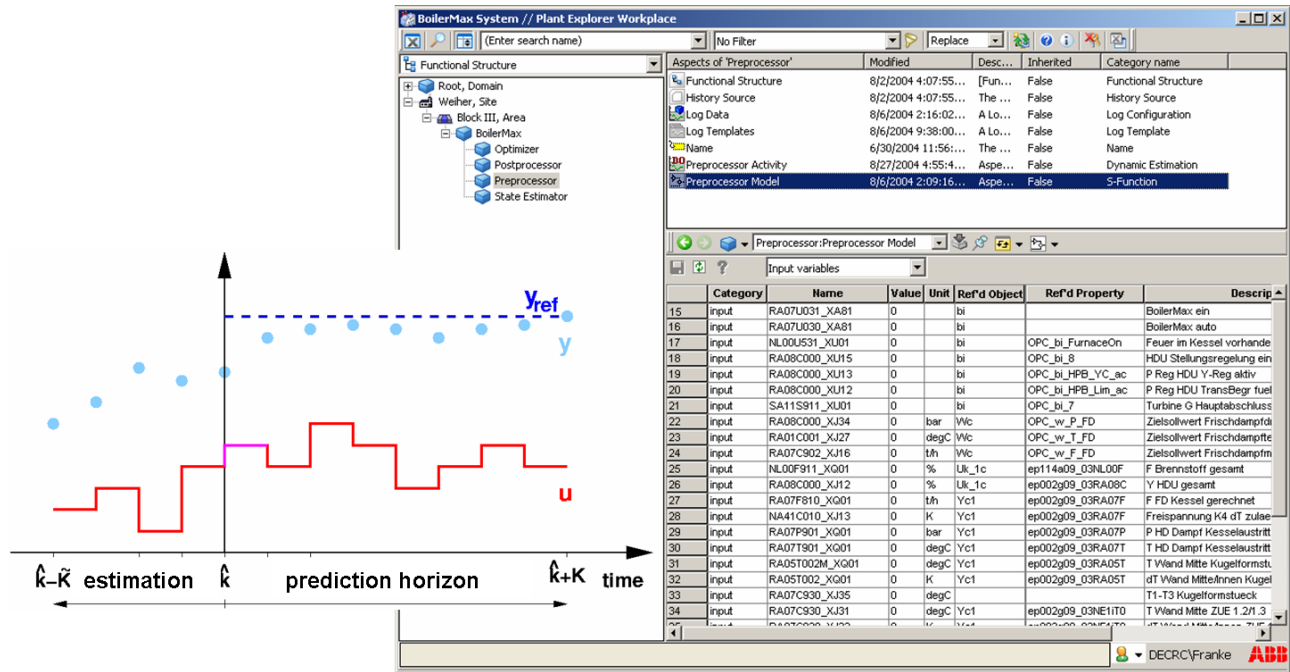


Figure 1: Plant Explorer Workplace showing the Functional Structure of an NMPC.

drawings, process graphics, reports and trends that are assigned to the objects in the system.

Aspect Objects are organized in hierarchical structures that represent different views of the plant. One object may be placed multiple times in different structures. Examples for different types of structures are:

Functional Structure: Shows the plant from the process point of view.

Location Structure: Shows the physical layout of what equipment is located where in the plant.

Control Structure: Shows the control network in terms of networks, nodes, fieldbuses, and stations.

The idea of placing the same object in multiple structures is based on the IEC standard 1346 [9, 2]. A controller is a typical example: the real controller is represented by an Aspect Object. This object is placed in the control structure showing the logical arrangement of the controller in the control system, in the location structure showing the actual location, and in the functional structure showing the function of the controller for the operation of the process.

The Plant Explorer Workplace is the main tool used to create, delete, and organize Aspect Objects and aspects. It is based on a structural hierarchy, similar to Windows Explorer, as demonstrated in Figure 1. The object hierarchy is visible on the left hand side of the

window. The upper right pane shows the aspects of an object and the lower right pane views a selected aspect.

3 Integration of model based control

3.1 Example for a complex model-based control application

Figure 1 shows how the functional structure is set up for an Nonlinear Model-based Predictive Controller (NMPC) using Aspect Object technology and the Dynamic Optimization system extension discussed in this section. Different Aspect Objects represent the major processing steps of the NMPC algorithm.

1. The Preprocessor reads current measurements, validates the data, and generates a guess for the model state. Furthermore a short term history is assembled.
2. The State Estimator uses the short term history and estimates the initial model state.
3. The Optimizer predicts the optimal control into the future, starting from the estimated initial state.
4. The Postprocessor checks optimization results and communicates set points to the underlying control system.

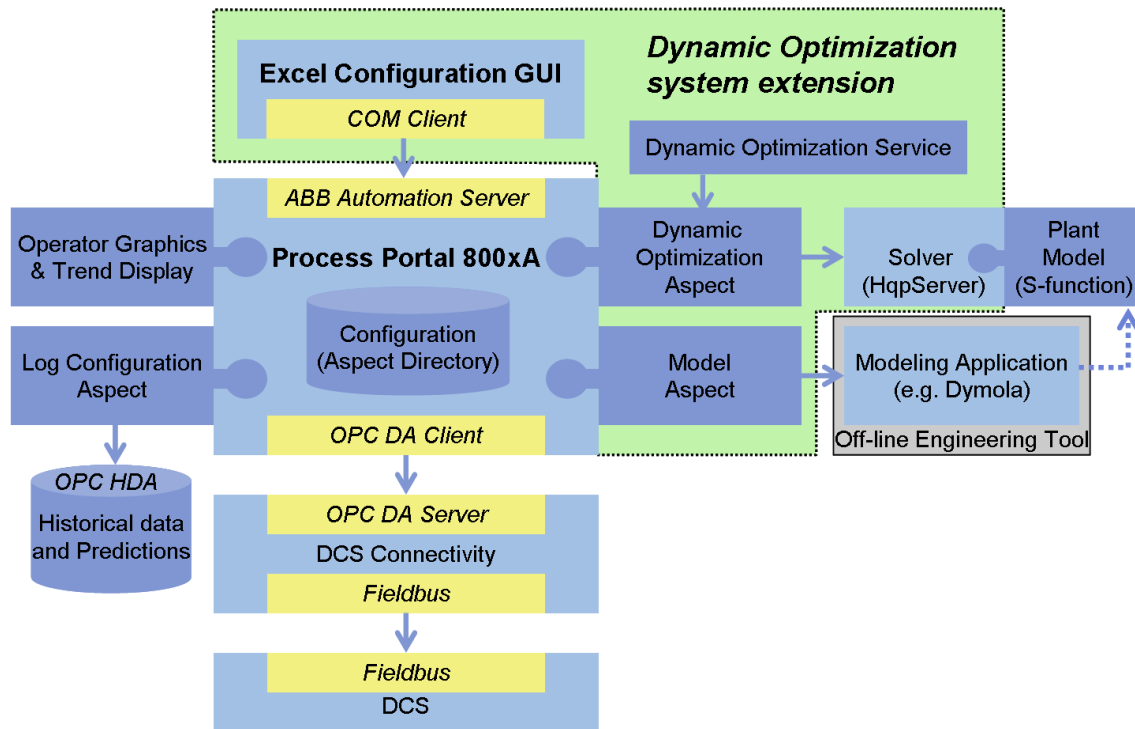


Figure 2: Software architecture of the Dynamic Optimization system extension.

An additional scheduler activity periodically triggers the other activities and supervises their successful completion.

The state estimator and the optimizer are based on the same plant model. This model is built efficiently on available model libraries [5]. Moreover, specific preprocessor and the postprocessor models are formulated as computational algorithms in Modelica. The scheduler model is formulated as a state graph [12].

Based on the models, the activities are formulated as estimation (State Estimator), optimization (Optimizer) or initial-value simulation (Preprocessor, Postprocessor, Scheduler). The Dynamic Optimization system extension provides the required aspects.

3.2 Dynamic Optimization system extension

Figure 2 shows the Dynamic Optimization system extension in the context of System 800xA. The framework underlying the Industrial IT System 800xA contains a scalable client-server object oriented database as one of its key components, seen as Aspect Directory in Figure 2. This database is generally used to store configuration data.

The System 800xA provides multiple predefined aspects that cover the basic functionality of a control system, such as control connection, process graphics and

history logs of process values. Additional functionality is added through system extensions.

The Dynamic Optimization system extension provides two new aspects: the Model aspect and the Dynamic Optimization solver aspect. The new aspects allow the seamless integration of model-based applications. Moreover a configuration GUI is provided as an add-in for Microsoft Excel, allowing the efficient engineering of model based activities. Last but not least the Dynamic Optimization service manages the instantiation of solver activities in online applications.

The model integration exploits the principle of Modelica to clearly separate model specification and model analysis [10]. For the applications conducted so far, the tool Dymola [3] is used for model editing and translation. The implementation of the executable model is treated as a Simulink S-function [13]. The used solver HQP [6] allows the treatment of different model based activities, including initial value simulation, estimation of model parameters and initial states as well as nonlinear optimal control with constraints on model inputs and outputs.

3.3 The Model Aspect

The model aspect of an aspect object provides applications with the necessary information to apply a model

based activity in the context of the aspects object. A model aspect thus augments the aspect object representation of a real-world object with model meta-data. The responsibilities of the model aspect are:

1. Persistent Storage of the model meta-data
2. Exposing a convenient API for the programmatic retrieval and manipulation of the model meta-data and
3. Providing a user interface to allow viewing and manipulating the model meta-data.

The Model aspect does not provide any functionality nor does it deal with implementation details. Instead it references an external implementation. In this way available modeling tools can be applied, such as Dymola, and expensive re-implementation is avoided.

3.3.1 Persistent Storage of the Model Meta-Data

The responsibility of actually storing the model meta data delegated by the model aspect to the Aspect Directory, ensuring qualities like security, redundancy and scalability and providing functionality like backup/restore and import/export of data. The stored data includes:

- Declaration of model variables in categories (Parameter, Input, Output, State, Generic),
- Values for model variables, e.g. for parameters,
- References to process signals, e.g. for inputs and outputs,
- Structural information for hierarchical sub-model structure,
- Reference to the implementation of the model.

3.3.2 APIs for Retrieval and Manipulation of Model Meta-Data

The 800xA framework also defines a generic means of exposing the data of aspect objects: Aspects can make their data available as a set of named properties with values of simple types (String, Real, Boolean, etc). Through these framework defined, generic interfaces to aspect properties, the model data can be made available to generic applications, i.e. non model-based ones. These generic interfaces were specially designed to ease access from many programming environments and languages. As an example, a tool providing import and export of aspect data to and from Excel could

use these generic interfaces. Another key component of the Industrial IT System 800xA can make data exposed as aspect properties available for clients using the widely recognized OPC standard for data access. For the convenience of model based applications, the model meta-data is also made available in a more structured way, using collections of complex types for model variables, their connection to process signals and other model meta-data. These API can be seen as a facade of the underlying, lower-level data structure exposed via the generic interfaces described above. Support for resolving references to process variables especially suited for modeling applications is added on this layer.

3.3.3 User Interface Integration

The framework underlying the Industrial IT System 800xA strongly supports user interface integration of the constituent applications. A consistent look and feel, support for services like drag-and-drop or copy-and-paste and the seamless integration of an applications user interface into workplaces like the Plant Explorer Tool can be easily achieved based on that support as well as role based customization and security of a workplace, i.e. the ability to adapt and restrict the applications user interfaces depending on the role of the current user of the system. Based on this framework features, and analog to the two levels model meta-data API described above, the model aspect provides three views of the model meta-data:

The first one reflects the lower-level data structure as a set of aspect object properties that can be viewed and individually manipulated (if sufficient permission is granted). This generic UI component is not specific to modeling application; it actually ships with the Industrial IT System 800xA Core and is reused by the model aspect to provide users with a well known view of the underlying data.

The second view specially presents information about the model variables in an Excel-like grid. Model variables can be sorted and filtered by their category (input, output, parameter, state or generic) and associated with process variables by drag-and-drop of aspect objects, e.g. from a tree view in a Plant Explorer, and selection of control connection aspects and properties from combo boxes in the grid. Features like undo functionality, sorting and Excel-like auto fill of this variable table is provided by the underlying, 3rd party, grid implementation.

Last but not least the third view embeds an Internet Explorer control that can be configured with an URL.

This view can e.g. be used to launch the modeling application Dymola to view the Modelica model graphically.

3.3.4 Mathematical view on a model

Mathematically, a model has the form of a hybrid differential algebraic equation system (hybrid DAE)

$$\mathbf{0} = \mathbf{F}[\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{m}(t), \mathbf{u}(t), \mathbf{z}(t), \mathbf{y}(t), \mathbf{p}, t], \quad (1)$$

$$\mathbf{F} : \mathbf{R}^{n_x} \times \mathbf{R}^{n_x} \times \mathbf{R}^{n_m} \times \mathbf{R}^{n_u} \times \mathbf{R}^{n_z} \times \mathbf{R}^{n_y} \\ \times \mathbf{R}^{n_p} \times \mathbf{R}^1 \mapsto \mathbf{R}^{n_x},$$

$$\mathbf{m}(t) := \mathbf{G}[\mathbf{x}(t), \mathbf{m}(t), \mathbf{u}(t), \mathbf{z}(t), \mathbf{y}(t), \mathbf{p}, t], \quad (2)$$

$$\mathbf{G} : \mathbf{R}^{n_x} \times \mathbf{R}^{n_m} \times \mathbf{R}^{n_u} \times \mathbf{R}^{n_z} \times \mathbf{R}^{n_y} \\ \times \mathbf{R}^{n_p} \times \mathbf{R}^1 \mapsto \mathbf{R}^{n_m}.$$

Here \mathbf{x} denote continuous-time states, \mathbf{m} are discrete modes, \mathbf{u} and \mathbf{z} are controlled and not-controlled inputs, respectively, \mathbf{y} are outputs and \mathbf{p} are model parameters. Discrete modes are variables that change their values only at discrete time instants, so called event instants t_e , see [10].

3.4 The Dynamic Optimization solver Aspect

A model can be applied to perform one or more model-based activities. A second aspect, the Dynamic Optimization aspect has been developed to interface a numerical solver, hold the solver configuration, and to exchange data between the solver and the control system. The exchanged data includes: configuration data, current process values (like sensor values and controller set-points), and history logs. Predictions are written back to the control system as history logs with future time stamps. Each aspect is working with its own instance of the numerical solver, allowing multiple model-based activities to run at the same time.

The integrated solver HQP is primarily intended for structured, large-scale nonlinear optimization [6]. It implements a Sequential Quadratic Programming algorithm that treats nonlinear optimization problems with a sequence of linear-quadratic sub-problems. The sub-problems are formed internally by simulating the model and by analyzing sensitivities. They are solved with an interior point method that is especially suited for a high number of inequality constraints, e.g. resulting from the discretization of path constraints. See [7] and [6] for more details about the solver.

Based on the system model (1),(2), several model-based activities can be formulated and solved numerically over a time horizon $[t_0, t_f]$. The treated model based activities include

- Initial value simulation for specified initial states $\mathbf{x}(t_0)$ and model inputs,
- Estimation of model parameters and initial states,
- Nonlinear optimal control with constraints on model inputs and outputs,
- Steady-state simulation, estimation and optimization at one time instant.

