MODELICA

2008

Proceedings of the 6th International Modelica Conference

March 3rd – 4th, 2008
University of Applied Sciences Bielefeld,
Bielefeld, Germany

Bernhard Bachmann (editor)

Volume 2

organized by
The Modelica Association and
University of Applied Sciences Bielefeld

All papers of this conference can be downloaded from
http://www.Modelica.org/events/modelica2008/

Proceedings of Modelica' 2008
University of Applied Sciences Bielefeld,
Bielefeld, Germany, March 2008
Editor:
Prof. Dr. B. Bachmann
Published by:
The Modelica Association (http://www.Modelica.org) and
University of Applied Sciences Bielefeld (http://www.fh-bielefeld.de)
Printed by:
Printmedien Elbracht

Preface

The first International Modelica Conference took place in October 2000 in Lund, Sweden. Since then, Modelica has increasingly become the preferred language tool for physical modelling of complex systems. This is indicated by the high number of registrations from industry and science at the 6^{th} International Modelica Conference held between 3^{rd} and 4^{th} March 2008 at the University of Applied Sciences, Bielefeld, Germany. It is also indicated by the number of excellent papers submitted to the program committee which made the task of selecting papers for oral and poster presentation very difficult and, last but not least, by the exhibition during the conference at which several companies will be represented. This volume contains the papers of the 68 oral presentations and 14 poster presentations at the conference. The ability of Modelica as a multi domain simulation language is demonstrated impressively by the various fields the papers are covering.

Due to the special features of the Modelica language, such as object-oriented modelling and the ability to reuse and exchange models, Modelica strongly supports an integrated engineering design process. Thus in various fields Modelica has become the standard tool for model exchange between suppliers and OEM's. A key issue for the success of Modelica is the continuous development of the Modelica language as well as the Modelica Standard Library under strict observance of compatibility to previous versions by the Modelica Association. 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 6^{th} International Modelica conference was organized by the Modelica Association and by the University of Applied Sciences, Bielefeld, Germany. I 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. Together with the entire team of the local organizing committee I would like to wish all participants an excellent and fruitful conference.

Bielefeld, March 1^{st}, 2008

Bernhard Bachmann

Program Chair

- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Bielefeld, Germany

Program Board

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

Program Committee

- Prof. Karl-Erik Årzén, Lund University, Lund, Sweden
- Dr. John Batteh, Emmeskay Inc., Michigan, USA
- Dr. Ingrid Bausch-Gall, Bausch-Gall GmbH, Munich, Germany
- Daniel Bouskela, EDF, Paris, France
- Prof. Felix Breitenecker, University of Technology, Vienna, Austria
- Dr. Thomas Christ, BMW, Michigan, USA
- Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
- Prof. François E. Cellier, ETH Zürich, Zürich, Switzerland
- Mike Dempsey, Claytex Services Limited, Leamington, U.K.
- Denis Fargeton, LMS Imagine, Roanne, France
- Dr. Rüdiger Franke, ABB, Heidelberg, Germany
- Rui Gao, Dassault Systèmes K.K., Nagoya, Japan
- Anton Haumer, Technical Consulting, Vienna, Austria
- Dr. Christian Kral, arsenal research, Vienna, Austria
- Gerard Lecina, Dassault Systèmes, Paris, France
- Dirk Limperich, Daimler AG, Sindelfingen, Germany
- Kilian Link, Siemens AG, Erlangen, Germany
- Dr. Jakob Mauss, QTronic GmbH, Berlin, Germany
- Dr. Ramine Nikoukhah, INRIA, Rocquencourt, France

Local Organizing Committee

Contents

Index of Authors

Session 4a

Language, Tools and Algorithms

Frequency-Domain Analysis Methods for Modelica Models

Andreas Abel Tobias Nähring

ITI GmbH
Webergasse 1
01067 Dresden, Germany

{andreas.abel,tobias.naehring}@iti.de

Abstract

In addition to time-domain simulation methods, engineers from different application fields require further types of analysis to be performed on their systems. In particular results from frequency domain analysis play an important role – this includes the calculation of natural frequencies and vibration modes, but also the computation of transfer functions or the simulation of steady-state behaviour.

If the system equations are formulated using the Modelica language, there is the potential to use one and the same model for time-domain as well as frequency-domain computations.

In this paper we will show, how the different methods can be applied to a Modelica model, what kind of prerequisites and adjustments are required in order to perform the different types of analysis and how these methods can be seamlessly integrated into a Modelica simulation environment.

Keywords: Modelica, Steady State Simulation, Transfer Function Analysis, Natural Frequency Analysis

1 Introduction

In many engineering disciplines frequency-domain methods play an important role. Powertrain engineers for instance not only exploit transient simulations, but to a large extend assess the behaviour of their systems based on the natural frequencies, the resulting vibration models, and also in terms of steady state results, which show vibrations under stationary conditions resulting from the uneven and multi-order excitation of the driveline by the engine. Other engineering domains and tasks also require frequency-domain approaches.

However, all these tasks would typically be assigned to different software tools, which is not really necessary.

Modelica forms the ideal base also for frequency-domain analyses, since it provides complete system descriptions in an analytic form. However, so far Modelica is used almost exclusively for transient time-domain simulation.

In this paper we will show, how Modelica models are used in order to compute frequency-domain results and how these processes are integrated into the Modelica simulation environment SimulationX.

The paper will treat the following topics:

- Nonlinear periodic steady-state simulation and generation of spectral results based on harmonic balance
- Natural frequencies, vibration modes and energy distributions based on models linearized in an operating point
- Computation of transfer functions based on models linearized in an operating point

We focus on the periodic steady-state simulation since this is the most recent innovation in SimulationX.

2 Periodic Steady-State Simulation

2.1 Application to Modelica Models

The main area of application for the nonlinear periodic steady-state simulation in SimulationX is the vibration analysis of powertrains.

The example Modelica model in Fig. 1 is an adaption from [4] p. 246 with some added damping and cylinders including oscillating masses and driven by some typical combustion engine cylinder pressure.

The steady-state for a range of mean rotational speeds of the engine has to be computed. The oscillation time period is determined by the engine speed and the periodicity of the excitation over the crank angle.

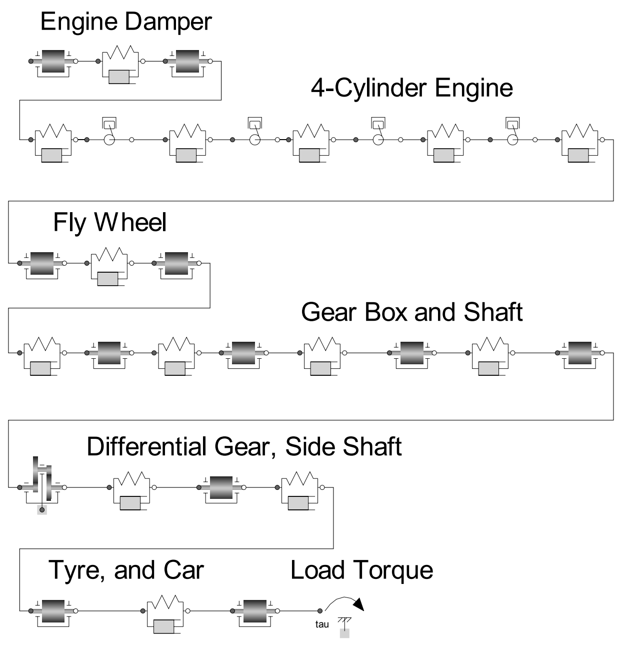


Fig. 1: Example of a Modelica model of a powertrain analyzed with the periodic steady-state simulation

But the method is also applicable to driven systems in other physical domains. For non-linear electronic amplifiers and filters most often the frequency or amplitude of the driving generator is swept and the period is measured at its phase. Therefore, a general approach is needed. In SimulationX the following procedure has been implemented: The user chooses the varying reference quantity (e.g. mean engine speed or generator frequency) and the period variable (e.g. crank angle or generator phase) from Modelica model trees containing all variables and parameters. For powertrains (or more general whenever the reference quantity is not a parameter but the mean value of a variable) the user also distinguishes some model parameter as compensation parameter - such as the load torque of the powertrain. The algorithm adjusts the compensation parameter for the steady-state, i.e. the mean engine torque and the load torque are balanced out by the algorithm. No special preparation of the Modelica model is needed to enable the steady-state simulation. The same model may be used for a simulation in time-domain too.