An initial-value simulation covers hybrid DAEs (1),(2). However, optimization and estimation problems can currently only be solved for a simplified hybrid DAE \mathbf{F} , \mathbf{G}' of the form:

$$\mathbf{m}(t) := \mathbf{G}'[\mathbf{m}(t), \mathbf{z}(t), t], \quad (3)$$

$$\mathbf{G}' : \mathbf{R}^{n_m} \times \mathbf{R}^{n_z} \times \mathbf{R}^1 \mapsto \mathbf{R}^{n_m},$$

where discrete modes do not depend on states or optimized variables.

3.4.1 Simulation Problem

The model behavior is completely determined by the system equations \mathbf{F} and \mathbf{G} , if initial states $\mathbf{x}_0 = \mathbf{x}(t_0)$, external inputs $\mathbf{u}(t), \mathbf{z}(t), t \in [t_0, t_f]$, and parameters \mathbf{p} are given. The outputs $\mathbf{y}(t), t \in [t_0, t_f]$ can then be obtained by solving the system of differential equations using initial-value simulation.

However, often some of the required information is not explicitly known, but can be obtained by minimizing a cost function. In many of those cases, a feasible solution can be further specified by constraining model variables. Optimization is a universal tool for treating those inverse problems.

3.4.2 Estimation Problem

An example for an inverse problem is the estimation of unknown parameters \mathbf{p} and/or initial states \mathbf{x}_0 based on measured inputs and outputs. The estimation problem can be solved by minimizing a least squares criterion

$$\sum_{i=1}^{n_{\bar{\mathbf{y}}}} \|\mathbf{y}(t_i) - \bar{\mathbf{y}}(t_i)\|^2 \rightarrow \min_{\mathbf{x}_0, \mathbf{p}} \quad (4)$$

for the set of measurement data $\{\bar{\mathbf{y}}(t_i), t_i \in [t_0, t_f], i = 1, \dots, n_{\bar{\mathbf{y}}}\}$.

3.4.3 Optimization Problem

The control inputs $\mathbf{u}(t), t \in [t_0, t_f]$ or the initial states \mathbf{x}_0 might be free to be chosen so that a criterion

$$F_0[t_f, \mathbf{x}(t_f)] + \int_{t_0}^{t_f} f_0[t, \mathbf{x}(t), \mathbf{u}(t)] dt \rightarrow \min_{\mathbf{x}_0, \mathbf{u}(t)}, \quad (5)$$

$$F_0 : \mathbf{R} \times \mathbf{R}^{n_x} \mapsto \mathbf{R},$$

$$f_0 : \mathbf{R} \times \mathbf{R}^{n_x} \times \mathbf{R}^{n_u} \mapsto \mathbf{R}.$$

is minimized subject to constraints on model inputs $\mathbf{u}_{\min}(t) \leq \mathbf{u}(t) \leq \mathbf{u}_{\max}(t)$ and outputs $\mathbf{y}_{\min}(t) \leq \mathbf{y}(t) \leq \mathbf{y}_{\max}(t)$, $t \in [t_0, t_f]$.

Generally it cannot be guaranteed that a solution exists for an optimization problem with output constraints as the model outputs are determined by model states and model inputs. This is why output constraints should be relaxed to soft constraints, augmenting the optimization criterion (5) with penalties for violations. The HQP solver provides support for soft constraints.

3.4.4 Steady-state problem

The dynamic estimation and optimization problems discussed above can also be formulated as steady-state problems at one time instant $t = t_0 = t_f$. The steady-state condition

$$\dot{\mathbf{x}}(t) = \mathbf{0} \quad (6)$$

is formulated as constraint for the HQP optimization solver.

3.5 Discrete-Time Optimal Control Problem

Dynamic Optimization and Estimation problems are treated internally as discrete-time optimal control problems, applying multi-stage control vector parameterization. The time horizon $[t_0, t_f]$ is divided into K stages with $t_0 = t^0 < t^1 < \dots < t^K = t_f$. The controls $\mathbf{u}(t)$ are described in each interval $[t^k, t^{k+1}]$, $k = 0, \dots, K-1$ as function of the discrete-time input variables $\mathbf{u}^k \in \mathbf{R}^m$. The unknown parameters \mathbf{p} are converted to state variables with the state equation $\dot{\mathbf{p}} = \mathbf{0}$ and with unknown initial values $\mathbf{p}_0 = \mathbf{p}(t_0)$. They are described together with the continuous-time model states $\mathbf{x}(t)$ with the discrete-time state variables $\mathbf{x}^k \in \mathbf{R}^n$, $n = n_x + n_p$. The state equation (1) is solved for the stage k with the initial values \mathbf{x}^k and the controls \mathbf{u}^k using a numerical integration formula.

This results in the multistage optimization problem:

$$F^K(\mathbf{x}^K) + \sum_k f_0^k(\mathbf{x}^k, \mathbf{u}^k) \rightarrow \min_{\mathbf{u}^k, \mathbf{x}_0} \quad (7)$$

$$F^K : \mathbf{R}^n \mapsto \mathbf{R}^1, f_0^k : \mathbf{R}^n \times \mathbf{R}^m \mapsto \mathbf{R}^1$$

with respect to the discrete-time system equations

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{f}^k(\mathbf{x}^k, \mathbf{u}^k), \\ \mathbf{f}^k : \mathbf{R}^n \times \mathbf{R}^m &\mapsto \mathbf{R}^n \end{aligned} \quad (8)$$

and the additional constraints

$$\begin{aligned} \mathbf{c}_{\min}^k &\leq \mathbf{c}^k(\mathbf{x}^k, \mathbf{u}^k) \leq \mathbf{c}_{\max}^k, \\ \mathbf{c}_{\min}^K &\leq \mathbf{c}^K(\mathbf{x}^K) \leq \mathbf{c}_{\max}^K, \\ \mathbf{c}^k : \mathbf{R}^n \times \mathbf{R}^m &\mapsto \mathbf{R}^{m_k}, \mathbf{c}^K : \mathbf{R}^n \mapsto \mathbf{R}^{m_K}. \end{aligned} \quad (9)$$

Note that initial conditions of the system model are formulated as general constraints (9) as well. Discretization formulae, known parameter values, and predetermined disturbances are included into the discrete-time functions F^K , f_0^k , \mathbf{f}^k , \mathbf{c}^k , and \mathbf{c}^K . The discrete-time functions are assumed to be two times continuously differentiable with respect to their variables.

3.6 Large-Scale Nonlinear Programming Problem

Discrete-time optimal control problems can be solved as structured large-scale nonlinear optimization problems. This has the main advantage that powerful methods for large-scale nonlinear optimization can be applied to their efficient solution [11].

The discrete-time control and state variables for all stages k are collected to one large vector of optimization variables

$$\mathbf{v} = \begin{pmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^{K-1} \\ \mathbf{u}^{K-1} \\ \mathbf{x}^K \end{pmatrix}. \quad (10)$$

One specific feature of the optimization approach discussed here is that the discrete-time state variables at all stages are treated as optimization variables as well, even though they are determined by initial conditions and the control parameters. This leads to a significant increase of the size of the optimization problem. However, the consideration of states as constrained optimization variables generally improves robustness and efficiency of the solution. For instance trajectory constraints can be formulated directly on the discrete-time state variables. Furthermore the separation of the overall problem into multiple stages often leads to a reduction of the required number of nonlinear iterations. The computational overhead is relatively low if the number of state variables n_x is not too high, compared to the number of control variables n_u and if the sparse multistage structure of the large-scale nonlinear optimization problem is exploited appropriately.

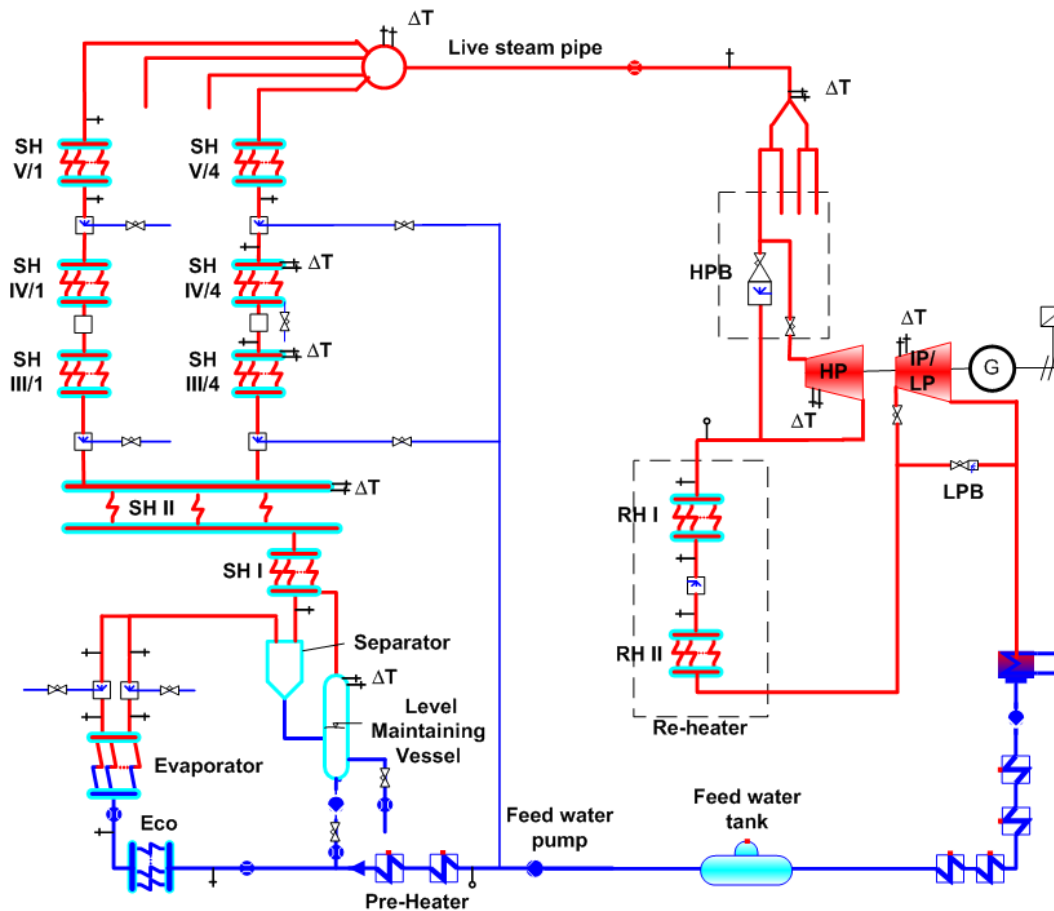


Figure 3: Simplified process diagram of a power plant.

4 Application example

A Nonlinear Model-based Predictive Controller (NMPC) for power plant start-up serves as example. The start-up problem is challenging as it is highly nonlinear in the covered large range of operation. Thermal stress occurring in thick walled components needs to be kept in given limits. Multiple manipulated variables have to be coordinated. A long prediction horizon is required to fulfill the constraints during a start-up.

Figure 3 shows a process diagram of a power plant. Feed water enters through pre-heaters and the economizer into the evaporator (lower left side). Saturated steam leaving the evaporator gets super-heated within several super-heater stages (the example diagram shows five super-heater stages and 4 parallel streams in the upper left part). The live steam leaving the boiler goes to the turbine (the example shows 2 turbine sections). There the thermal energy gets transformed to mechanical energy, driving the generator. Afterwards the steam gets condensed and water flows back to the feed water tank (lower right side of the di-

agram).

During start-up, the boiler first has to produce steam as required for starting the turbine. Within this phase, the steam bypasses the turbine through the high-pressure (HP) and low pressure (LP) bypass valves. The boiler gets heated up by several hundred degrees centigrade. This causes spatial temperature differences in thick walled parts, in particular headers behind the super-heaters and spherical fittings in the live steam pipe. Depending on the material properties, the spatial temperature differences cause thermal stress, which again causes fatigue up to destruction. This is why the thermal stress needs to be carefully observed and kept in prescribed limits.

A boiler model was built using the Modelica technology [5]. The model needs to be carefully designed so that it expresses the relationship between optimized control actions (fuel flow rate and valve positions) and constrained process values (pressures, temperatures and thermal stresses). In the example described here, a system of differential-algebraic equations (DAE) with about 1000 variables was built. The Dynamic Opti-

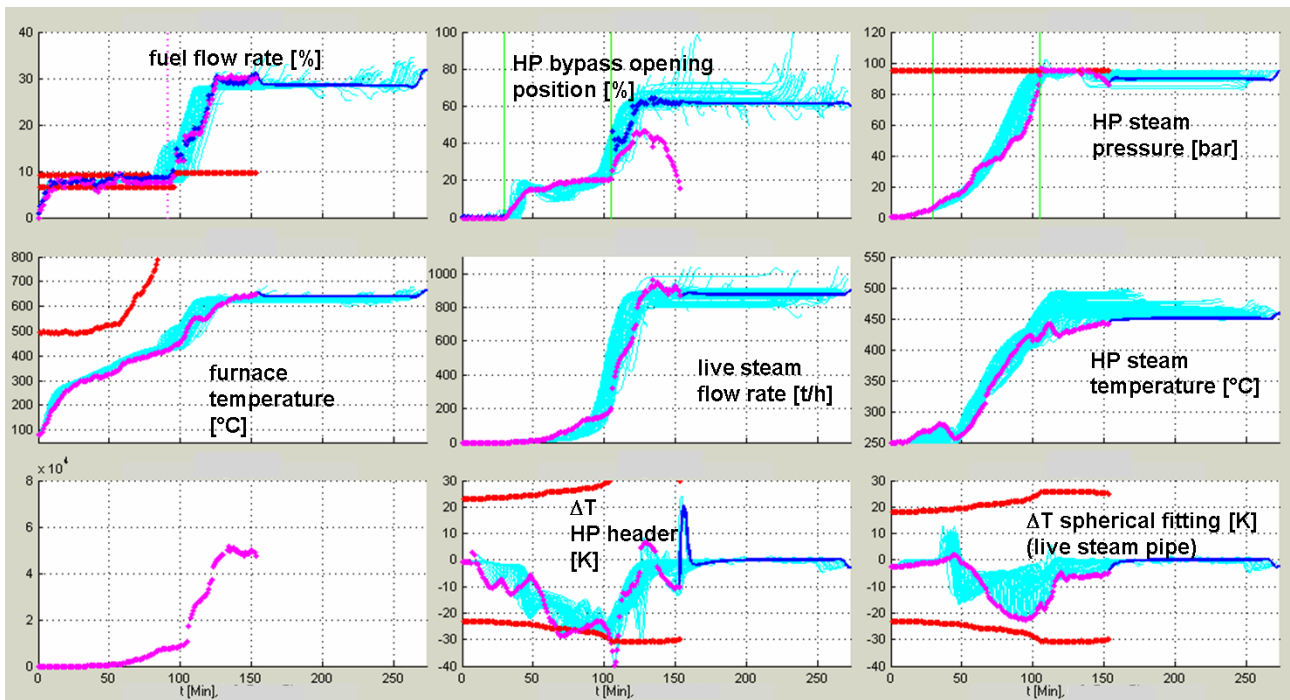


Figure 4: Traditional start-up. The dots show actual process values and limits, the light lines show predictions of process values over 90 minutes that are recalculated every minute. The dark lines show the most recent prediction.

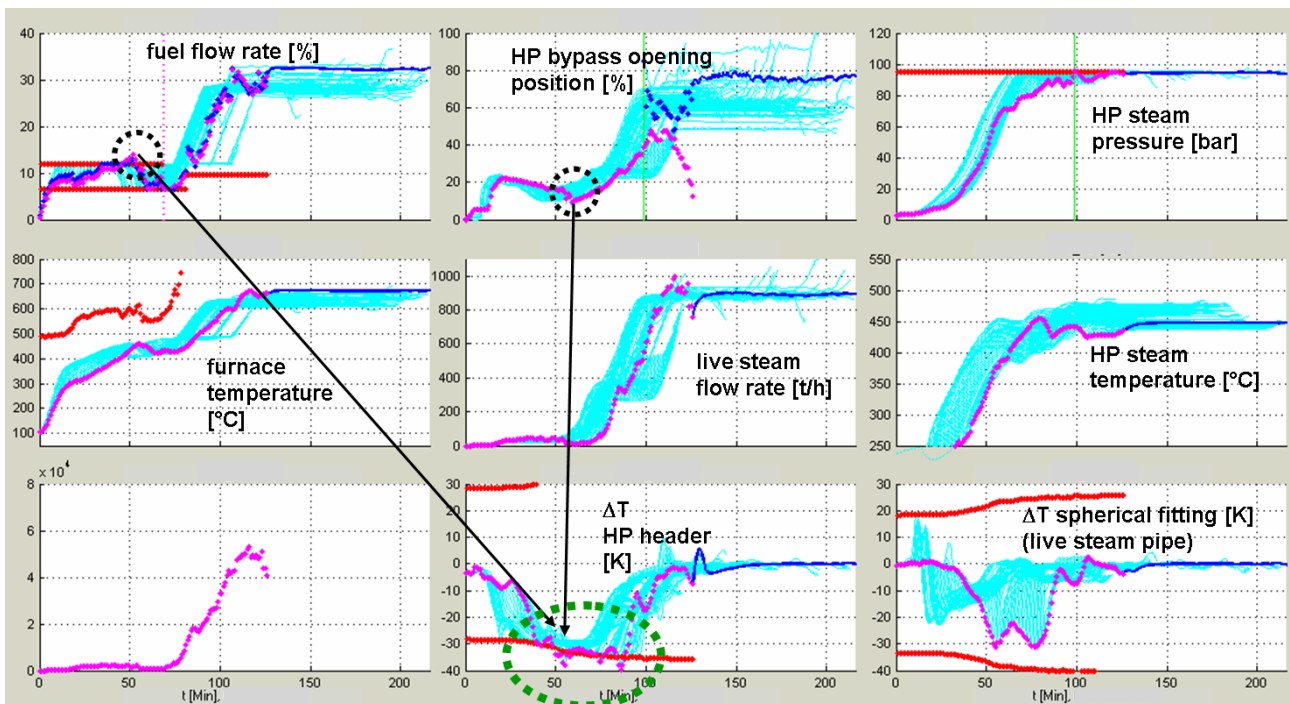


Figure 5: Optimized start-up performed with the NMPC online in closed loop.

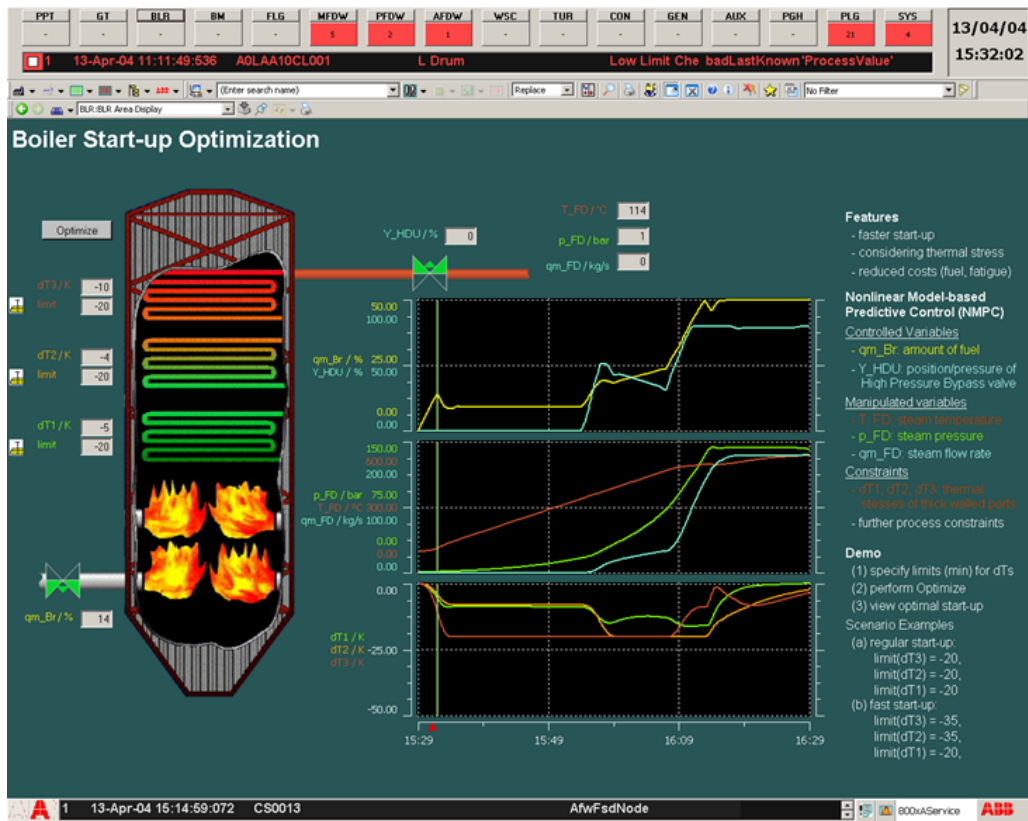


Figure 6: Operator display showing the optimal start-up predicted by the NMPC, in addition to current process values and history logs.

mization aspect system was used offline to identify model parameters based on data logs available for historical start-ups.

Figure 4 shows major process values for a start-up that was performed using well tuned standard control. The plant model was used online, open loop to check its ability to predict the future behavior of the process. The fuel flow rate and the HP bypass position are manipulated variables. The most important process variables are the furnace temperature, live steam pressure, temperature and flow rate, as well as thermal stresses. It can be seen that the allowed limits for thermal stress are not exploited during long time periods on the one hand side and that they exceed the allowed limits at other times (in particular ΔT HP header). The model was able to predict the behavior of the plant sufficiently well.

During a run of the NMPC, an optimization problem is solved online every minute. The time horizon (prediction and control) is 90 minutes in the example. It gets divided into 90 sample periods. The optimized manipulated variables are parameterized piecewise linear. All other model variables are evaluated at the sample time points. This means that overall about 91000 vari-

ables are present in the online optimization problem. The solution time is about five minutes for a cold start of the solver and about 40 seconds for a subsequent solver run.

Figure 5 shows the results of a start-up performed with the NMPC. Due to optimized use of the manipulated variables fuel flow rate and HP bypass position, the constraint on thermal stress of the HP header stays active during almost one hour. After about 50 minutes the fuel flow rate accidentally shot over, resulting in a violation of the thermal stress constraint. It can be seen how the NMPC reacted by immediately throttling the HP bypass valve and by reducing the fuel flow rate. Overall the start-up time could be reduced with the NMPC by about 20 minutes and the start-up costs by about 10% in a 700 MW coal fired power plant.

Figure 6 shows an operator display for boiler start-up optimization. Traditionally an operator display shows current process values and history logs. As a by-product of model predictive control, the operator can additionally see the prediction of the future behavior of the plant. As the NMPC runs integrated with the control system, this display can easily be configured.

5 Conclusions

The Modelica technology and the available model libraries allow an efficient modeling of many processes. Nevertheless nowadays the application of the models normally remains restricted to simulation studies conducted offline. A considerable additional effort is required to bring a model online and to deploy a mature model-based application.

The Dynamic Optimization system extension has been developed for the Industrial IT System 800xA by ABB to integrate model-based applications. Exploiting the powerful framework of the System 800xA, the effort for the development of the Dynamic Optimization system extension could be restricted to few additional software components. The new Model aspect exposes model data to the System 800xA. An additional modeling application like Dymola is used to build a Modelica model and to export C-code. The C-code is compiled to a stand-alone executable DLL and loaded by the HQP optimization solver at runtime. The new Dynamic Optimization aspect configures the HQP solver for a specific model based activity and it exchanges data like model parameters, process values and history logs between System 800xA and the HQP solver. The new aspects can be combined with other existing aspects in Aspect Objects. This allows the flexible structuring of complex model-based applications, consisting of multiple models and model-based activities. A configuration GUI has been developed as Excel add-in, which turned out to be a good compromise between development effort and achieved productivity. The Dynamic Optimization service manages the instantiation of solver activities in online applications. The Dynamic Optimization system extension has been applied so far in a number of different model-based applications. Nonlinear Model-based Predictive Control (NMPC) for the start-up of power plants is discussed in this paper as an example. The overall controller consists of four different model-based activities, including the pre-processing of process signals, the estimation of the model state, the prediction of the optimal start-up, and the post-processing of optimization results. The process models are based on the Modelica.Media and Modelica.Fluid libraries. The scheduling and supervision of the four activities has been implemented in the same framework as additional model-based activity, based on the Modelica.StateGraph library. After the successful application of the NMPC to the start-up of a 700 MW coal fired power plant, several more start-up optimizations are currently being deployed in gas, oil and coal fired power plants.

References

- [1] ABB Automation Technologies. Industrial IT System 800xA – System Architecture Overview. <http://www.abb.com>, Document Id: 3BUS092080R0101, 2005.
- [2] L.G. Bratthall, R. van der Geest, H. Hoffmann, E. Jellum, Z. Korendo, R. Martinez, M. Orkisz, C. Zeidler, and J. S Andersson. Integrating hundred's of products through one architecture – the Industrial IT architecture. In *International Convergence on Software Engineering*. Orlando, Florida, USA, 2002.
- [3] Dynasim AB. Dymola: Dynamic Modeling Laboratory. <http://www.dynasim.se>.
- [4] J. Eborn, H. Tummescheit, and K. Prölb. Airconditioning – a Modelica library for dynamic simulation of AC systems. In *Proceedings of the 4th International Modelica Conference*. Modelica Association, Hamburg-Harburg, Germany, March 2005.
- [5] H. Elmqvist, H. Tummescheit, and M. Otter. Modeling of thermo-fluid systems – Modelica.Media and Modelica.Fluid. In *Proceedings of the 3rd International Modelica Conference*. Modelica Association, Linköping, Sweden, November 2003.
- [6] R. Franke, E. Arnold, and H. Linke. HQP: a solver for nonlinearly constrained large-scale optimization. <http://hqp.sourceforge.net>.
- [7] R. Franke, K. Krüger, and M. Rode. Nonlinear model predictive control for optimized startup of steam boilers. In *GMA-Kongress 2003*. VDI-Verlag, Düsseldorf, 2003. VDI-Berichte Nr. 1756, ISBN 3-18-091756-3.
- [8] R. Franke, K. Krüger, and M. Rode. On-line optimization of drum boiler startup. In *Proceedings of the 3rd International Modelica Conference*. Modelica Association, Linköping, Sweden, November 2003.
- [9] International Electrotechnical Commission. Industrial systems, installations and equipment and industrial products – structuring principles and reference designations. IEC Standard 61346, 1996.
- [10] Modelica Association. Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Version 2.2. <http://www.modelica.org>, 2005.
- [11] Walter Murray. Sequential quadratic programming methods for large-scale problems. *Computational Optimization and Applications*, 7(1):127–142, 1997.
- [12] M. Otter, J. Årzén, and A. Schneider. StateGraph – a Modelica library for hierarchical state machines. In *Proceedings of the 4th International Modelica Conference*. Modelica Association, Hamburg-Harburg, Germany, March 2005.
- [13] The MathWorks, Inc. Simulink: for model-based and system level design. <http://www.mathworks.com>.

Types in the Modelica Language

David Broman Peter Fritzson Sébastien Furic
Linköping University, Sweden Linköping University, Sweden Imagine, France
davbr@ida.liu.se petfr@ida.liu.se

Abstract

Modelica is an object-oriented language designed for modeling and simulation of complex physical systems. To enable the possibility for an engineer to discover errors in a model, languages and compilers are making use of the concept of types and type checking. This paper gives an overview of the concept of types in the context of the Modelica language. Furthermore, a new concrete syntax for describing Modelica types is given as a starting point to formalize types in Modelica. Finally, it is concluded that the current state of the Modelica language specification is too informal and should in the long term be augmented by a formal definition.

Keywords: type system; types; Modelica; simulation; modeling; type safety

1 Introduction

One long term goal of modeling and simulation languages is to give engineers the possibility to discover modeling errors at an early stage, i.e., to discover problems in the model during design and not after simulation. This kind of verification is traditionally accomplished by the use of *types* in the language, where the process of checking for such errors by the compiler is called *type checking*. However, the concept of types is often not very well understood outside parts of the computer science community, which may result in misunderstandings when designing new languages. Why is then types important? Types in programming languages serve several important purposes such as naming of concepts, providing the compiler with information to ensure correct data manipulation, and enabling data abstraction. Almost all programming or modeling languages provide some kind of types. However, few language specifications include precise and formal definitions of

types and type systems. This may result in incompatible compilers and unexpected behavior when using the language.

The purpose of this paper is twofold. The first part gives an overview of the concept of types, states concrete definitions, and explains how this relates to the Modelica language. Hence, the first goal is to augment the computer science perspective of language design among the individuals involved in the Modelica language design. The long-term objective of this work is to provide aids for further design considerations when developing, enhancing and simplifying the Modelica language. The intended audience is consequently engineers and computer scientists interested in the foundation of the Modelica language.

The second purpose and likewise the main contribution of this work is the definition of a concrete syntax for describing Modelica types. This syntax together with rules of its usage can be seen as a starting point to more formally describe the type concept in the Modelica language. To the best of our knowledge, no work has previously been done to formalize the type concept of the Modelica language.

The paper is structured as follows: Section 2 outlines the concept of types, subtypes, type systems and inheritance, and how these concepts are used in Modelica and other mainstream languages. Section 3 gives an overview of the three main forms of polymorphism, and how these concepts correlate with each other and the Modelica language. The language concepts and definitions introduced in Section 2 and 3 are necessary to understand the rest of the paper. Section 4 introduces the type concept of Modelica more formally, where we give a concrete syntax for expressing Modelica types. Finally, Section 5 state concluding remarks and propose future work in the area.

2 Types, Subtyping and Inheritance

There exist several models of representing types, where the *ideal model* [3] is one of the most well-known. In this model, there is a universe V of all values, containing all values of integers, real numbers, strings and data structures such as tuples, records and functions. Here, types are defined as sets of elements of the universe V . There is infinite number of types, but all types are not legal types in a programming language. All legal types holding some specific property, such as being an unsigned integer, are called *ideals*. Figure 1 gives an example of the universe V and two ideals: real type and function type, where the latter has the *domain* of integer and *codomain* of boolean.

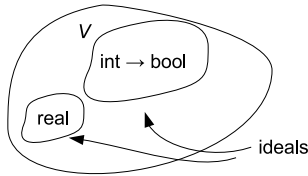


Figure 1: Schematic illustration of Universe V and two ideals.