During the simulation the computed spectra of the Modelica variables are written to special steady-state protocols. Those results can be visualized in several different representations (amplitudes, phases, fluctuations, spectral powers and so on). For the powertrain example from Fig. 1 some of the amplitudes of the calculated harmonic torque components in the mass-damper spring are shown in Fig. 2.

In SimulationX the initial conditions corresponding to the results of the periodic steady-state simulation can be calculated and used to initialize a successive transient simulation.

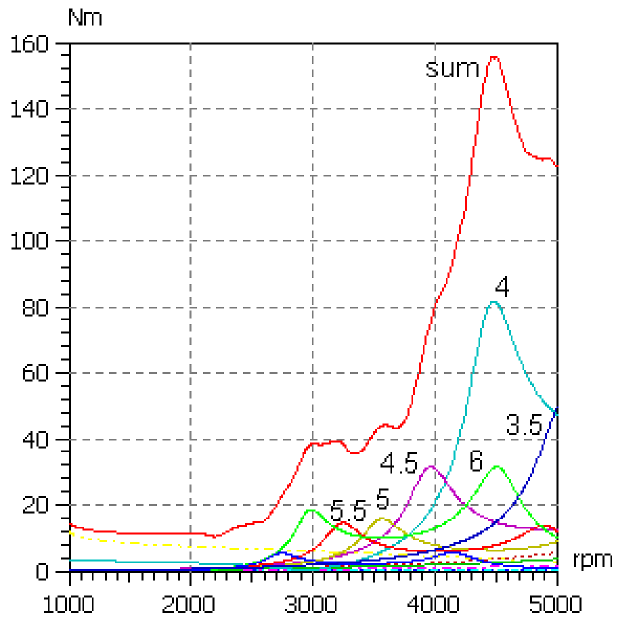


Fig. 2: Spectral results for the torque of the spring in the engine damper; the sum curve and the amplitudes of the first harmonic components are shown, the largest amplitudes are labelled with the oscillation orders

In this way the periodic solution can be recalculated with a transient simulation and the steady-state results can easily be checked. Fig. 3 shows a very good match of the steady-state simulation result with the transient simulation result.

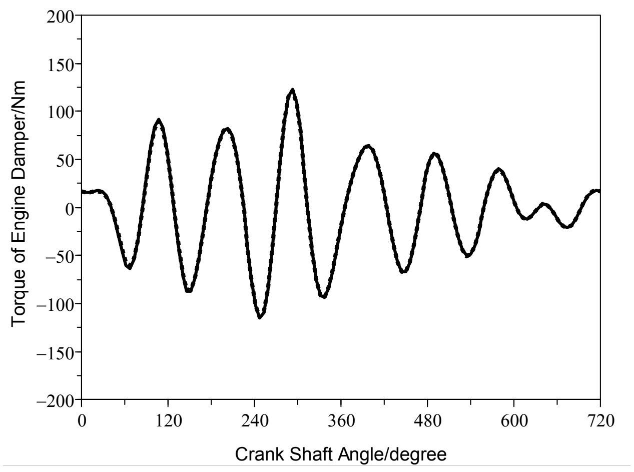


Fig. 3: Torque in the spring of the engine damper (full line: steady-state simulation, dashed line: transient simulation); the results are almost identical

2.2 Computational Background

In this subsection we give some insight in the computational background specific to the periodic steady-state simulation. If the reader is only interested in applications he may safely skip to subsection 2.3.

For the periodic steady-state simulation the harmonic balance method is employed. This method gives a high spectral precision of the results and prepares the numerical base for behavioural modelling in the frequency domain.

2.2.1 System Equations

The symbolic analysis compiles from the Modelica model a system of equations for the stationary simulation. If the simulation time appears explicitly in the model equations (for instance in a driven system) it is replaced by a state x_{time} with $dx_{\mathsf{time}} / dt = 1$ which leaves us with an autonomous algebraic differential equation system

$$f\left(x(t), \dot{x}(t), x_{\mathsf{C}}\right) = 0 \tag{1}$$

where x is the $\mathbf{R}^n$-valued state vector with corresponding time-derivative $\dot{x}$, and $x_{\mathsf{C}} \in \mathbf{R}$ is the compensation parameter (see section 2.1). It is convenient to represent oscillations not over time but over the phase angle $\varphi := \omega t$ for which the period length keeps constant at 2π independent of the period duration (ω is the phase velocity of the oscillation). Substituting the derivative w.r.t. time through the derivative w.r.t. phase $\dot{x}(t) = \omega\, x'(\varphi)$ in eq. (1) gives

$$f\left(x(\varphi), \omega\, x'(\varphi), x_{\mathsf{C}}\right) = 0\,. \tag{2}$$

Throughout the remainder of this section we represent x in dependence of the phase angle.

The system is assumed to be freely displaceable in one direction of the state space. Therefore, we chose a combination of a 2π-periodic function $\tilde{x}$ and a component linearly dependent on the phase angle as a solution ansatz

$$x(\varphi) = \frac{x_{\mathsf{P}}\varphi}{2\pi} + \tilde{x}(\varphi) \tag{3}$$

for the system equation (2) with a constant vector $x_{\mathsf{P}} \in \mathbf{R}^n$, called period vector in the sequel.

This setup is rather general. It includes freely rotating powertrains and periodically driven systems.

Solving (2) can now be divided into the two tasks

- computation of the period vector x_{P}
- computation of the periodic function $\tilde{x}$

which will be described in the following two sections.

2.2.2 Period Vector Computation

The user selects one model variable as the period variable (cf. section 2.1). We denote the index of that variable as $i\mathsf{P}$. For this variable the user specifies the period length p. The model equations (2) are then solved for the static case (i.e. $\omega = 0$) once with $\varphi = 0$ and once with $\varphi = 2\pi$. Because of the 2π-periodicity of $\tilde{x}$ the difference of these two solutions just gives the period vector

$$x_{\mathsf{P}} = x(2\pi) - x(0). \tag{4}$$

At $\varphi = 0$ the displacement of the system (e.g. the rotational position of a powertrain) is determined by the additional condition $x_{i\mathsf{P}}(0) = 0$. This together with (2) and (3) results in the overall system

$$f\left(x(0), 0, x_{\mathsf{C}}\right) = 0;\ x_{i\mathsf{P}}(0) = 0 \tag{5}$$

for the case $\varphi = 0$ which consists of $n+1$ equations for the $n+1$ unknowns composed of the n states $x(0)$ and the compensation quantity x_{C} (e.g. the load torque of a powertrain).

For $\varphi = 2\pi$ we use the user-defined periodicity of the state vector component $i\mathsf{P}$ and solve

$$f\left(x(2\pi), 0, x_{\mathsf{C}}\right) = 0;\ x_{i\mathsf{P}}(2\pi) = p\,. \tag{6}$$

The condition that x_{C} is the same in (5) and (6) offers a possibility to check the computed solutions.