In most mainstream languages, such as Java and C++, types are *explicitly typed* by stating information in the syntax. In other languages, such as Standard ML and Haskell, a large portion of types can be *inferred* by the compiler, i.e., the compiler deduces the type from the context. This process is referred to as *type inference* and such a language is said to be *implicitly typed*. Modelica is an explicitly typed language.

2.1 Language Safety and Type Systems

When a program is executed, or in the Modelica case: during simulation, different kinds of execution errors can take place. It is practical to distinguish between the following two types of runtime errors [2].

- *Untrapped errors* are errors that can go unnoticed and later cause arbitrary behavior of the system. For example, writing data out of bound of an array might not result in an immediate error, but the program might crash later during execution.
- *Trapped errors* are errors that force the computation to stop immediately; for example division by zero. The error can then be handled

by the run-time system or by a language construct, such as exception handling.

A programming language is said to be *safe* if no untrapped errors are allowed to occur. These checks can be performed as *compile-time checks*, also called *static checks*, where the compiler finds the potential errors and reports them to the programmer. Some errors, such as array out of bound errors are hard to resolve statically. Therefore, most languages are also using *run-time checks*, also called *dynamic checking*. However, note that the distinction between compile-time and run-time becomes vaguer when the language is intended for interpretation.

Typed languages can enforce language safety by making sure that *well-typed* programs cannot cause type errors. Such a language is often called *type safe* or *strongly typed*. This checking process is called *type checking* and can be carried out both at run-time and compile-time.

The behavior of the types in a language is expressed in a *type system*. A type system can be described informally using plain English text, or formally using *type rules*. The Modelica language specification is using the former informal approach. Formal type rules have much in common with logical inference rules, and might at first glance seem complex, but are fairly straightforward once the basic concepts are understood. Consider the following:

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{ (t-if)}$$

which illustrates a type rule for the following Modelica `if`-expression:

if e_1 **then** e_2 **else** e_3

A type rule is written using a number of *premises* located above the horizontal line and a *conclusion* below the line. The *typing judgement* $\Gamma \vdash e : T$ means that expression e has type T with respect to a static typing environment Γ . Hence, the rule (t-if) states that *guard* e_1 must have the type of a boolean and that e_2 and e_3 must have the same type, which is also the resulting type of the `if`-expression after evaluation. This resulting type is stated in the last part of the conclusion, i.e., $: T$.

If the language is described formally, we can attempt to prove the *type soundness theorem* [15]. If the theorem holds, the type system is said to be *sound* and the language *type safe* or just *safe*.

The concept of type safety can be illustrated by Robin Milner’s famous statement ”Well-typed programs cannot go wrong”[9]. Modern type soundness proofs are based on Wright and Felleisen’s approach where type systems are proven correct together with the language’s operational semantics [15]. Using this technique, informally stated, type safety hold if and only if the following two statements holds:

- *Preservation* - If an expression e has a type T and e evaluates to a value v , then v also has type T .
- *Progress* - If an expression e has a type T then either e evaluates to a new expression e' or e is a value. This means that a well typed program never gets ”stuck”, i.e., it cannot go into a undefined state where no further evaluations are possible.

Note that the above properties of type safety corresponds to our previous description of absence of untrapped errors. For example, if a division by zero error occurs, and the semantics for such event is undefined, the progress property will not hold, i.e., the evaluation gets ”stuck”, or enters an undefined state. However, if dynamic semantics are defined for throwing an exception when the division by zero operation occurs, the progress property holds.

For the imperative and functional parts of the Modelica language, the safety concept corresponds to the same methodology as other languages, such as Standard ML. However, for the instantiation process of models, the correspondence to the progress and preservation properties are not obvious.

Table 1 lists a number of programming languages and their properties of being type safe [10][2]. The table indicates if the languages are primarily designed to be checked statically at compile-time or dynamically at run-time. However, the languages stated to be statically type checked typically still perform some checking at runtime.

Although many of the languages are commonly believed to be safe, few have been formally proven to be so. Currently, ML [9] and subsets of the Java language [14] [7] has been proven to be safe.

Language	Type Safe	Checking
Standard ML	yes	static
Java	yes	static
Common LISP	yes	dynamic
Modelica	yes	static ¹
Pascal	almost	static
C/C++	no	static
Assembler	no	-

Table 1: Believed type safety of selected languages.

2.2 Subtyping

Subtyping is a fundamental language concept used in most modern programming languages. It means that if a type S has all the properties of another type T , then S can be safely used in all contexts where type T is expected. This view of subtyping is often called *the principle of safe substitution* [12]. In this case, S is said to be a subtype of T , which is written as

$$S <: T \quad (1)$$

This relation can be described using the following important type rule called the *rule of subsumption*.

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T} \text{ (t-sub)}$$

The rule states that if $S <: T$, then every *term*² t of type S is also a term of type T . This shows a special form of *polymorphism*, which we will further explore in Section 3.

2.3 Inheritance

Inheritance is a fundamental language concept found in basically all class based *Object-Oriented (OO)* languages. From an existing *base class*, a new *subclass* can be created by *extending* from the base class, resulting in the subclass *inheriting* all properties from the base class. One of the main purposes with inheritance is to save programming

¹One can argue whether Modelica is statically or dynamically checked, depending on how the terms compile-time and run-time are defined. Furthermore, since no exception handling is currently part of the language, semantics for handling dynamic errors such as array out of bound is not defined in the language and is therefore considered a compiler implementation issue.

²The word *term* is commonly used in the literature as an interchangeable name for expression.

and maintenance efforts of duplicating and reading duplicates of code. Inheritance can in principle be seen as an implicit code duplication which in some circumstances implies that the subclass becomes a subtype of the type of the base class.

Figure 2 shows an example³ where inheritance is used in Modelica. A *model* called `Resistor` extends from a base class `TwoPin`, which includes two elements `v` for voltage and `i` for current. Furthermore, two instances `p` and `n` of connector `Pin` are public elements of `TwoPin`. Since `Resistor` extends from `TwoPin`, all elements `v`, `i`, `p` and `n` are "copied" to class `Resistor`. In this case, the type of `Resistor` will also be a subtype of `TwoPin`'s type.

```
connector Pin
  SI.Voltage v;
  flow SI.Current i;
end Pin;

partial model TwoPin
  SI.Voltage v;
  SI.Current i;
  Pin p, n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;

model Resistor
  extends TwoPin;
  parameter SI.Resistance R=100;
equation
  R*i = v;
end Resistor;
```

Figure 2: Example of inheritance in Modelica, where a new subclass `Resistor` is created by extending the base class `TwoPin`.

However, a common misunderstanding is that subtyping and inheritance is the same concept [10]. A simple informal distinction is to say that "subtyping is a relation on interfaces", but "inheritance is a relation on implementations". In the resistor example, not only the public elements `v`, `i`, `p` and `n` will be part of class `Resistor`, but also the meaning of this class, i.e., the equations $v = p.v - n.v$, $0 = p.i + n.i$ and $i = p.i$.

³These classes are available in the Modelica Standard Library 2.2, but are slightly modified for reasons of readability.

A famous example, originally stated by Alan Snyder [13], illustrates the difference between subtyping and inheritance. Three common *abstract data types* for storing data objects are *queue*, *stack* and *dequeue*. A queue normally has two operations, *insert* and *delete*, which stores and returns object in a *first-in-first-out (FIFO)* manner. A stack has the same operations, but are using a *last-in-first out (LIFO)* principle. A dequeue can operate as both a stack and a queue, and is normally implemented as a list, which allows inserts and removals at both the front and the end of the list.

Figure 3 shows two C++ classes modeling the properties of a dequeue and a stack. Since class `Dequeue` implements the properties also needed for a stack, it seems natural to create a subclass `Stack` that inherits the implementation from `Dequeue`. In C++, it is possible to use so called *private inheritance* to model inheritance with an *exclude operation*, i.e., to inherit some, but not all properties of a base class. In the example, the public methods `insFront`, `delFront`, and `delRear` in class `Dequeue` are inherited to be private in the subclass `Stack`. However, by adding new methods `insFront` and `delFront` in class `Stack`, we have created a subclass, which has the property of a stack by excluding the method `delRear`. `Stack` is obviously a subclass

```
class Dequeue{
public:
  void insFront(int e);
  int delFront();
  int delRear();
};

class Stack : private Dequeue{
public:
  void insFront(int e)
    {Dequeue::insFront(e);}
  int delFront()
    {return Dequeue::delFront();}
};
```

Figure 3: C++ example, where inheritance does not imply a subtype relationship.

of `Dequeue`, but is it a subtype? The answer is no, since an instance of `Stack` cannot be safely used when `Dequeue` is expected. In fact, the opposite is true, i.e., `Dequeue` is a subtype of `Stack` and not the other way around. However, in the following section we will see that C++ does not

treat such a subtype relationship as valid, but the type system of Modelica would do so.

2.4 Structural and Nominal Type Systems

During type checking, regardless if it takes place at compile-time or run-time, the type checking algorithm must control the relations between types to see if they are correct or not. Two of the most fundamental relations are *subtyping* and *type equivalence*.

Checking of type equivalence is the single most common operation during type checking. For example, in Modelica it is required that the left and right side of the equality in an equation have the same type, which is shown in the following type rule.

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 = e_2 : \text{Unit}} \text{ (t-equ)}$$

Note that type equivalence has nothing to do with equivalence of values, e.g., equation $4 = 10$ is type correct, since integers 4 and 10 are type equivalent. However, this is of course not a valid equation, since the values on the right and left side are not the same.

The *Unit* type (not to confuse with physical units), shown as the resulting type of the equation, is often used as a type for uninteresting result values.

A closely related concept to type equivalence is *type declaration*, i.e., when a type is declared as a specific *name* or *identifier*. For example, the following Modelica record declaration

```
record Person
  String name;
  Integer age;
end Person;
```

declares a type with name *Person*. Some languages would treat this as a new unique type that is not equal to any other type. This is called *opaque* type declaration. In other languages, this declaration would simply mean that an alternative name is given to this type. However, the type can also be expressed by other names or without any name. This latter concept is commonly referred as *transparent* type declaration.

In a pure *nominal type system*, types are compared (subtyping and type equivalence) by using the *names* of the declared types, i.e., opaque type

declarations are used. Type equivalence is controlled by checking that the same declared name is used. Furthermore, the subtype relation in OO languages is checked by validating the inheritance order between classes. The C++ language is mainly using a nominal type system, even if parts of the language does not obey the strict nominal structure.

Consider the listing in Figure 4, which illustrates a C++ model similar to the resistor example earlier given as Modelica code in Figure 2. In this case, *Resistor* is a subclass of *TwoPin* and the type of *Resistor* is therefore also a subtype of *TwoPin*'s type. However, the type of *Inductor* is not a subtype to the type of *TwoPin*, since *Inductor* does not inherit from *TwoPin*. Moreover, *Resistor2* is not type equivalent to *Resistor* even if they have the same structure and inherits from the same base class, since they are opaquely declared.

```
class Pin{
public:
  float v, i;
};

class TwoPin{
public:
  TwoPin() : v(0), i(0) {};
  float v, i;
  Pin p, n;
};

class Resistor : public TwoPin{
public:
  Resistor() : r(100) {};
  float r;
};

class Resistor2 : public TwoPin{
public:
  Resistor() : r(200) {};
  float r;
};

class Inductor{
public:
  TwoPin() : v(0), i(0) {};
  float v, i;
  Pin p, n;
  const float L;
};
```

Figure 4: Resistor inheritance example in C++.

In a *structural type system* [12], declarations are introducing new names for type expressions, but no new types are created. Type equivalence and subtype relationship is only decided depending on the structure of the type, not the naming.

The Modelica language is inspired by the type system described by Abadi and Cardelli in [1] and is using transparent type declarations, i.e., Modelica has a structural type system. Consider the `Resistor` example given in Figure 2 and the two complementary models `Inductor` and `Resistor2` in Figure 5. Here, the same relations holds between `TwoPin` and `Resistor`, i.e., that the type of `Resistor` is a subtype of `TwoPin`'s type. The same holds between `TwoPin` and `Resistor2`. However, now `Resistor` and `Resistor2` are type equivalent, since they have the same structure and naming of their public elements. Furthermore, the type of `Inductor` is now a valid subtype of `TwoPin`'s type, since `Inductor` contains all public elements (type and name) of the one available in `TwoPin`.

```

model Resistor2
  extends TwoPin;
  parameter SI.Resistance R=200;
equation
  R*i = v;
end Resistor;

model Inductor
  Pin p, n;
  SI.Voltage v;
  SI.Current i;
  parameter SI.Inductance L=1;
equation
  L*der(i) = v;
end Inductor;

```

Figure 5: Complementary `Inductor` and `Resistor2` models to the example in Figure 2.

It is important to stress that *classes* and *types* in a structural type system are **not** the same thing, which also holds for Modelica. The type of a class represents the interface of the class relevant to the language's type rules. The type does not include implementation details, such as equations and algorithms.

Note that a nominal type system is more restrictive than a structural type system, i.e., two types that have a structured subtype relation can always have a subtype relation by names (if the language's semantics allows it). However, the op-

posite is not always true. Recall the `Dequeue` example listed in Figure 3. The class `Stack` has a subclass relation to `Dequeue`, but a subtype relation cannot be enforced, due to the structure of the class. The converse could be true, but the type system of C++ would not allow it, since it is nominal and subtype relationships are based on names. Hence, a structural type system can be seen as more *expressive* and *flexible* compared to a nominal one, even if both gives the same level of language type safety.

3 Polymorphism

A type system can be *monomorphic* in which each value can belong to at most one type. A type system, as illustrated in Figure 1, consisting of the distinct types function, integer, real, and boolean is a monomorphic type system. Conversely, in a *polymorphic* type system, each value can belong to many different types. Languages supporting polymorphism are in general more expressive compared to languages only supporting monomorphic types. The concept of polymorphism can be handled in various forms and have different naming depending on the paradigm where it is used. Following John C. Mitchell's categorization, polymorphism can be divided into the following three main categories [10]:

- Subtype Polymorphism
- Parametric Polymorphism
- Ad-hoc Polymorphism

There are other similar categorizations, such as Cardelli and Wegner's [3], where the ad-hoc category is divided into *overloading* and *coercion* at the top level of categories.

3.1 Subtype Polymorphism

Subtyping is an obvious way that gives polymorphic behavior in a language. For example, an instance of `Resistor` can be represented both as an `TwoPin` type and a `Resistor` type. This statement can also be shown according to the rule of subsumption (t-sub) described in Section 2.2.

When a value is changed from one type to some supertype, it is said to be an *up-cast*. Up-casts can be viewed as a form of *abstraction* or *information hiding*, where parts of the value becomes

invisible to the context. For example, an up-cast from `Resistor`'s type to `TwoPin`'s type hides the parameter `R`. Up-casts are always type safe, i.e., the run-time behavior cannot change due to the upcast.

However, for subtype polymorphism to be useful, typically types should be possible to *down-cast*, i.e., to change to a subtype of a type's value. Consider function `Foo`

```
function Foo
  input TwoPin x;
  output TwoPin y;
end Foo;
```

where we assume that down-casting is allowed⁴. It is in this case valid to pass either a value of type `TwoPin` (type equivalence) or a subtype to the type of `TwoPin`. Regardless if a value of `TwoPin`'s or `Inductor`'s type is sent as input to the function, a value of `TwoPin`'s type will be returned. It is not possible for the static type system to know if this is a `TwoPin`, `Resistor` or a `Inductor` type. However, for the user of the function, it might be crucial to handle it as an `Inductor`, which is why a down-cast is necessary.

Down-casting is however not a safe operation, since it might cast down to the wrong subtype. In Java, before version 1.5 when *generics* were introduced, this safety issue was handled using dynamic checks and raising dynamic exceptions if an illegal down-cast was made. Subtype polymorphism is sometimes called "poor-man's polymorphism", since it enables polymorphic behavior, but the safety of down-casts must be handled dynamically [12].

The Modelica language supports subtyping as explained previously, but does not have any operation for down-cast. Since the language does not include this unsafe operation, only a limited form of subtype polymorphism can be used with functions. For example, a function can operate on a polymorphic type as input, such as `TwoPin`, but it only makes sense to return values of a type that can be instantly used by the caller.

However, subtype polymorphism is more extensively used when reusing and replacing components in models, i.e., by using the `redeclare` keyword.

⁴This function type or example is not valid in the current Modelica standard. It is used only for the purpose of demonstrating subtype polymorphism.

3.2 Parametric Polymorphism

The term *parametric polymorphism* means that functions or classes can have *type parameters*, to which types or *type expressions* can be supplied. The term parametric polymorphism is often used in functional language communities, while people related to object-oriented languages tend to use the term *generics*.

The C++ *template* mechanism is an example of *explicit parametric polymorphism*, where the type parameter must be explicitly declared. Consider for example Figure 6, where a template function `swap` is implemented. The type parameter `T` must be explicitly stated when declaring the function. However, the type argument is not needed when calling the function, e.g., both `int x, y; swap(x, y);` and `float i, j; swap(i, j)` are valid usage of the function.

```
template<typename T>
void swap(T& x, T& y) {
  T tmp = x;
  x = y;
  y = tmp;
}
```

Figure 6: Explicit parametric polymorphism in C++.

Standard ML on the other hand is making use of *implicit parametric polymorphism*, where the type parameters do not need to be explicitly stated when declaring the function. Instead, the *type inference algorithm* computes when type parameters are needed.

A notable difference of parametric and subtype polymorphism is that all type checking of parametric polymorphism can take place at compile-time and no unsafe down-cast operation is needed.

Standard ML and C++ are internally handling parametric polymorphism quite differently. In C++ templates, instantiation to compiled code of a function is done at link time. If for example function `swap` is called both using `int` and `float`, different code of the function is generated for the two function calls. Standard ML on the other hand is using *uniform data representation*, where all data objects are represented internally as pointers/references to objects. Therefore, there is no need to create different copies of code for different types of arguments.

Modelica can be seen to support a limited version of parametric polymorphism, by using the *re-declare* construct on local class declarations.

3.3 Ad-hoc Polymorphism

In parametric polymorphism the purpose is to declare one implementation that can be used with different types of arguments. *Ad-hoc polymorphism*, by contrast, allows a polymorphic value to be used differently depending on which type the value is viewed to have.

There are several language concepts that fall under the concept of ad-hoc polymorphism [3], where *Overloading* and *Coercion* are most notable. Other related concepts that also fall under this category are Java's `instanceOf` concept and different form of *pattern matching* [12].

3.3.1 Overloading

A symbol is *overloaded* if it has two or more meanings, which are distinguished by using types. That is, a single function symbol or identifier is associated with several implementations.

An example of overloading that exists in many programming languages is *operator overloading* for built in types. For example, the symbol `+` is using infix notation and have two operands associated with it. The type of these operands decide how the operation should be carried out, i.e., which implementation that should be used.

Overloading can take place at either compile-time or at run-time. Overloading used at run-time is often referred to as *dynamic lookup* [10], *dynamic dispatch* or *multi-method dispatch*. In most cases, the single term overloading refers to static overloading taking place at compile-time. The distinction becomes of course vague, if the language is *interpreted* and not compiled.

Another form of overloading available in some languages is user-defined *function overloading*, where a function identifier can represent several implementations for different type arguments. Modelica is currently not supporting any form of user defined overloading.

3.3.2 Coercion

Another form of ad-hoc polymorphism is *coercion* or *implicit type conversion*, which is run-time conversion between types, typically performed by code automatically inserted by the compiler. The

distinction between overloading and type coercion is not always clear, and the two concepts are strongly related. Consider the following four expressions of multiplication [3]:

```
7 * 9 //Integer * Integer
6.0 * 9.1 //Real * Real
6 * 5.2 //Integer * Real
6.0 * 8 //Real * Integer
```

All four of these expressions are valid Modelica expressions, but they can in the context of coercion and overloading be interpreted in three different ways:

- The multiplication operator is overloaded four times, one for each of the four expressions.
- The operator is overloaded twice; one for each of the the first two expressions. If the arguments have different types, i.e., one is `Real` and the other one `Integer`, type coercion is first performed to convert the arguments to `Real`.
- Arguments are always implicitly converted to `Real`, and the operator is only defined for `Reals`.

Type conversions can also be made *explicit*, i.e., code is inserted manually by the programmer that converts the expression to the correct type.

In Modelica, implicit type conversion is used when converting from `Integer` to `Real`. Of the three different cases listed above, the second one applies to the current Modelica 2.2 standard.

4 Modelica Types

In the previous sections we described different aspects of types for various languages. In this section we will present a concrete syntax for describing Modelica types, followed by rules stating legal type expressions for the language.

The current Modelica language specification [11] specifies a formal syntax of the language, but the semantics including the type system are given informally using plain English. There is no explicit definition of the type system, but an implicit description can be derived by reading the text describing relations between types and classes in the Modelica specification. This kind of implicit specification makes the actual specification open for interpretation, which may result in incompatible

compilers; both between each other, but also to the specification itself. Our work in this section should be seen as a first step to formalize what a type in Modelica actually is. Previous work has been performed to formally specify the semantics of the language [8], but without the aim to more precisely define the exact meaning of a type in the language.

Why is it then so important to have a precise definition of the types in a language? As we have described earlier, a type can be seen as an interface to a class or an object. The concept of interfaces forms the basis for the widely accepted approach of separating *specification* from *implementation*, which is particularly important in large scale development projects. To put it in a Modelica modeling context, let us consider a modeling project of a car, where different modeling teams are working on the wheels, gearbox and the engine. Each team has committed to provide a set of specific attributes for their component, which specifies the interface. The contract between the teams is not violated, as long as the individual teams are following this commitment of interface (the specification) by adding / removing equations (the implementation). Since the types state the interfaces in a language with a structural type system, such as Modelica, it is obviously decisive that they have a precise definition.

Our aim here is to define a precise notation of types for a subset of the Modelica language, which can then further be extended to the whole language. Since the Modelica language specification is open for interpretation, the presented type definition is our interpretation of the specification.

4.1 Concrete Syntax of Types

Now, let us study the types of some concrete Modelica models. Consider the following model B, which is rather uninteresting from a physical point of view, but demonstrates some key concepts regarding types.

```

model B
  parameter Real s=-0.5;
  connector C
    flow Real p;
    Real q;
  end C;
protected
  Real x(start=1);
equation
  der(x) = s*x;
end B;

```

What is the type of model B? Furthermore, if B was used and instantiated as a component in another model, e.g., B b;, what would the resulting type for element b be? Would the type for B and b be the same? The answer to the last question is definitely no. Consider the following listing, which illustrates the type of model B.

```

model classtype //Class type of model B
  public parameter Real objtype s;
  public connector classtype
    flow Real objtype p;
    nonflow Real objtype q;
  end C;
  protected Real objtype x;
end

```

This type listing follows the grammar syntax listed in Figure 7. The first thing to notice is that the name of model B is not visible in the type. Recall that Modelica is using a structural type system, where the types are determined by the structure and not the names, i.e., the type of model B has nothing to do with the name B. However, the names of the *elements* in a type are part of the type, as we can see for parameter s and variable x.

The second thing to observe is that the equation part of the model is missing in the type definition. The reason for this is that equations and algorithms are part of the implementation and not the model interface. Moreover, all elements s, C and x are preserved in the type, but the keywords model, connector and basic type Real are followed by new keywords classtype or objtype. This is one of the most important observations to make regarding types in a class based system using structural subtyping and type equivalence. As we can see in the example, the type of model B is a *class type*, but parameter s is an *object type*. Simply stated: A class type is the type of one of Modelica's restricted classes, such as model, connector, record etc., but an *object type* is the type of an instance of a class, i.e., an object. Now, the following shows the object type of b, where b represents an instance of model B:

```

model objtype //Object type of b
  parameter Real objtype s;
end

```

Obviously, both the type of connector C and variable x have been removed from the type of b. The reason is that an object is a run-time entity, where neither local classes (connector C) nor protected elements (variable x) are accessible from

outside the instance. However, note that this is not the same as that variable `x` does not exist in an instance of `B`; it only means that it is not visible to the outside world.

Now, the following basic distinctions can be made between *class types* and *object types*:

- Classes can inherit (using `extends`) from class types, i.e., the type that is bound to the name used in an `extends` clause must be a class type and not an object type.
- Class types can contain both object types and class types, but object types can only hold other object types.
- Class types can contain types of protected elements; object types cannot.
- Class types are used for compile time evaluation, such as inheritance and redeclarations.

```

type ::= (model | record | connector |
        block | function | package)
        kindoftype
        {{prefix} type identifier ;} end
    | (Real | Integer | Boolean |
        String) kindoftype
    | enumeration kindoftype
        enumlist
kindoftype ::= classtype | objtype
prefix ::= access | causality |
        flowprefix | modifiability |
        variability | outerinner
enumlist ::= ( identifier {, identifier} )
access ::= public | protected
causality ::= input | output |
            inputoutput
flowprefix ::= flow | nonflow
modifiability ::= replaceable | modifiable |
              final
variability ::= constant | parameter |
            discrete | continuous
outerinner ::= outer | inner |
            notouterinner

```

Figure 7: Concrete syntax of partial Modelica types.

Let us now take a closer look at the grammar listed in Figure 7. The root non-terminal of the grammar is *type*, which can form a class or object type of the restricted classes or the built in types `Real`, `Integer`, `Boolean`, `String`, or enumeration. The grammar is given using a variant of *Extended Backus-Naur Form* (EBNF), where terms enclosed in brackets `{ }` denote zero, one or more repetitions. Keywords appearing in the concrete syntax are given in bold font. All prefixes, such as `public`, `flow`, `outer` etc. can be given infinitely many times. The correct usage of these prefixes is not enforced by the grammar, and must therefore be handled later in the semantic analysis. We will give guidelines for default prefixes and restrictions of the usage of prefixes in the next subsection.

Now, let us introduce another model `A`, which extends from model `B`:

```

model A
  extends B(s=4);
  C c1;
equation
  c1.q = -10*der(x);
end A;

```

The question is now what the type of model `A` is and if it is instantiated to an object, i.e., `A a;`, what is then the type of `a`? The following shows the type of model `A`.

```

model classtype //Class type of A
  public parameter Real objtype s;
  public connector classtype
    flow Real objtype p;
    nonflow Real objtype q;
  end C;
  public connector objtype
    flow Real objtype p;
    nonflow Real objtype q;
  end c1;
  protected Real objtype x;
end

```

First of all, we see that the type of model `A` does not include any `extends` keyword referring to the inherited model `B`. Since Modelica has a structural type system, it is the structure that is interesting, and thus a type only contains the collapsed structure of inherited elements. Furthermore, we can see that the protected elements from `B` are still available, i.e., inheritance preserves the protected element after inheritance. Moreover, since model `A` contains an instance of connector `C`, this is now available as an object type for element `c1` in the class type of `A`. Finally, consider the type of an

instance a of class A:

```

model objtype //Object type of a
  parameter Real objtype s;
  connector objtype
    flow Real objtype p;
    nonflow Real objtype q;
  end cl;
end

```

The protected element is now gone, along with the elements representing class types. A careful reader might have noticed that each type definition ends without a semi-colon, but elements defined inside a type such as `model classtype` ends with a semi-colon. A closer look at the grammar should make it clear that types themselves do not have names, but when part of an element definition, the type is followed by a name and a semi-colon. If type expressions were to be ended with a semi-colon, this recursive form of defining concrete types would not be possible.

4.2 Prefixes in Types

Elements of a Modelica class can be prefixed with different notations, such as `public`, `outer` or `replaceable`. We do not intend to describe the semantics of these prefixes here, instead we refer to the specification [11] and to the more accessible description in [5]. Most of the languages prefixes have been introduced in the grammar in Figure 7. However, not all prefixes are allowed or have any semantic meaning in all contexts.

In this subsection, we present a partial definition of when different prefixes are allowed to appear in a type. In currently available tools for Modelica, such as Dymola [4] and OpenModelica [6], the enforcement of these restrictions is sparse. The reason for this can both be the difficulties to extract this information from the specification and the fact that the rules for the type prefixes are very complex.

In Figure 8 several abbreviations are listed. The lower case abbreviations a , c , c' etc. define sets of prefixes. The uppercase abbreviations M , R etc. together with a subscription of c for class type and o for object type, represents the type of an element part of another type. For example M_c is a model class type, and R_o is a record object type.

Now, consider the rules for allowed prefixes of elements shown in the tables given in Figure 9, Figure 10, and Figure 11.

In Figure 9 the intersection between the column (the type of an element) and the row (the

M	=	model	
R	=	record	
C	=	connector	
B	=	block	
F	=	function	
P	=	package	
X	=	Integer, Boolean, enumeration, String	
Y	=	Real	
a	=	{ <u>public</u> , <u>protected</u> }	Access
a'	=	{ <u>public</u> }	
c	=	{ <u>input</u> , <u>output</u> , <u>inputoutput</u> }	Causality
c'	=	{ <u>input</u> , <u>output</u> }	
f	=	{ <u>flow</u> , <u>nonflow</u> }	Flowprefix
m	=	{ <u>replaceable</u> , <u>modifiable</u> , <u>final</u> }	Modifiability
m'	=	{ <u>modifiable</u> , <u>final</u> }	
v	=	{ <u>constant</u> , <u>parameter</u> <u>discrete</u> , <u>continuous</u> }	Variability
v'	=	{ <u>constant</u> , <u>parameter</u> <u>discrete</u> }	
v''	=	{ <u>constant</u> }	
o	=	{ <u>outer</u> , <u>inner</u> , <u>notouterinner</u> }	Outerinner

Figure 8: Abbreviation for describing allowed prefixes. Default prefixes are underlined.

	M_c	R_c	C_c	B_c	F_c	P_c	X_c	Y_c
M_c	<i>amo</i>	<i>amo</i>	<i>amo</i>	<i>amo</i>	<i>amo</i>	.	<i>amo</i>	<i>amo</i>
R_c
C_c
B_c	<i>amo</i>	<i>amo</i>	<i>amo</i>	<i>amo</i>	<i>amo</i>	.	<i>amo</i>	<i>amo</i>
F_c	.	<i>am</i>	.	.	<i>am</i>	.	<i>am</i>	<i>am</i>
P_c	<i>am</i>	<i>amv''</i>	<i>am</i>	<i>am</i>	<i>am</i>	<i>d'm</i>	<i>am</i>	<i>am</i>

Figure 9: Prefixes allowed for elements of class type (columns) inside a class type (rows).