For driven systems the equations in (5), (6) may not be simultaneously solvable. In that case in each of these systems the static equation

$$f\left(x, 0, x_{\mathsf{C}}\right) = 0;$$

is replaced by the condition

$$f\left(x, v, x_{\mathsf{C}}\right) = 0;\quad \|v\|_2 \to \min$$

where $\|v\|_2$ denotes the Euclidian norm of v.

In practice it has proven sufficient to solve the resulting restricted minimization problems by a modified Gauss-Newton algorithm.

2.2.3 Harmonic Balance

For the computation of the periodical part $\tilde{x}$ in the ansatz (3) equation (2) is reformulated as the variational equation

$$\frac{1}{2\pi}\int_0^{2\pi}\psi(\varphi)\cdot y(\varphi)\,d\varphi = 0 \tag{7}$$

with ψ varying over all continuous $\mathbf{R}^n$-valued functions fulfilling the condition $\psi(0)=\psi(2\pi)$ and with

$$y(\varphi) := f\left(\frac{x_\mathrm{P}\varphi}{2\pi}+\tilde{x}(\varphi), \omega\left(\frac{x_\mathrm{P}}{2\pi}+\tilde{x}'(\varphi)\right), x_\mathrm{C}\right). \tag{8}$$

For a fixed phase velocity ω the solution $\tilde{x}$ is only determined up to a multiple of x_P and a corresponding phase shift (e.g. for a powertrain the arbitrary initial angular position). To formally fix the initial disposition, additionally the mean value of the period variable is balanced to zero:

$$\frac{1}{2\pi}\int_0^{2\pi}\tilde{x}_{i\mathrm{P}}(\varphi)\,d\varphi = 0. \tag{9}$$

In some cases the user does not want to prescribe the phase velocity ω directly (e.g. for powertrains it is usual to prescribe the mean rotational speed of the engine instead). For that reason the user-chosen reference quantity was introduced in section 2.1. Let $i\mathrm{R}$ be the index of the reference quantity and r the wanted mean value for that variable. Then instead of a direct assignment to ω the equation

$$\frac{1}{2\pi}\int_0^{2\pi}\tilde{x}_{i\mathrm{R}}(\varphi)\,d\varphi = r \tag{10}$$

is added to the variational system.

Following Galerkin for the numerical treatment of (**7**,**8**) the function space for ψ and $\tilde{x}$ is restricted to the finite-dimensional space spanned by the harmonic orthogonal system of base functions

$$\psi[k] := \exp(\mathrm{j}k\varphi) \text{ with } k=-N,\ldots,N. \tag{11}$$

In the following we keep using lower indexes for the state vector components but we use Modelica index notation to organize the frequency components (as we have already done so by defining $\psi[k]$ above). Using the base (11) for the periodical part $\tilde{x}$ in (3) the ansatz becomes

$$\tilde{x}(\varphi) = \sum_{k=-N}^{N}\exp(\mathrm{j}k\varphi)\,\hat{x}[k] \tag{12}$$

where $\hat{x}[k]$ is the k-th frequency component of the state space vector (we use a hat $\hat{x}$ or $(x)^\wedge$ to denote complex amplitudes). Since $\tilde{x}$ is real $\hat{x}[k]$ is the complex conjugate of $\hat{x}[-k]$. Thus, the values of $\hat{x}$ are determined by $n(2N+1)$ real numbers. With ψ replaced by $\psi[k]$ for $k=-N,\ldots,N$ the resulting $2N+1$ left-hand sides of (**7**) become the first $2N+1$ Fourier coefficients $\hat{f}(\hat{x},\omega,x_\mathrm{c})[k]$ of the left-hand side of (2), i.e. Fourier coefficients of the time-domain residuals. Equations (**7**,**8**,9,10) together then give the harmonic balance equation system

$$\begin{aligned}&\hat{f}(\hat{x},\omega,x_\mathrm{C}) = 0\\ &\quad\hat{x}_{i\mathrm{P}}[0] = 0\\ &\quad\hat{x}_{i\mathrm{R}}[0] = r\end{aligned} \tag{13}$$

of $n(2N+1)+2$ scalar equations for the $n(2N+1)$ unknowns in $\hat{x}$ and the additional two unknowns ω, x_C. The fast Fourier transformation (FFT) is used to approximate the Fourier-coefficients of y. Because of the nonlinearities in f the spectrum of y is wider than that one of x and some oversampling is needed for the FFT to keep the aliasing error low.

For solving system (13) Newton's algorithm is applied. Deriving the Newton corrector equation in time-domain and then transforming it into frequency-domain gives good insight into the structure of the resulting system of equations. A first order Taylor approximation of (2) in the current numerical approximation of $(\tilde{x},\omega,x_\mathrm{C})$ yields the equation

$$\begin{aligned}f+\partial_1 f\cdot\delta x+\partial_2 f\cdot(\omega\delta x'+x'\,\delta\omega)+&\\ +\partial_3 f\cdot\delta x_\mathrm{C}=0&\end{aligned} \tag{14}$$

which determines with (3) the Newton correction $(\delta\tilde{x},\delta\omega,\delta x_\mathrm{C})$ (note: (i) here $\partial_k f$ stands for the derivative of f w.r.t. the kth argument, and (ii) for clarity we have omitted the arguments $(x,\omega x',x_\mathrm{C})$ of f, (iii) $x,\delta x,\delta\tilde{x}$ are functions of φ). The time-domain products in (**14**) correspond to frequency-domain convolutions. E.g., the FFT transforms $\partial_1 f\cdot\delta x$ into

$$\left((\partial_1 f)^\wedge * \delta\hat{x}\right)[k] = \sum_{l=-N}^{N}(\partial_1 f)^\wedge[k-l]\,\delta\hat{x}[l]. \tag{15}$$

With $I:(I\hat{x})[k] := k\hat{x}[k]$ the spectrum of the derivative $\dot{x}$ can be written as $(x')^\wedge = \mathrm{j}I\hat{x}$. So, after shifting f to the right-hand side (**14**) is transformed by the FFT into the equation

$$\begin{aligned}\left((\partial_1 f)^\wedge*\delta\hat{x}\right)+\mathrm{j}\omega\left((\partial_2 f)^\wedge*(I\delta\hat{x})\right)+&\\ +(\partial_2 f\cdot x')^\wedge\delta\omega+(\partial_3 f)^\wedge\delta x_\mathrm{C}=-\hat{f}&\end{aligned} \tag{16}$$

for the unknown Newton correction $(\delta\hat{x},\delta\omega,\delta x_{\mathsf{C}})$ in the frequency domain. Together with (9) and (10) written as

$$\delta\hat{x}_{i\mathsf{P}}[0]=0;\quad \delta\hat{x}_{i\mathsf{R}}[0]=0 \tag{17}$$

this system formally determines the Newton correction in the frequency domain completely.

With the number of $n(2N+1)+2$ real unknowns the system is rather large and the convolution operator in (16) causes large fill-in of the system matrix making direct solving infeasible in real-world applications. Therefore, the iterative GMRES algorithm is used instead (see e.g. [5]). This method only requires the evaluation of the left-hand side of (16) for known $(\delta\hat{x},\delta\omega,\delta x_{\mathsf{C}})$. This also makes it possible to replace the frequency-domain convolutions in (16) by the cheaper corresponding time-domain products in (**14**) (together with the therefore needed FFT-operations). GMRES only works well with an appropriate pre-conditioner. Thus, one must be able to roughly solve systems with the left-hand side of (16) fast. For this end the block-diagonal preconditioner is used (see e.g. [6]). This approximates the convolutions by only retaining the mean value component of $(\partial_k f)^\wedge$:

$$\begin{aligned}\left((\partial_1 f)^\wedge * \delta\hat{x}\right)[k] &\approx (\partial_1 f)^\wedge[0]\cdot\delta\hat{x}[k]\\ \left((\partial_2 f)^\wedge * (I\delta\hat{x})\right)[k] &\approx (\partial_2 f)^\wedge[0]\cdot k\delta\hat{x}[k]\end{aligned} \tag{18}$$

The so approximated system (16) can be solved frequency-component wise.