	M_o	R_o	C_o	B_o	F_o	P_o	X_o	Y_o
M_c	<i>amo</i>	<i>acmo</i>	<i>acmo</i>	<i>amo</i>	<i>amo</i>	.	<i>acmv'o</i>	<i>acmvo</i>
R_c	.	<i>mo</i>	<i>mv'o</i>	<i>mvo</i>
C_c	.	<i>mo</i>	<i>mo</i>	.	.	.	<i>m</i>	<i>mcfv'o</i>
B_c	<i>amo</i>	<i>ac'mo</i>	<i>ac'mo</i>	<i>amo</i>	<i>amo</i>	.	<i>ac'mv'o</i>	<i>ac'mvo</i>
F_c	.	<i>ac'm</i>	.	.	<i>am</i>	.	<i>ac'mv'</i>	<i>ac'mv</i>
P_c	.	<i>amv''</i>	<i>amv''</i>	<i>amv''</i>

Figure 10: Prefixes allowed for elements of object type (columns) inside a class type (rows).

type that contains this element) states the allowed prefixes for this particular element. This table shows which prefixes that are allowed for a class type that is part of another class type. For example, recall the connector C in model A . When looking at the type of A , we have a class type (the model class type) that contains an-

	M_o	R_o	C_o	B_o	F_o	P_o	X_o	Y_o
M_o	<i>o</i>	<i>cm'o</i>	<i>co</i>	<i>o</i>	<i>o</i>	.	<i>cm'v'o</i>	<i>cm'vo</i>
R_o	.	<i>m'o</i>	<i>m'v'o</i>	<i>m'vo</i>
C_o	.	<i>m'o</i>	<i>o</i>	<i>cfm'vo</i>
B_o	<i>o</i>	<i>c'o</i>	<i>c'o</i>	<i>o</i>	<i>o</i>	.	<i>c'm'v'o</i>	<i>c'm'vo</i>
F_o	.	<i>c'</i>	<i>m'v'</i>	<i>m'v</i>
P_o

Figure 11: Prefixes allowed for elements of object type (columns) inside an object type (rows).

other class type (the connector class type), i.e., the allowed prefixes are given in the intersection of row 1 and column 3. In this case, *access* prefixes `public` and `protected`, *modifiability* prefixes `replaceable`, `modifiable`, and `final`, and *outer/inner* prefixes `outer`, `inner` and `notouterinner` are allowed.

We have introduced a number of new prefixes: `inputoutput`, `notouterinner`, `nonflow`, `modifiable`, and `continuous`. These new prefixes are introduced to enable a complete type definition, e.g., it should be possible to explicitly specify that a variable in a connector is not a flow variable by giving a `nonflow` prefix. However, for simplicity, sometimes it is more convenient to leave out some of the prefixes, and instead use default prefixes. The defined default prefixes are show underlined in Figure 8. If no underlined prefix exists in a specific set, this implies that the prefix must be explicitly stated.

Analogous to the description of Figure 9, Figure 10 shows the allowed prefixes for elements of object types contained in a class type and Figure 11 shows object types contained in object types. There are no tables given for class types contained in object types for the simple reason that object types are not allowed to contain class types.

In some of the cells in the tables described above, a dot symbol is shown. This means that the specific type of element inside a certain type is not allowed. Hence, such a combination should not be allowed by the compiler at compile-time.

Now, let us observe some general trends between the allowed attributes. First of all, object types cannot contain class types, which is why there are only 3 tables. Secondly, *access* prefixes (`public`, `protected`) are only allowed in class types, which is why Figure 11 does not contain any abbreviation *a*. Thirdly, the *replaceable* prefix does not make sense in object types, since redeclarations may only occur during object cre-

ation or inheritance, i.e., compile-time evaluation. Then when an object exists, the type information for *replaceable* is of no interest any more. Finally, we can see that package class types can hold any other class types, but no other class type can hold package types.

Note that several aspects described here are our design suggestions for simplifying and making the language more stringent from a type perspective. Currently, there are no limitations for any class to contain packages in the Modelica specification. Furthermore, there are no strict distinctions between object- and class types, since elaboration and type checking are not clearly distinguished. Hence, redeclaration of elements in an object are in fact possible according to the current specification, even if it does not make sense in a class based type perspective.

4.3 Completeness of the Type Syntax

One might ask if this type definition is complete and includes all aspects of the Modelica language and the answer to that question is no. There are several aspects, such as arrays, partial and encapsulated classes, units, constrained types, conditional components and external functions that are left out on purpose.

The main reason for this work is to pinpoint the main structure of types in Modelica, not to formulate a complete type definition. As we can see from the previous sections, the type concept in the language is very complex and hard to define, due to the large number of exceptions and the informal description of the semantics and type system in the language specification.

The completeness and correctness of the allowed type prefixes described in the previous section depend on how the specification is interpreted. However, the notation and structure of the concrete type syntax should be consistent and is intended to form the basis for incorporating this improved type concept tighter into the language.

Finally, we would like to stress that defining types of a language should be done in parallel with the definition of precise semantic and type rules. Since the latter information is currently not available, the precise type definition is obviously not possible to validate.

5 Conclusion

We have in this paper given a brief overview of the concept of types and how they relate to the Modelica language. The first part of the paper described types in general, and the latter sections detailed a syntax definition of how types can be expressed for the Modelica language.

The current Modelica specification uses *Extended Backus-Naur Form* (EBNF) for specifying the syntax, but the semantics and the type system are informally described. Moreover, the Modelica language has become difficult to reason about, since it has grown to be fairly large and complex. By giving the types for part of the language we have illustrated that the type concept is complex in the Modelica language, and that it is non-trivial to extract this information from the language specification.

Consequently, we think that it is important to augment the language specification by using more formal techniques to describe the semantics and the type system. We therefore propose that a subset of Modelica should be defined, which models the core concepts of the language. This subset should describe using operational semantics including formal type rules. For some time, denotational semantics has been used as the semantic language of choice, however it has been shown to be less cumbersome to prove type soundness using operational semantics [15].

In the short term, this proposed core language is supposed to be used as basic data for better design decision-making, not as an alternative or replacement of the official Modelica specification. However, the long term goal should, in our opinion, be to describe the entire Modelica language formally.

Acknowledgments

Thanks to Thomas Schön and Kaj Nyström for many useful comments of this paper.

This research work was funded by CUGS (the Swedish National Graduate School in Computer Science), by SSF under the VISIMOD project, and by Vinnova under the NETPROG Safe and Secure Modeling and Simulation on the GRID project.

References

- [1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, New York, USA, 1996.
- [2] Luca Cardelli. Type Systems. In *The Computer Science and Engineering Handbook*, chapter 97. CRC Press, second edition, 2004.
- [3] Luca Cardelli and Peter Wegner. On Understanding Types, Data Abstraction, and Polymorphism. *ACM Comput. Surv.*, 17(4):471–523, 1985.
- [4] Dynasim. Dymola - Dynamic Modeling Laboratory with Modelica (Dynasim AB). <http://www.dynasim.se/> [Last accessed: 8 May 2006].
- [5] Peter Fritzon. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, New York, USA, 2004.
- [6] Peter Fritzon, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.
- [7] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.*, 23(3):396–450, 2001.
- [8] David Kågedal and Peter Fritzon. Generating a Modelica Compiler from Natural Semantics Specifications. In *Proceedings of the Summer Computer Simulation Conference*, 1998.
- [9] Robin Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.
- [10] John C. Mitchell and Krzysztof Apt. *Concepts in Programming Languages*. Cambridge University Press, 2003.
- [11] Modelica Association. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification Version 2.2*, February 2005. Available from: <http://www.modelica.org> [Last accessed: 29 March 2006].
- [12] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.
- [13] Alan Snyder. Encapsulation and Inheritance in Object-Oriented Programming Languages. In *OOPSLA '86: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 38–45, New York, USA, 1986. ACM Press.
- [14] Don Syme. Proving Java Type Soundness. *Lecture Notes in Computer Science*, 1523:83, 1999.
- [15] Andrew K. Wright and Matthias Felleisen. A Syntactic Approach to Type Soundness. *Information and Computation*, 115(1):38–94, 1994.

Session 3d

Electric Systems and Applications 1

Modeling and Simulation of Generator Circuit Breaker Performance

Oliver Fritz¹ and Martin Lakner²

ABB Switzerland Ltd.,

¹Corporate Research, Segelhofstrasse 1K, CH-5405 Baden-Dättwil, Switzerland

²High-Current Systems, Brown-Boveri-Strasse 5, CH-8050 Zürich, Switzerland

oliver.fritz@ch.abb.com, martin.lakner@ch.abb.com

Abstract

The authors describe a performance-evaluation tool for Generator Circuit Breakers (GCB) that assists in the process of selecting the right model and specifications from a number of customer requirements. The tool is based on a thermal-network library implemented in Modelica. A wrapper application based on Excel and .NET technology serves as user interface and driving application for the necessary simulations.

1 Introduction

Various methods are in use in order to map customer requirements to the performance of a selection of available products. Generator circuit breakers (GCB), i.e. circuit breakers installed between the generator and the step-up transformer, are available with rated power up to 2000 MVA and must be able to switch short-circuit currents up to 200 kA.

The GCB types modeled here are 3-phase systems containing a circuit breaker and a disconnecter (in series) in single phase enclosures (confer Fig. 1). Optionally the system can be equipped with auxiliary switches (earthing switches and switches to start up gas turbines), surge arrestors, voltage and current transformers (all integrated in the enclosure).

In order to find the proper current carrying capability of a specific type of GCB, particular conditions found at the customer's site have to be considered: ambient temperature, solar irradiation, temperatures of connecting parts, and special equipment (like e.g. current transformers) installed in the GCB are typical examples of such conditions. The maximum rated current of a GCB under certain operating conditions derives from the requirement that a certain limit for the temperature increase at the contacts should not be exceeded. For a GCB the maximum allowable temperature is $T_{\max} = 105^{\circ}\text{C}$ at silver-coated contacts and $T_{\max} = 70^{\circ}\text{C}$ at enclosure parts which could be

touched by an operator [1]. This ensures a high degree of reliability and a long service life since a deterioration of the materials involved is thus excluded.

Often, detailed studies (i.e. CFD or thermo-electric PDE simulations) for mapping requirements are time-consuming and find their place rather in the product-development cycle than in the sales phase.

The presented tool supporting the sales process of GCBs had to be able to produce fast and reliable results, be simple to use by non-expert personnel and still keep the necessary modularity and maintainability in order to allow for systematic refinements of the underlying model. The tools and methods chosen were therefore a thermal-network model formulated in Modelica, wrapped in an Excel-based application.



Fig. 1: Generator Circuit Breaker System

2 Thermal-network library

A Modelica library of thermal-network elements with detailed implementations of various aspects of heat-transfer physics (conduction, radiation, forced and free convection at various pressures and for a number of materials) serves as reusable base of the model. A set of full models to be used for standard performance evaluations of a complete family of GCBs represent the available product range.

The library is fully self-contained. While its basic parts are modeled along the philosophy of the Heat Transfer Library shipped with Modelica 2.2, it adds a substantial number of standardized, reusable components modeling the heat transfer for objects of the size of some centimeters to meters.

2.1 Heat sources

Main heat sources in the GCB are “Ohmic” resistors, which transform a current flow (typically between 5 and 20 kA) into heat through Joule heating. First-order nonlinearity (α) in temperature describes the dependence of the electrical resistance R for the temperature-range of interest (from ambient to about 105°C) sufficiently well:

$$R = kR_0(1 + \alpha(T - T_0))$$

The effective resistance of the current-carrying parts considers an effective increase through the (frequency-dependent) skin effect (k). For simple conductor geometries like rods or hollow cylinders k can be determined analytically. For more complex geometries (like conductor parts with attached cooling fins or the rectangular enclosure of the individual GCB poles) k was calculated with the help of FE models. For the GCB types modeled here typical values for k are in the range of 1.1 to 1.3 for frequencies between 50 and 60 Hz (the GCBs are used for 50 and 60 Hz).

Besides ohmic losses only constant heat sources are used in the model to take account of solar irradiation (in case of an outdoor installation of the GCB) as well as of losses of components like current transformers which are installed in the GCB.

2.2 Heat transfer

Heat flow by conduction is implemented for a variety of shapes and geometries. Mostly, flat geometries and surfaces are modeled; size and thickness of the parts determine their total heat-conduction properties. In the simplest case of a plate or rod (thickness d , cross section A and thermal conductivity λ) the effective thermal resistance is then given by:

$$R_{cond} = \frac{d}{\lambda A}$$

Radiation resistances model radiation-based heat transport between two plates of unequal size and unequal emissivity (gray-factor). The effective radiation resistance can be parameterized as

$$R_{rad} = \frac{1}{\alpha_r A_1} \text{ with}$$

$$\alpha_r = \varepsilon_{12} C_S \frac{T_2^4 - T_1^4}{T_2 - T_1}$$

C_S is the Stefan-Boltzmann constant. For $A_1 < A_2$ ε_{12} is given by the emissivities and areas of the plates by:

$$\varepsilon_{12} = \left(\frac{1}{\varepsilon_1} + \frac{A_1}{A_2} \left(\frac{1}{\varepsilon_2} - 1 \right) \right)^{-1}$$

Convection is implemented for air, insulating gases, and again a number of shapes and flow patterns according to effective models of fluid dynamics. The convection elements cover free and forced convection, and they require typically only experimentally available or tabulated values for parameters such as, kinematic viscosity or heat conduction λ (all as functions of temperature and pressure).

The effective resistance for free convection is modeled as:

$$R_{conv} = \frac{1}{\alpha_c A_c} \text{ with}$$

$$\alpha_c = \frac{\lambda}{l_w} c_1 (Gr Pr)^{n_1}$$

The geometry of the flow pattern is taken into account by effective length-scales (l_w) of the element-shapes. Basically, the same formula holds also for the case of forced convection: Besides an appropriate choice of the parameters c_1 and n_1 the product of Grashof and Prandl numbers in the above formula has to be replaced by the Reynolds number. The model parameters c_1 and n_1 were taken from literature with the exception of special configurations like tilted heat fin arrangements. For such configurations c_1 and n_1 have been determined experimentally.

The library is programmed in a hierarchical way and consists of several levels of partial classes in order to allow for a consistent treatment of phenomenological constants and systematic extension and specialization. All elements extend base classes with icons and descriptive text.

3 Full Models

In the stationary state the highest temperatures in a GCB are reached at the center phase of the 3-phase system since for this phase heat transfer from the enclosure via radiation is impeded by the two neighboring phases. The side walls of the center phase are facing walls with almost identical tempera-

ture (from the two outer phases) and radiation heat transfer is negligible for these parts. Thus it is sufficient to model the center phase alone, taking into account appropriate boundary conditions for the enclosure.

The model of the full circuit breaker (central phase) is composed of two to three levels of sub-elements. The variability between the individual types within the family of GCBs has been implemented fully on a parameter level. This leads to the clear advantage of keeping the topology of the full model unchanged while allowing simulations for the whole product range by simply changing some input parameters at simulation time.

Basically, all model classes extend a (unevaluated) parameter-class. Those parameters that differ from model to model are stored in arrays, selecting the model index leads to a full propagation of all parameters to the individual sub-models. As the wrapper application will only set parameters and initial values via the `dsin.txt` file governing each simulation, parameters must all be evaluated at run-time only.

The topology of the thermal network representing the GCB is based on earlier work and new experience. It has been unified and newly arranged making use of the practice of avoiding redundancy and inconsistency through a systematic identification of similarities and structural components. Still, a physical picture of the GCB was kept in order to make the model not too abstract and in consequence too hard to maintain.

As an example Fig. 2 shows the breaking chamber of a GCB. It consists of two metallic castings (including the connection zones to neighboring system components) the insulator (white) and the contact system (invisible underneath the insulator). Each component (castings, insulator, and contact zone) is modeled as sub-element of the breaking chamber in which GCB-type specific features like the cooling fins are controlled by parameters.

The main structure of sub-elements on the lowest hierarchy level of the model is shown in Fig. 3 representing a section of the main conductor. The structure contains a heat source (P_{TR}) which obtains the current flowing in the conductor via the electrical pins p and n , thermal conduction resistances whose “outer” heat ports are connected on the next hierarchy level to the neighboring sections, a convection (R_{conv}), and a radiation resistance (R_{rad}). The latter elements transfer heat to the air surrounding the section and to the enclosure, respectively (via the corresponding heat ports of the sub-element).

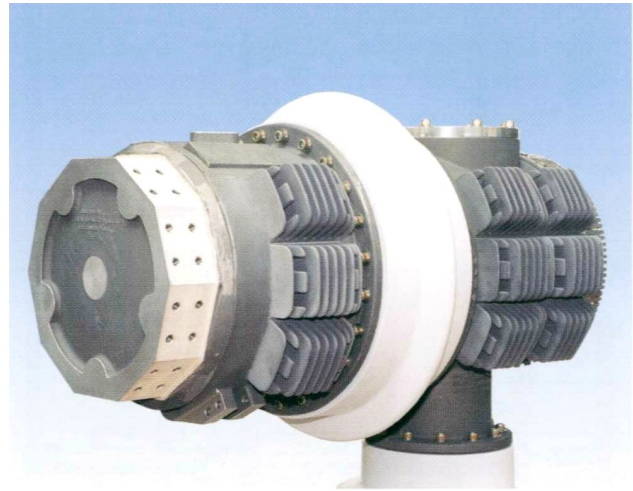


Fig. 2: Breaking chamber of a GCB. The type shown is equipped with cooling fins to enhance the heat transfer to the ambient air.

A straight forward and simple extension of the model would be to add a heat capacity element to the central node in Fig. 3. This would allow to perform dynamic simulations, e.g. to evaluate the influence of temporary overloads (changing load currents). However, at the present state the model is intended to determine the stationary state only.

The final models consist to a high degree of nonlinear systems of equations. They are almost exclusively algebraic in nature, as the steady-state solution of a model is of main interest.

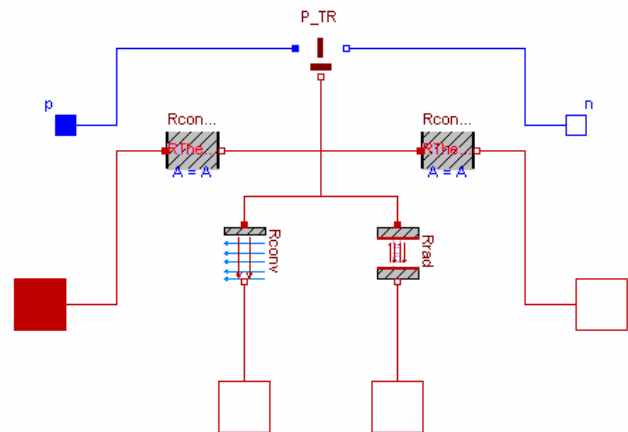


Fig. 3: Thermal network for a section of the conductor path of a GCB.

4 Wrapper application

Interface libraries developed in MS.NET and VBA form the components of an Excel-based application with a simple top-level user interface. A form for input and triggering simulations is presented to the user (Fig. 4). A number of additional tables and charts are used for presenting the results.

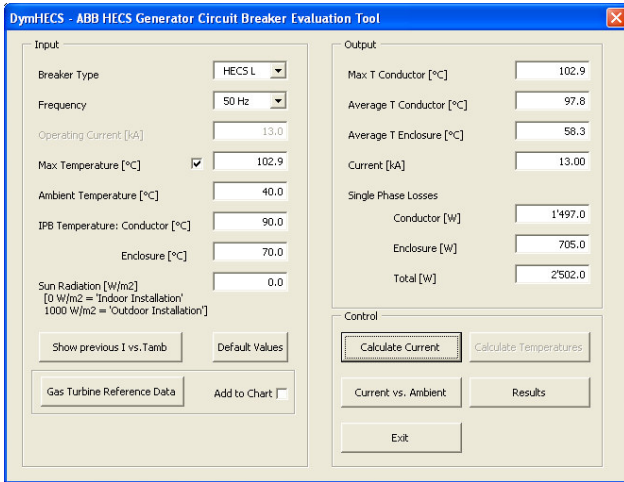


Fig. 4: Front end of the user-interface. A limited number of parameters and simulation options are presented to the user.

Fig. 5 shows a component view of the full application and its main task flow. As a result of the modeling, a simulation server (dymosim.exe) is produced and packaged with the application. An interface library (dsaccess.dll), implemented in C, serves as proxy between Excel and the simulation server.

While the user defines parameters and nature of a desired evaluation fully in Excel, the definition file (dsin.txt) as well as the results of the simulation run (dsout.txt) are written and read with the use of routines contained in the proxy library. This task flow is completely transparent to the user; the user does not have to interfere with the actual formats and types of input and output of the simulation at all.

A typical evaluation is done within a few seconds. The Excel application is implemented as a template, such that results and used cases can be saved and restored in a natural way.

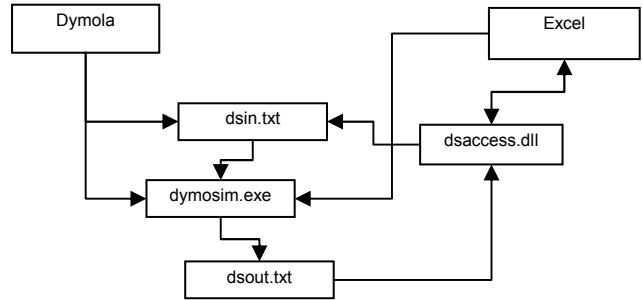


Fig. 5: Component view of the full application.

5 Results and Conclusions

Fig. 6 shows the cross section of a GCB that was used in a thermal type test at the ABB high current test laboratory. The enclosure of the circuit breaker pole (surrounding rectangular box) is only indicated. The resulting temperature profile along the axis of the circuit breaker (inner conductor) of the thermal type tests are shown in Fig. 7 as well as the simulation result.

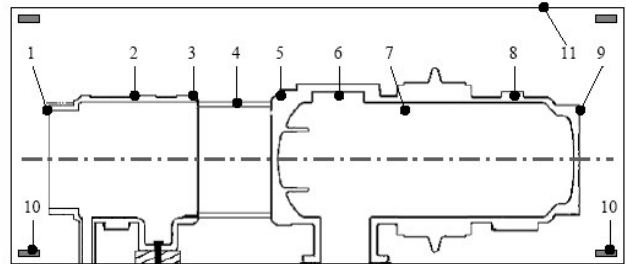


Fig. 6: Schematic cross section of a GCB. The numbered dots indicate positions of temperature sensors in temperature rise tests.

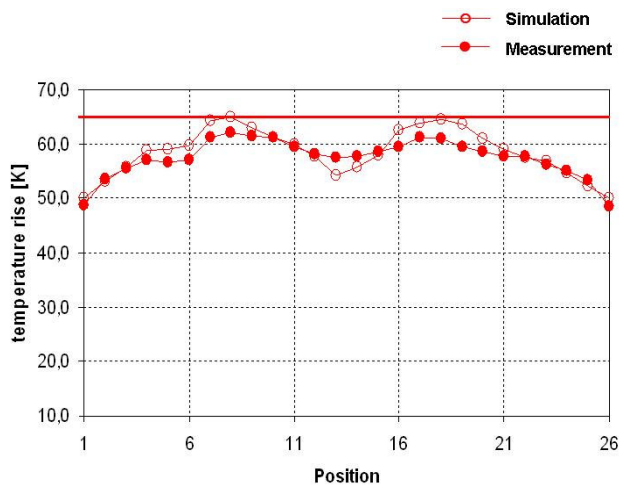


Fig. 7: Temperature rise above ambient temperature vs. position along the conductor of a GCB. The red line indicates the maximum permissible temperature rise.

Fig. 7 shows that the calculated temperature distribution agrees very well with the measured temperatures. The maximum deviation is approximately 4 K at a temperature rise of >60 K (7%) at the contact zones. The reason that the model yields conservative results is mainly due to slightly pessimistic values used in the model for the material parameters and the contact resistances. The agreement between measurement and simulation obtained for all other GCB types covered by the thermal network model was also better than 10%.

For benchmarking the results of the simulations to previously used methods, a great number of standard experiments were defined and extensively tested. In particular, the identification of the temperature of hot spots in the GCB under a given current load and the inverse problem, i.e., the identification of a maximum load current while staying below a given acceptable operating temperature were performed. Finally, the whole application (without the modeling environment) was packaged and rolled out, the productive phase started roughly half a year after the project definition.

While the implementation of the necessary proxy library and VBA code presented no major challenges, a particular difficulty of the model refinement was the choice of stable guess-values for the substantial amount non-linear equations. Through a restriction of the empirical parts of certain model classes to physically meaningful value ranges, a sufficient reliability of the initial-value calculation could be achieved.

The authors conclude that a thermal-network based application is an industrially productive solution for a performance-evaluation tool with an underlying model of high complexity and accuracy. The case for such a tool is underpinned by substantial savings on license costs, the use of standard tools requiring no training, and the competitive advantage of a substantially reduced time to offer.

References

- [1] IEEE C37.013-1997: IEEE Standard for AC High-Voltage Generator Circuit Breakers Rated on a Symmetrical Current Basis

Parallel Simulation with Transmission Lines in Modelica

Kaj Nyström Peter Fritzson

Dept. of Computer and Information Science, Linköping University

SE-581 83 Linköping, Sweden

kajny@ida.liu.se petfr@ida.liu.se

Abstract

Parallelization of simulations has traditionally been an important way of improving the performance of complex simulations. However, this often requires knowledge in parallel programming, something few modellers have. In this paper we present a way of parallelizing Modelica simulations at the component level requiring no prior knowledge in parallel programming. Our method of parallelizing simulations uses the equation based and unconditionally stable Transmission Line Modeling technique which uses simple time delays to decouple a model into submodels. The method is independent of compiler implementation and thus supports all of the Modelica language supported by a given Modelica compiler. An evaluation of our implementation of this method shows speedups of up to 2.3 times with a variation in speedup that is highly dependent on the model structure and how successful the users parallelization is.

Keywords: TLM; parallel; simulation; transmission line; modeling;

1 Introduction

As knowledge in modeling and simulation becomes more common throughout both industry and academia, the need to simulate systems with higher complexity grows stronger. However, computational power effectively sets the upper limit for how complex our models can be before simplifications have to be made in order for the simulation to finish in reasonable time.

Traditionally one of the most common ways of achieving better performance from a simulation has been to parallelize it. While this is usually difficult for a simulation in a low level language, for example Fortran or C, it is even harder in the

Modelica language [4, 5, 6] since the Modelica user has little control over the inner workings of a simulation, something that is often necessary in order to parallelize it. In addition, parallelization of a simulation almost always requires expert knowledge in the area of parallel programming, something that few Modelica users have.

One possibility to simulate in parallel would be to use parallel solvers. These parallel solvers are however not suitable for all problems and can sometimes suffer from numerical instability. Another solution is to automatically parallelize the simulation, either at the Modelica level or at the generated code level. This typically gives better performance for some tightly coupled simulations. However, available tools can not handle for example hybrid models and performance increase could possibly be better if the user can help the modeling environment with the parallelization in some way.

The problem with user interaction when parallelizing a model in Modelica is that user interaction has to be done on a level that the user can access and understand. The typical Modelica user works on the component level. Thus, this is where the parallelization should be specified. It should also be in an application domain neutral fashion, since that is how the Modelica core language is intended to be used. Parallelizing a mechanical application should ideally be no different from parallelizing an electrical application.

2 Contributions

In this paper, we present a domain neutral and numerically stable method of parallelizing Modelica simulations at the component level requiring virtually no knowledge in parallel programming from the user. We base our method on the Transmission Line Modeling theory.

3 Transmission Line Modeling

Central to the task of parallelizing a model is the task of partitioning the model into submodels which can then be simulated on separate computers. We have chosen the Transmission Line Modeling method for parallelizing simulations for a number of reasons. It is a proven way of decoupling equation systems and is also equation based itself which makes it fit nicely into a Modelica component. Furthermore, the TLM method has been proven to be unconditionally stable, an almost absolute requirement as an unstable simulation can be close to useless. This stability holds for as long as the TLM parameters are within physical boundaries.

The theory evolved from the Telegraphers Equations [18] which concerns signal propagation in cables. In the 1970's the TLM method was first used for computer based modeling by among others A. Fettweiss[11] and P.B. Johns[12]. The method has previously been used to decouple and solve previously unsolvable problems, for co-simulation and to some extent also for distributed simulation.

3.1 TLM Theory

The idea behind the TLM technique is to use physically motivated delays in signal propagation media to decouple a simulation. This time, called T_{ilm} is the time it takes for signals from system 1 to reach system 2 (see figure 1).

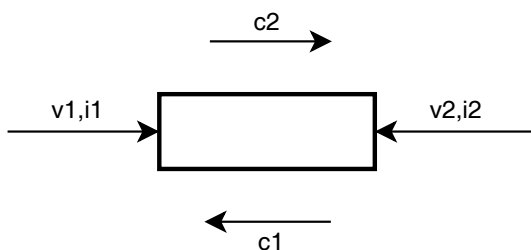


Figure 1: A TLM connection and the governing variables. Notation is from the electrical domain, voltages v_1, v_2 and currents i_1, i_2 . c_1 and c_2 are the characteristics of the transmission line .

T_{ilm} can be computed from the signal propagation speed in the medium and the medium length. The equations which govern the exchange of infor-

mation between the systems are

$$c_1(t) = V_2(t - T_{ilm}) + Z_F I_2(t - T_{ilm}) \quad (1)$$

$$c_2(t) = V_1(t - T_{ilm}) + Z_F I_1(t - T_{ilm}) \quad (2)$$

$$P_1(t) = Z_F I_1(t) + c_1(t) \quad (3)$$

$$P_2(t) = Z_F I_2(t) + c_2(t) \quad (4)$$

The parameters c_1 and c_2 are called the characteristics of the transmission line and represents the propagated information in every time step, delayed as the theory prescribes. P and Q are the variables in the TLM connection and could be from any domain, for example current and voltage. Z_F is an implicit impedance for the connection.

The advantage with introducing TLM connection is that the previously implicit parallelization problem now becomes explicit. Consider the equations 1 and 3. These equations state that P_1 at time t only depends on system 1 and on previous $(t - T_{ilm})$ values from system 2. Thus we have transformed the problem of solving one implicit equation system into two smaller implicit equation systems. The possibility for parallel processing is obvious.