If the dynamical system is linear then the Jacobians $\partial_1 f,\partial_2 f$ are constant in time and the corresponding higher spectral components in the convolutions (e.g. $(\partial_1 f)^\wedge[k-l]$ with $k-l\neq 0$ in (15)) are zero. In this case `$\approx$' in (18) can be replaced by `$=$' and the approximations are exact. For increasing nonlinearities the higher spectral components of $\partial_1 f,\partial_2 f$ omitted in the preconditioner gain influence, the approximations become more coarse. In general one can say that with stronger nonlinearities the number of GMRES iterations per Newton step and the number of Newton-iterations increase.

If the local Newton method does not converge fast enough then the Newton-algorithm with backward-error minimization via backtracking (see [1] and [7]) is applied. For a better numerical condition the states are automatically scaled during the computation.

In section 2.3 we will give an example of a nonlinear system with a turning point in its frequency response. To make the computation of such points possible a curve tracing algorithm with variable step-size is implemented in SimulationX. A short outline of this algorithm shall conclude this subsection.

Only at the starting value r_{Start} and the end value r_{Stop} of the interval for the reference quantity $x_{i\mathsf{R}}$ the full system (13) is solved. At intermediate points for $x_{i\mathsf{R}}$ the last equation determining the value of the reference quantity is removed resulting in

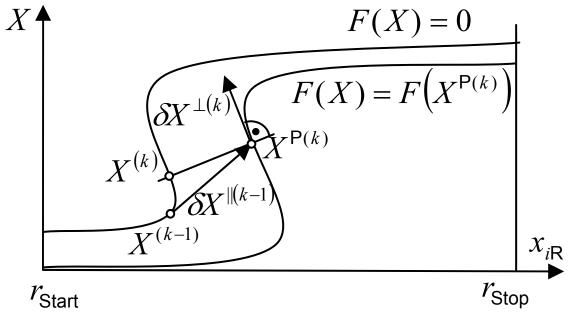


Fig. 4: Curve tracing algorithm (see text for details)

$$F(X)=0 \text{ with } F(X):=\begin{pmatrix}\hat{f}(\hat{x},\omega,x_{\mathsf{C}})\\ \hat{x}_{i\mathsf{P}}[0]\end{pmatrix} \tag{19}$$

and with the unknowns collected in $X:=(\hat{x},\omega,x_{\mathsf{C}})$.

Since (**19**) has one scalar equation less than unknowns it formally defines a solution curve (see also upper branch in Fig. 4) instead of a single point.

Given the last solution point $X^{(k-1)}$ on the solution curve and the tangent direction $\delta X^{\|(k-1)}$ of the solution curve in that point a prediction

$$X^{\mathsf{P}(k)}=X^{(k)}+s\,\delta X^{\|(k-1)}$$

for the new solution point is computed. Thereby, the step size s is chosen in dependence of the estimated curvature of the solution path, the estimated distance of $X^{\mathsf{P}(k)}$ to the solution path, and the local convergence behaviour of Newton's algorithm (for details see [2]). In the predicted point a new estimation $\delta X^{\perp(k)}$ for the tangent vector is computed as the solution of the system

$$\begin{aligned}&\mathrm{D}F(X^{\mathsf{P}(k)})\delta X^{\perp(k)}=0,\\ &(\delta X^{\|(k-1)})^T\cdot\delta X^{\perp(k)}=1.\end{aligned}$$

This is not the tangent direction to the solution curve but to the curve defined by $F(X)=F(X^{\mathsf{P}(k)})$ (see Fig. 4). Nevertheless, these curves and their tangents are supposed to be close to each other. The Newton correction for the computation of the next solution $X^{(k)}$ of (**19**) is then carried out in the affine plane with $X^{\mathsf{P}(k)}$ as origin and $\delta X^{\perp(k)}$ as normal direction. The point $X^{(k,0)} := X^{\mathsf{P}(k)}$ is used as an initial guess and the Newton corrections $\delta X^{(k,i)}$ as well as the iterated solution approximations $X^{(k,i)}$ $(i = 0,1,\ldots)$ are defined by the system

$$\begin{aligned} DF\left(X^{(k,i)}\right)\cdot \delta X^{(k,i)} &= -F\left(X^{(k,i)}\right), \\ \left(\delta X^{\perp(k)}\right)^T \cdot \delta X^{(k,i)} &= 0, \\ X^{(k,i+1)} &= X^{(k,i)} + \delta X^{(k,i)}. \end{aligned} \tag{20}$$

As Fig. 4 suggests $X^{\perp(k)}$ is a better approximation of the tangent to the solution curve at the new solution point $X^{(k)}$ than $X^{\parallel(k-1)}$. Using $X^{\perp(k)}$ lets the Newton iterations run on nearly the shortest path to the solution curve, gives (20) a better numerical condition, and avoids jumping between different solution branches at sharp turning points of the solution path.

2.3 Example: Nonlinear Spring-Mass-System with Turning-Point in Frequency Response

Unlike linear systems nonlinear systems may exhibit turning points in the frequency characteristic. The curve tracing algorithm implemented in SimulationX makes the computation of such kind of frequency characteristics possible.

The simple mechanical system of Fig. 5 is a torque excited spring-mass-oscillator. The frequency of the sinusoidal torque source is chosen as the reference quantity and swept between $0.2\,\text{Hz}$ and $0.7\,\text{Hz}$. Since this reference quantity is a parameter and not a variable SimulationX chooses it automatically as compensation parameter as well. The phase of the sine oscillator is the period variable with period 2π. The quadratic term added to the spring characteristic makes the system nonlinear in such a way that it shows a turning point in the frequency characteristic (see Fig. 6).

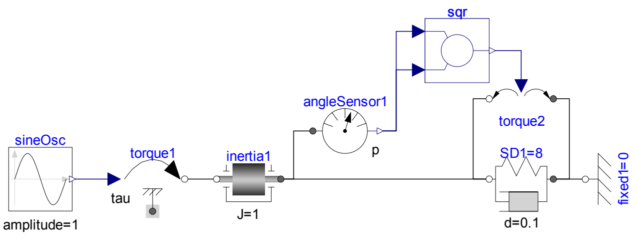


Fig. 5: Nonlinear Spring-Mass-system

In the interval from $0.397\,\text{Hz}$ to $0.426\,\text{Hz}$ the frequency characteristic is multi-valued. That corresponds to multiple periodic limit cycles at those excitation frequencies.

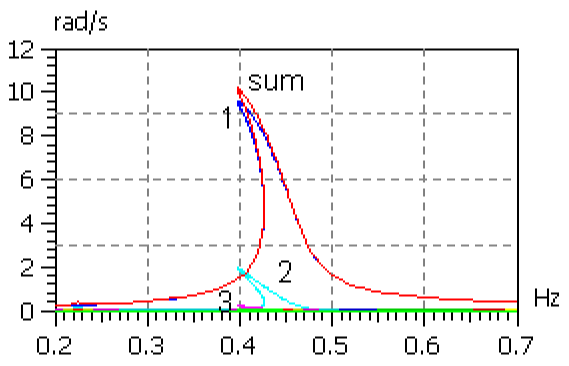


Fig. 6: Frequency response with turning-point for the angular speed of inertia1 in the nonlinear spring-mass-system; the sum curve and the first three harmonic components are distinguishable in this diagram

As an example in Fig. 7 the limit cycles from the two stable branches (lowest and highest) of the frequency characteristic at $0.405\,\text{Hz}$ are shown.