An additional advantage is that we can use both different solvers and different time steps in the two subsystems, as long as we interpolate propagated values reasonably well if needed. This means that we can greatly reduce the stiffness for some problems.

The TLM method has been proved to be unconditionally stable [14, 16], provided that the TLM parameters are computed correctly. The method does not introduce any additional numerical error into the model. Instead, it actually transforms a numerical error to a modeling error[13]. This often makes it easier for a user to identify and compensate for the error rather than if the error would be purely numerical due to the fact that the normal Modelica user is probably a modeling expert rather than a numerical expert.

3.2 Theory Extensions

The TLM theory prescribes that we compute the transmission line delay time T_{ilm} from the propagation speed and the length of the line. However, in order to maximize the degree of decoupling and achieve maximum speedup, we can allow for non-physical T_{ilm} . This is useful since we do not want any stalling in the simulation of the subsystems due to lack of data. This stalling can happen if

it takes too long for the data to be transmitted through the computer network from the computer which simulates system 1 to the computer simulating system 2. If we increase T_{ilm} to a value greater than its corresponding computed value, we allow for higher latency which will avoid stalling computations. Such an increase in T_{ilm} is however not without problems. If we increase T_{ilm} too much, the system might become unstable and/or produce wrong results. It is not easy to give an answer on what a good value for T_{ilm} is when you move beyond the strictly physical value as choosing a good T_{ilm} is a trade off between performance and robustness and depends on system dynamics.

The TLM theory was originally developed for electromagnetical signals. Over the years it has been used for a wide variety of domains (hydraulics, mechanics etc). However, we see no reason to limit ourselves to any specific domains since the Modelica language gives us such exceptional possibilities for building generic components.

This method of parallelization should work for most domains in one dimension which propagate one flow and one non-flow variable. Extending the TLM theory to use vectors has been investigated previously[16, 13] in a somewhat different context and we foresee no problem with extending our method to handle different sets of propagated variables.

Since the transmission line has an undamped resonance, it is sometimes beneficial [15] to low pass filter c_i in a transmission line as

$$c_i(t) = \alpha c_i(t - \delta)$$

where α is the filter parameter (0.2 is usually a good value) and δ is the time step. Without this filtering, the resulting signals might contain unwanted high frequency components, resulting in a slightly staircase shaped signal as can be seen in figure 6

4 Implementation

We have implemented and tested our way of parallelization of Modelica simulations with the TLM method as outlined in section 2. The framework consists of 3 parts which we shall now describe in detail.

4.1 A Generic Modelica TLM package

The TLM package (depicted in figure 2) contains the TLM components which the user inserts into his model when he wants to partition it. More on how this is done in section 5.1. The package also contains external functions which take care of the message passing in the simulation. However, the user never needs to see or use these functions.

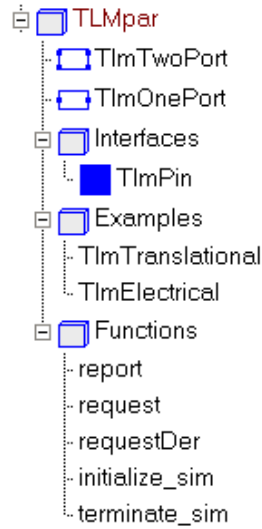


Figure 2: Structure of the TLM package

4.2 The Model Partitioner

This small program transforms the original model into a new Modelica package which in turn consists of the separate submodels derived from the original model. The program also divides and propagates the TLM components so that the correct parts of it are present in all submodels. This partitioning is depicted in figure 3.

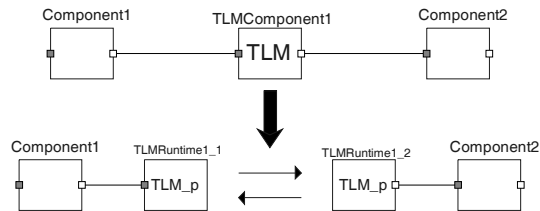


Figure 3: Splitting one model into two models on TLM boundaries

The arrows between $TLM1_1$ and $TLM1_2$ symbolize the communication of TLM variables between the two submodels, which takes place over a local network using a simple sockets-based protocol.

Algorithm 1: Partition a model into submodels

Input: An arbitrary component based model

Output: A mapping component-submodel number for all components

```

1 subModelNr ← 0
2 foreach component ∈ components do
3   if notvisited(component) then
4     visited(component) ← true
5     component(component) ← subModelNr
6     push(componentStack,
7         allNeighbours(component))
8     while not empty(componentStack) do
9       component ← pop(componentStack)
10      visited(component) ← true
11      component(component) ←
12        subModelNr
13      push(componentStack,
14          allNeighbours(component))
15    end
16  end
17  subModelNr ← subModelNr + 1
18 end

```

The partitioning is done using a repeated breadth-first search with visitor recognition as described in algorithm 1. TLM-components and associated connect equations are filtered out before the algorithm is applied to the model as they should be considered as separators (interfaces) between submodels.

After applying this algorithm, all components have an association to a submodel number. We can now insert all components present in the original model in their respective submodels. Next, we check into which submodel the first component in each connect equation belongs and use this information to add the connect equation in the components. For example, if component R1 belongs in submodel 2, the connect equation `connect (R1.p, C1.n)` should be entered in submodel 2.

The previously filtered out TLM components are now substituted for runtime TLM components as in figure 3. These runtime TLM components contain all necessary functionality for requesting and reporting information necessary for the simulation to their counterpart TLM runtime component.

4.3 A Simulation Dispatcher and Manager

When the partitioning is done, the simulation is built using any Modelica compiler. We have successfully used OpenModelica[1, 2, 3] and Dymola[17]. As far as the compiler is concerned, it is now compiling two or more completely different models with no association between them, so the compiler itself needs no modification.

This gives us some additional advantages. During compilation of our submodels we can customize the simulation of our different submodels, for example choosing different solvers for different submodels if desired. We can also specify different time steps or fault tolerance levels which if done right can significantly reduce the stiffness in the original model.

Finally, the dispatcher takes care of distributing the jobs on separate machines, such as on a computational grid or a PC cluster, and to manage reports and request for data from the simulations.

5 Discussion and Results

In this section we will present and discuss our results with respect to user interaction, performance and fault tolerance.

5.1 User Interaction

One of our primary goals with the work presented in this paper is to provide a way of parallelizing simulation that the average Modelica user can actually use without too much effort. This means that it should require little change in the way the user builds his model. At the time of writing, no scientific study has been done on the usability of this framework so we will settle for briefly describing what has to be done by the user in order to parallelize his model and let the reader decide whether this is usable or not.

The only additional task the user has to undertake in order to use our framework is to partition his model by inserting TLM elements where he wants to partition his model. This can be done either graphically or textually and works exactly like inserting any normal Modelica component in a model. The hardest part for the user is to determine *where* to insert these TLM elements.

The best and most general advice we can give at a model level is to decouple the model at domain

boundaries since these are usually the boundaries between fast and slow subsystems. Decoupling such subsystems combined with using different solver settings can significantly increase performance. Another advice is to partition the model in equally complex parts. This is quite difficult to do at a component level since it may be hard to see at a component diagram level what the complexity is of a certain part of a model.

5.2 Performance

Evaluation of a parallel programming framework is difficult at best. Many factors are involved and the framework designer tends to choose the problems and environments that favour his framework the most. When evaluating our framework for Modelica models, we have found that our largest problem by far is to choose our models. Few sufficiently large models are available to the general public, especially models which can be understood and parallelized by a non-expert in the modeling domain.

The best thing would obviously be to have a set of more or less standardized and independent benchmarking models. Lacking this, we have chosen to build our own models for benchmarking using only Modelica Standard Library components and examples. Given this bias problem bias, it is uncertain if we should really present any figures of speedup at all before our framework has been tested with independently built models. Even so, we choose to present our preliminary findings regarding performance here for what they are.

The framework can handle Modelica models of arbitrary size but as usually is the case with parallelization, little or no speedup or even a performance decrease can be expected when parallelizing small models as the communication overhead then becomes a significant factor in simulation time. Then again, there is probably no need to parallelize small models as these will most likely run just fine on a single CPU.

Using these models we have registered speedups ranging from 0.5 (negative speedup) to 2.3 depending on the structure of the problem and how we parallelize it. Stiff problems which we can easily decouple will give us the greatest speedup while some homogeneous problems might not be suitable for parallelization at all.

All tests were done on a standard PC-cluster with the following nodes

- OS: Rocks Linux [19]
- CPU: PIII-800Mhz
- Memory: 512MB
- Network: 100Mbit Ethernet

The cluster is quite old fashioned but still demonstrates the general effectiveness of our framework. It is likely that a more modern cluster with faster nodes and faster network will increase performance as we can then decrease the granularity of our model partitioning.

Our figures have been derived by comparing total simulation time on one node to total simulation time using two to six nodes, depending on the model structure. We wish to stress that we do not rule out the possibility that a modeling expert could achieve better performance as he or she might be better suited to parallelize the models.

5.3 Fault Tolerance

Just as the theory predicts, the error in the models we have tested is well within normal values for numerical simulations as long as the TLM parameters are within physical limits. When we go outside physical values however, we will inevitably introduce an error. How large or significant the error is depends on the model. The modeler is obviously best suited to be the judge of if this is within his fault tolerance limits or not. As we are just working with a simple delay in the time domain it is generally easy for a domain expert to see beforehand what effect an extra time delay will have on his model and if this is acceptable or not.

For comparison on what a way too large T_{tlm} , we present two simulation runs of a standard DC-Motor example with a ramp as a voltage source as depicted in figure 4.

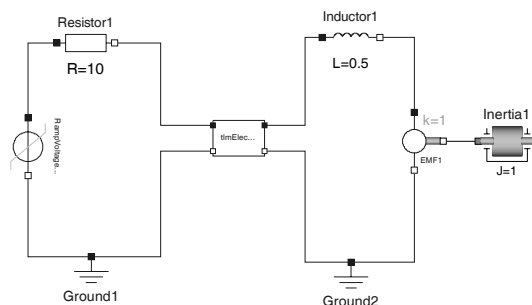


Figure 4: TLM partitioned DC-motor model

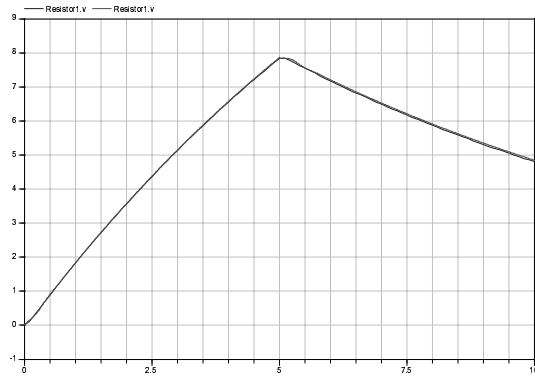


Figure 5: Plot of voltage over the resistor component, $T_{tlm}=0.01$, interval length=0.01s



Figure 6: Plot of voltage over the resistor component, $T_{tlm}=0.1$, interval length=0.01s

From figures 5 and 6, we can clearly see that $T_{tlm} = 0.1$ was probably a too high value for most applications, although it still does show the general shape of the result. Still, since it is a delay in the electrical application domain in a circuit where the propagation speed is usually very large and in a model with no capacitor elements, delaying the signal by one tenth of a second seems rather a lot to any electrical engineer. As always, the modeler must use his judgement when setting the parameter values, in this case T_{tlm} .

6 Conclusions

We have been able to parallelize Modelica simulations and to abstract user interaction at a component level which we believe will be a usability improvement compared to other parallelization techniques. Communication and scheduling are completely hidden from the user. However, no scientific evaluation has yet been done on the usability aspects of our work.

Speedup is up to 2.3 times so far but varies

greatly and depends on model structure and if the model is partitioned successfully. Performance also depends on accuracy requirements on the model and is easily configurable by the user. Additional advantages with the method are that it reduces model stiffness if properly used and that it is also possible to use different solvers for different submodels if desired.

7 Future work

Obviously, a usability study is one of the most important items for future work as that has been one of the major goals of our work. We would at the same time like to continue to develop heuristics for better partitioning of models. This process might even be automated using such heuristics. There is also plenty of more work on automatic estimation of non-physical delays that do not lead to errors beyond a given tolerance level, perhaps using static analysis on the model.

A better performance evaluation is also prioritized but largely dependent on the availability of large models. Such models have proven to be difficult to find.

On the implementation side, a better communication implementation pattern (e.g. peer to peer) should be established in order to reduce communication cost. Also, adaptable value reuse depending on model dynamics should not be hard to implement and should lead to a significant decrease in communication overhead.

More static and dynamic analysis of the performance bottlenecks for individual simulations could be a way of aiding the user in both model partitioning and choosing TLM parameters.

8 Acknowledgments

This work was supported by MathCore Engineering[9] and Vinnova[8] in the GRID-Modelica project[10].

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, 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*, Issue 44/45, December 2005.
- [3] The OpenModelica Users Guide, version 1.3.2, Apr 2006. <http://www.ida.liu.se/projects/OpenModelica>
- [4] The Modelica Association. The Modelica Language Specification Version 2.2, March 2005. <http://www.modelica.org>. accessed 2005-05-02
- [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] Michael Tiller. Introduction to Physical Modeling with Modelica. 366 pages. ISBN 0-7923-7367-7, Kluwer Academic Publishers, 2001.
- [7] Peter Aronsson and Peter Fritzson, Task Merging and Replication using Graph Rewriting, Tenth International Workshop on Compilers for Parallel Computers, Amsterdam Netherlands, Jan 8-10 2003
- [8] Vinnova, <http://www.vinnova.se>, accessed 2005-05-02
- [9] Mathcore Engineering, <http://www.mathcore.com>, accessed 2005-05-02
- [10] The GridModelica Project, <http://www.ida.liu.se/labs/pelab/modelica/GridModelica.html>, accessed 2005-05-02
- [11] A. Fettweiss, Digital Filter Structures Related to Classical Filter Networks. *Arch. Elek. Übertragungst.*, 23(2):79-89, 1971
- [12] P.B. Johns and M.A. Brien, Use of the Transmission Line Modeling (t.l.m.) Method to Solve Non-Linear Lumped Networks, *The Radio Electron and Engineer*, 50:59-70, Jan/Feb 1980
- [13] Petter Krus, Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints, *ASME journal of Dynamic Systems, Measurements and Control*, 1995
- [14] S.H. Pulko, A. Mallik, R. Allen, and P.B. Johns. Automatic Timestepping in TLM Routines for the Modelling of Thermal Diffusion Processes. *Int. Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 3:127-136, 1990.
- [15] P.Krus, A. Jansson, J-O. Palmberg and K. Weddfelt. Distributed simulation of hydromechanical systems. In *Third Bath International Fluid Power Workshop*, Bath, UK, 1990.
- [16] Iakov Nakhimovski, Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis. Ph.D Thesis, Linköping University, Dept. of Computer and Information Science, April 2006.
- [17] The Dymola modeling tool, <http://www.dynasim.com> accessed 2005-05-02
- [18] The telegraphers equations, http://en.wikipedia.org/wiki/Transmission_line#Telegrapher.27s_equations, accessed 2005-05-02
- [19] Rocks Linux, <http://www.rocksclusters.org>, accessed 2005-07-20