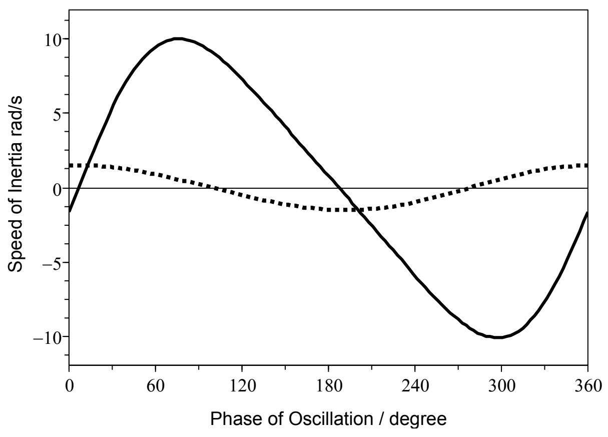


Fig. 7: Angular speed curves for the two possible stable limit cycles of the nonlinear spring-mass-system at excitation frequency $0.405\,\text{Hz}$ represented over phase.

We kept this example simple to demonstrate that even very basic nonlinear systems may have frequency responses with turning-points. More complicated examples can be found in [8], and [9].

2.4 Example: Active Electronic Filter

The periodic steady state simulation is not restricted to mechanical systems. As an example the periodic steady state simulation is applied to a Modelica model for an active electronic pass-band filter (see Fig. 8). The reference and compensation quantity in this example is the frequency of the sinusoidal source `vin` and its phase is the phase variable.

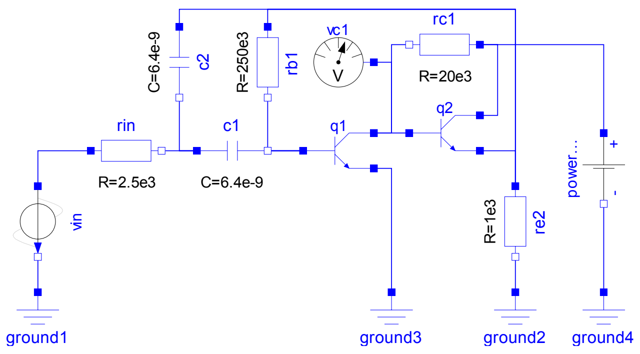


Fig. 8: Modelica model of the active electronical filter

At resonance frequency the transistor amplifier of the pass-band filter is overdriven which causes nonlinear harmonic distortions. The nonlinear frequency response of the collector voltage of transistor `q1` is shown in Fig. 9.

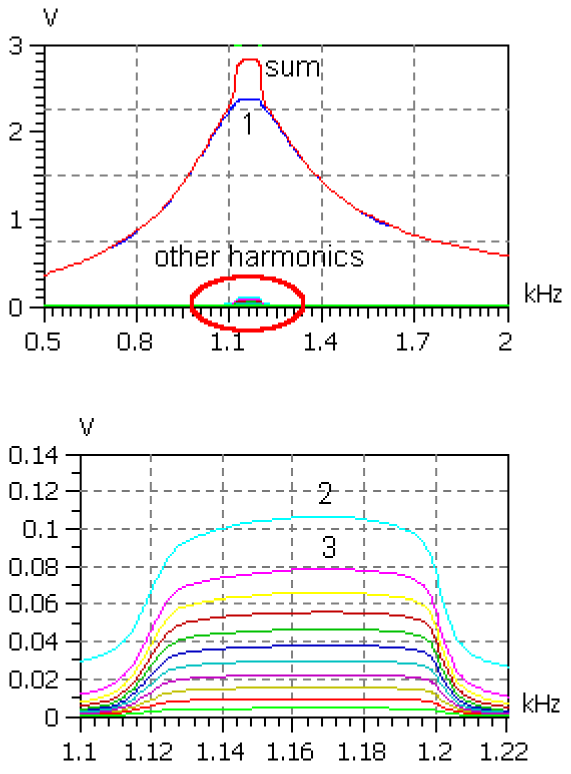


Fig. 9: Frequency response of the collector voltage of q1 in the active electronic filter; top: sum signal and first harmonic, bottom: zoomed view of the other harmonics in the resonance region where the amplifier is overdriven; the harmonics are decreasing with order, only the 2nd and 3rd harmonic are labelled

In Fig. 10 the periodic steady state result and the time domain result of this voltage over phase angle for an excitation frequency of 1.15 kHz are compared. At about 75° the base-emitter diode of q2 blocks and the voltage amplification of q1 grows which causes the spike in the collector voltage of q1.

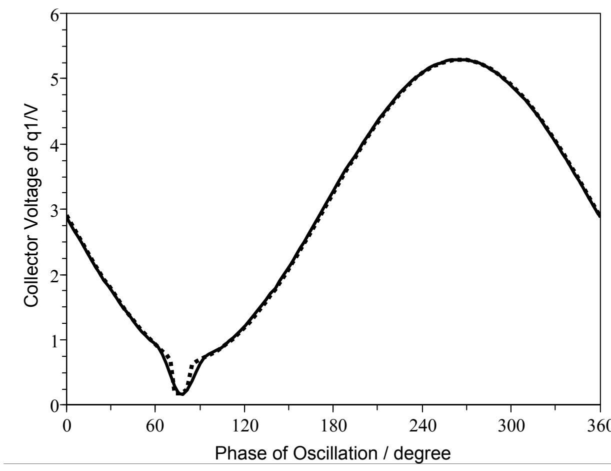


Fig. 10: Collector voltage of q1 in the active electronic filter at excitation frequency 1.15 kHz **represented over phase; full line: periodic steady state simulation, dashed line: transient simulation;**

The results are in good accordance. Nevertheless, a slight difference of the results from the periodic steady state simulation and the transient simulation is visible at about 75°. The steep slopes of the spike are somewhat smoothened by the limited number of equidistant sample-points for the steady state simulation (256 sample points per period were used).

3 Transfer Function Analysis and Natural Frequencies

3.1 Linear System Analysis

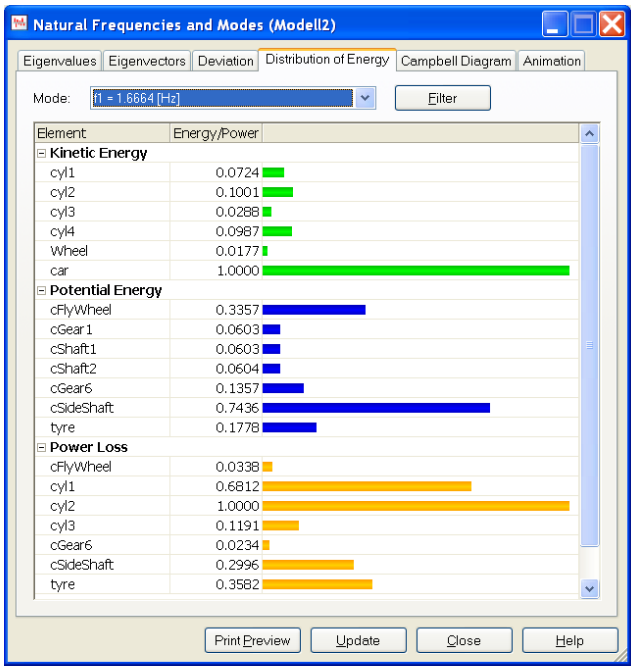
Beside the nonlinear algorithm for the steady-state simulation also linear frequency-domain analysis methods are applicable to Modelica models and are implemented in SimulationX. Those are based on the linear system which results from the linearization of the nonlinear system equations for the Modelica-model in the current operating point. The operating point may be determined by a previous transient simulation or an equilibrium computation (in electronics also called DC-analysis). Some of the algorithms may be applied to any Modelica model without changes by the user. This includes the computation of the eigensystems, the Campell diagram, and methods for the animation of the eigenmodes.

Other frequency-domain results such as the deviations in mechanical quantities (vibration modes) and the distribution of vibration energies and losses require special internal blocks that can be included into the Modelica-model. The following Modelica source code shows how the inertia from the standard Modelica library can be supplemented with an internal energy calculation block which SimulationX uses in order to compute the energy distribution.

```
model RotInertiaEnergyBlock
 import M=Modelica.Mechanics;
 extends M.Rotational.Inertia;
 Mechanics.Rotation.CalcEnergyBlock eb;
 equation
   eb.dom = w;
   eb.T = J*a;
end RotInertiaEnergyBlock;
```

The modification of the Type `SpringDamper` is similar. For a demonstration the (rotational and translational) masses and spring-dampers in the powertrain from Fig. 1 have been substituted by the modified types. The distribution of energy calculated by SimulationX for the eigenmode at 1.6664 Hz is shown in Fig. 12. In practical applications such representations show the engineer which masses, springs, and dampers dominate the behaviour in certain eigenmodes of the system, so he can take systematic countermeasures to avoid unwanted oscillations.

Up to now these blocks are not documented and only used for the internal element libraries of SimulationX. But this may change in future.



Natural Frequencies and Modes (Modell2)

Mode: f1 = 1.6664 [Hz]

Element	Energy/Power
Kinetic Energy	
cyl1	0.0724
cyl2	0.1001
cyl3	0.0288
cyl4	0.0987
Wheel	0.0177
car	1.0000
Potential Energy	
cFlyWheel	0.3357
cGear1	0.0603
cShaft1	0.0603
cShaft2	0.0604
cGear6	0.1357
cSideShaft	0.7436
tyre	0.1778
Power Loss	
cFlyWheel	0.0338
cyl1	0.6812
cyl2	1.0000
cyl3	0.1191
cGear6	0.0234
cSideShaft	0.2996
tyre	0.3582

Fig. 11: Distribution of energy for the powertrain example from Fig. 1

3.2 Input-Output Analysis

For the analysis of the input-output-behaviour the user must select the input and the output of the linearized system. Any result variable of the model may be used as the system output. SimulationX has a special class of signal inputs that may be open even for the top-level model. Those inputs may be used for the input-output-analysis. In Fig. 11 a cut-out of the powertrain from Fig. 1 is shown where a torque source with such an input has been added. The input-output behaviour is described by the frequency response function and the pole-zero diagram of the system.

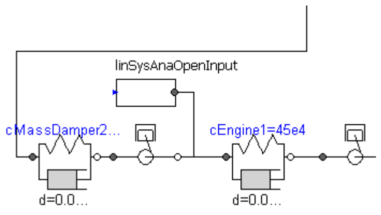


Fig. 12: Element linSysAnaOpenInput in the example from Fig. 1 with open input for the input-output-analysis

Fig. 13 and Fig. 14 show the pole-zero plot and the frequency characteristic, resp., for the powertrain from Fig. 1 with the torque at the first cylinder as input (Fig. 12) and the torque in the engine damper as output.

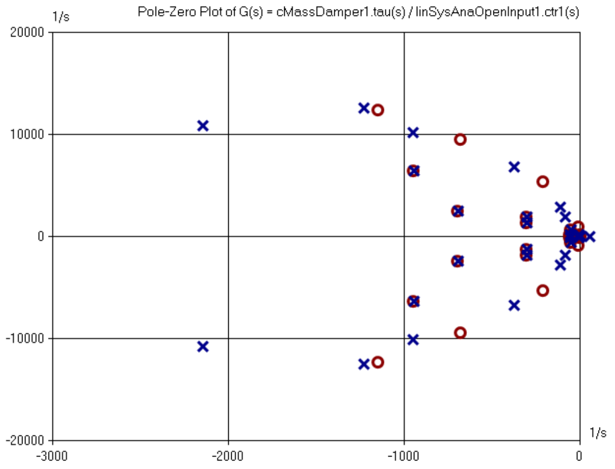


Fig. 13: Pole-zero plot of the system in Fig. 1; crosses: poles, circles: zeros

For further analysis in external tools the linearized system matrices may be exported in Modelica or MATLAB syntax.

4 Conclusions and Outlook

Periodic steady state simulation proves useful for the vibration analysis of nonlinear systems. SimulationX allows its application to Modelica models, in particular to powertrains, without the decomposition into nonlinear exciter and linear drivetrain. Furthermore, the method is applicable to driven systems of other physical domains since it is purely equation-based. Only very little knowledge of the system is required from the user. Two mechanical examples and one from electronics were given in the paper.

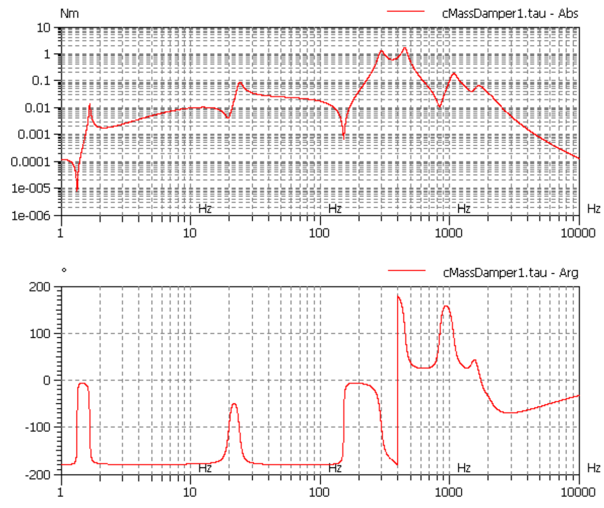


Fig. 14: Frequency response of the system in Fig. 1; top: amplitude, bottom: phase

Furthermore, we discussed methods for the small-signal analysis in the current operating point (resulting from a transient or equilibrium computation). Beside pole-zero plots and frequency response functions also some remarks about the deviation- and energy distribution analysis for oscillation modes were given. They are especially useful for the mechanical engineer to detect the powertrain elements which participate in selected oscillation modes.

• **Behaviour Description In Frequency Domain:** In future it is planned to include a behavioural description in frequency domain (e.g., for modeling of dynamic stiffness) for the periodic steady state simulation as well as for the frequency response computation, which was one main argument for the harmonic balance method to be preferred over the shooting method (see e.g. [11] for a short introduction and further references). One major reason for the frequency domain description not yet being implemented in SimulationX is that Modelica currently still lacks a standardized way for computations with complex numbers (even if some steps in this direction have already been taken, see e.g. [10]).

• **Event Iterations:** Event iterations are already embedded into the harmonic balance algorithm. But there remains still some work for the treatment of time-discrete variables in special cases.

• **Improved Convergence for Strongly Nonlinear Systems:** As long-term objective the convergence speed of the harmonic balance for strongly nonlinear systems can be improved by time domain preconditioners (see [6]).

• **Autonomous Systems:** The ansatz used for the harmonic balance also bears the potential for the simulation of autonomous systems. The required randomization of the start values for the harmonic balance could be implemented.

• **Detection of Stable/Unstable Limit Cycles:** Up to now there is no automatic discrimination of the stable and unstable branches in the nonlinear frequency response computed via harmonic balance. This can be implemented by an eigenvalue analysis of the monodromy matrix of the computed limit cycles.

References

[1] J. E. Jr. Dennis and Robert B. Schnabel: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM 1996.

[2] E. L. Allgower and K. Georg: Numerical Continuation Methods: An Introduction. Springer-Verlag, 1990.

[3] http://www.simulationx.com

[4] H. Dresig and F. Holzweißig: Maschinendynamik. 5th ed., Springer-Verlag Berlin, 2004.

[5] A. Meister: Numerik linearer Gleichungssysteme. Vieweg-Verlag, Wiesbaden, 2005.

[6] Ognen J. Nastov: Methods for Circuit Analysis. PHD-theses, Massachusetts Institute of Technology, 1999.

[7] U. Feldmann, U. A. Wever, Q. Zheng, R. Schultz, and H. Wriedt: Algorithms for Modern Circuit Simulation. AEÜ, Vol. 46 (1992), No. 4.

[8] A. Al-shyyab and A. Kahraman: Non-linear dynamic analysis of a multi-mesh gear train using multi-term harmonic balance method: period-one motions. Journal of Sound and Vibration, 284 (2005) 151-172.

[9] Wen-I Liao, Tsung-Jen Teng, and Chau-Shioung Yeh: A method for the response of an elastic half-space to moving sub-Rayleigh point loads. Journal of Sound and Vibration 284 (2005) 173-188.

[10] Peter Aronsson at al.: Meta Programming and Function Overloading in OpenModelica. Modelica 2003, November 3-4, 2003.

[11] Kenneth S. Kundert: Introduction to RF Simulation and Its Application. IEEE Journal of Solid-State Circuits, Vol. 34, No. 9, September 1999.

World3 in Modelica: Creating System Dynamics Models in the Modelica Framework

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

Abstract

This paper introduces a new release of the SystemDynamics library of Modelica and shows how it is being used by discussing a fairly large application code: Meadows' World3 model. The newest version of that model has been made available in the library.

Keywords: *System Dynamics, World Dynamics, Soft Science Modeling*

1 Introduction

System Dynamics represents a fairly low-level modeling paradigm. Its implementation does not place heavy demands on the modeling software. Hence Modelica may in fact be a bit of an overkill for dealing with System Dynamics models. However, it is considerably better suited than the state-of-the-art software for this type of modeling, i.e., Stella [11], the code that most System Dynamics modelers use today.

A first version of a System Dynamics library for Modelica was released in 2002 [3]. In the present paper, a new release, SystemDynamics 2.0, is being discussed. SystemDynamics 2.0 is not an upgrade of SystemDynamics 1.0, but rather a re-implementation of the methodology. Inherited from SystemDynamics 1.0 were only two application codes, a small introductory model concerning lynxes eating hares, and a considerably more complex model borrowed from Forrester's Industrial Dynamics book [4].

As already mentioned above, the basic models implementing the System Dynamics methodology, levels and rates, are so simple that their implementation in Modelica requires very little time and effort. The value of the library is not in its basic models, but rather in its application codes.

Among other applications, SystemDynamics 2.0 offers two full World models, namely Forrester's World2 model [5], and Meadows' World3 model [7,8].

Whereas Forrester described his model in full in his World Dynamics book [5], Meadows' only talked in Limits to Growth about the results obtained with the model [8]. The model itself, originally coded in Dynamo [10], was described in a separate book [7].

Meadows' World3 model has seen two major upgrades since its original inception, one in 1992, i.e. after 20 years, and the second in 2002, i.e., after 30 years. The World3 application code contained in SystemDynamics 2.0 implements the 2002 version of the World3 model. In the code, we offer not only the basic model, but also all 10 scenarios that Meadows and co-workers are talking about in Limits to Growth: The 30-Year Update [8].

Although the work of Forrester and Meadows caused quite a stir in the early 70s when their books first appeared, world modeling became unfashionable fairly quickly, because essentially all sources of funding dried out for political reasons.

Only very recently, in the context of the looming *Peak Oil* event and because of the ongoing discussions concerning *Global Warming*, has world modeling become respectable again.

It turned out that Forrester and Meadows were essentially correct in their assessments, in spite of the fact that their models were very crude in comparison with real world dynamics.

With this paper, I wish to open up world modeling to the community of Modelica users.

2 Short History of System Dynamics

The *System Dynamics* approach to modeling dynamic systems was developed in the 1960s by Jay Forrester with the aim of creating a modeling and simulation tool that economists would be able to handle.

Instead of talking about differential equations, he talked about "levels," the values of which were changed by "rates." Level variables are variables that can accumulate. For example, population might be used as a level variable. It is controlled by two rate variables, the birth rate and the death rate.

Forrester would draw this relationship in a diagram similar to the one shown in Fig.1.

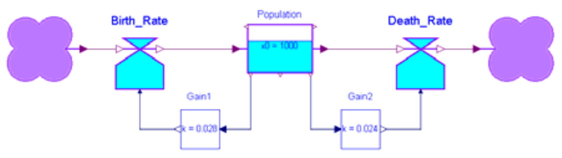
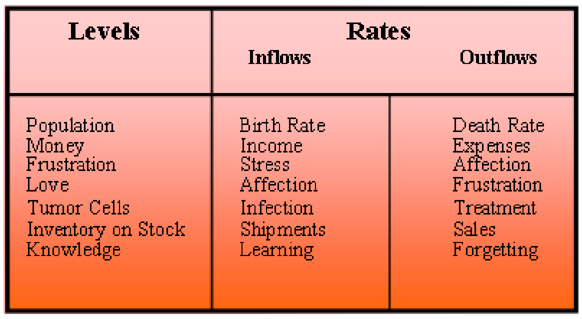


Figure 1: Population with birth and death rates

The blue square box represents a level. It requires an initial value. The blue icons to the left and right of the level represent rates. Both the birth and the death rate are proportional to the population. The two clouds represent sources and sinks of material. They are used for documentation purposes only. There are no equations associated with these models. The lilac lines represent material flows, whereas the blue lines represent information flows.

Of course, Forrester didn't have a computer available with a graphical user interface. He drew his diagrams only by hand and then translated them manually (and quite mechanically) down to a set of equations that he then encoded in Dynamo [10], a simulation "language" that had been outdated already at the time of its creation.

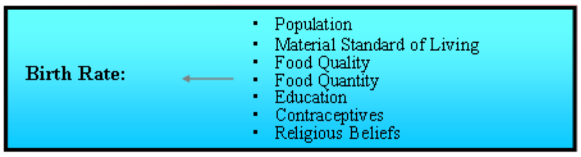
Forrester explained to his disciples that every modeling exercise should always start with pondering, which are the most important accumulator variables that ought to be captured in the model. These variables should be declared as level variables. Subsequently, it needs to be decided, what other variables can be viewed as inflows and outflows to and from these levels. The inflows and outflows would then become the rate variables. Fig.2 shows a typical set of levels and their rates.



Levels	Rates	
	Inflows	Outflows
Population	Birth Rate	Death Rate
Money	Income	Expenses
Frustration	Stress	Affection
Love	Affection	Frustration
Tumor Cells	Infection	Treatment
Inventory on Stock	Shipments	Sales
Knowledge	Learning	Forgetting

Figure 2: Typical level and rate variables

The modeler would then need to decide, which other variables the rates depend on, and write these down in a so-called "laundry list." A possible laundry list for the birth rate is offered in Fig.3.



Birth Rate: ←

- Population
- Material Standard of Living
- Food Quality
- Food Quantity
- Education
- Contraceptives
- Religious Beliefs

Figure 3: Birth rate laundry list

So far so good, but now comes the most daring assumption, the "quantum leap" of System Dynamics.

The functional relationship represented by such a laundry list can be assumed to be a static non-linear function in multiple variables, e.g.:

$$Birth\_rate = f(Population, Pollution, Food, Crowding, Material\_Standard\_of\_Living)$$

Yet, since such a function may be too difficult to identify, Forrester chose to ignore the mutual relationship among the different input variables, and postulate the following model instead:

$$Birth\_rate = BRN \cdot Population \cdot f_1(Pollution) \cdot f_2(Food) \cdot f_3(Crowding) \cdot f_4(Material\_Standard\_of\_Living)$$

The birth rate is essentially computed as the average birth rate, *BRN*, multiplied by the population. All other dependencies are expressed as small signal deviations from the norm. The single-valued functions can most of the time be easily approximated using information from the open literature, e.g. from statistical yearbooks.

Forrester was wildly successful with his approach to modeling. Whereas engineers and physicists mostly ignored him, if they didn't even sneer at his "methodology," researchers from the soft sciences loved it. Already by the early 1980s, several thousands of papers making use of System Dynamics for a variety of modeling projects had been published [6].

By 1984, the Macintosh became available, and with it, programmers were for the first time offered an easily programmable graphical user interface. Within a short time, a graphical modeling environment for System Dynamics modeling, Stella [11], became available that quickly replaced Dynamo [10] as the tool of choice for System Dynamics modeling.

Today, more than 20 years later, Stella is still the most widely used tool for System Dynamics modeling. The language has seen a few improvements over the years, but by and large, it is still the same software that had been created in the mid 1980s.

A Stella model of population and its two rate variables is shown in Fig.4.

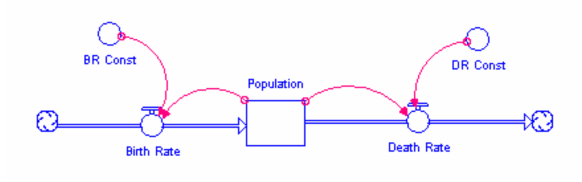


Figure 4: Stella model of population growth

3 The WORLD3 Model

World Dynamics became quickly one of the most prominent endeavors of System Dynamics modelers. Among the earliest world models created for the Club of Rome were Forrester's WORLD2 and Meadows' WORLD3 models. Both of these models are made available as part of the new SystemDynamics library.

Which are the most important drivers (accumultors) behind any world model? The list of levels ought to include at least:

- population
- pollution
- resource utilization
- invested capital
- work force
- food

Different world models vary in the degree of sophistication, with which they consider these sectors. In this paper, we shall primarily focus on the WORLD3 model, as this model has been upgraded several times, and therefore is still up-to-date.

3.1 Population Dynamics

The population dynamics model of WORLD3 is shown in Fig.5.

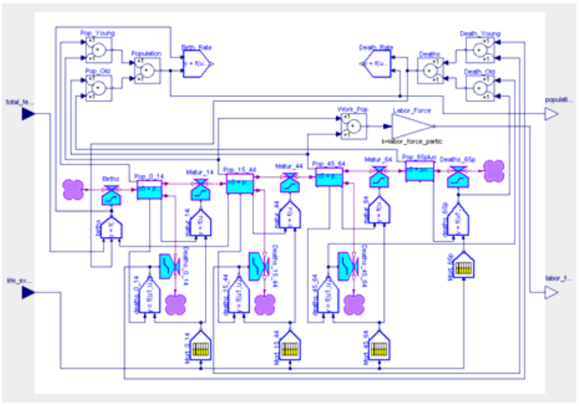


Figure 5: Population dynamics in WORLD3

The model is quite easy to read. The population is subdivided into four separate levels, representing:

1. children (until age 14)
2. young adults (until age 44)
3. older adults (until age 64)
4. seniors

This division makes sense, as the work force is comprised of groups #2 and #3 only, and people of reproductive age are those in group #2. The rates between the levels compute the maturation from one group into the next. Beside from the births and the final deaths, there are also people dying prematurely out of each of the four groups.

The birth rate depends on the fertility, which is computed by another module. The death rates in the four groups are modeled as tabular functions of the life expectancy, which is also computed elsewhere.

The model exports the total population and the labor force, as these variables are used by other modules.

Notice that WORLD3 is a global model. All variables are averaged over the entire globe. The model does not distinguish between Europe and Africa, for example. This limits the types of questions that may be answered by it.

3.2 Pollution Dynamics

The pollution dynamics model of WORLD3 is depicted in Fig.6.

The pollution model contained originally a single state variable: the accumulated pollution. New pollution is being generated in proportion to the total resource utilization and in proportion to the arable land used for agriculture. Pollution is being assimilated again in proportion to the accumulated pollution by the self-regulating mechanisms of this planet.

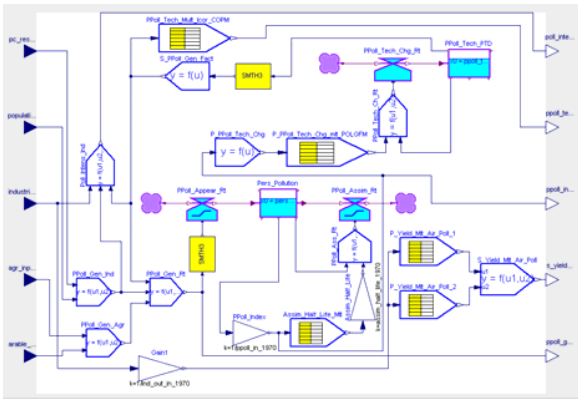


Figure 6: Pollution dynamics in WORLD3

Of a more recent vintage is the second state variable that denotes the capital invested in pollution avoidance technology. Meadows and coworkers recognized at some point in time that the amount of pollution generated may be partly mitigated by investing in pollution avoidance technology. The inflow rate associated with this second state variable is an unrestricted rate that can also assume negative values, thereby turning the inflow rate into an outflow rate.

Notice that this is not a greenhouse gas emission model. The model attempts to estimate total pollution of various kinds. The measurement units associated with pollution in the model are somewhat obscure.

This would, however, be the place where a global greenhouse gas emission model could (and probably should) be added at some point in time.

3.3 Resource Utilization Dynamics

The resource utilization dynamics model of WORLD3 is depicted in Fig.7.

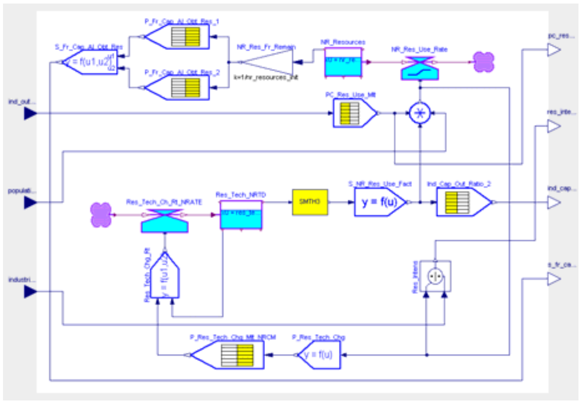


Figure 7: Resource utilization dynamics in WORLD3

The model is similar in structure to the pollution dynamics model. Originally, there was only a single state variable describing the non-recoverable natural resources that are being depleted. Resource depletion occurs approximately proportional to the total industrial output. The resources get consumed in the process of producing goods. As the resources get depleted, production inevitably slows down.

A second state variable was introduced in a later version of the model describing the effects of recycling. As resources get recycled rather than discarded, resource utilization for the same amount of produced goods slows down. The same technological advances that enable recycling also reduce the generation of pollution.

In WORLD3, the production sector is subdivided into three sub-areas concerning the production of consumer goods, the production of food, and the production of services.

Resource depletion is an important factor in the model as it negatively influences all three production sectors.

Notice that the resources, as computed by the model represent primarily minerals, not fossil fuels. WORLD3 does not model fossil fuel utilization directly.

Fossil fuels could (and probably should) be included as a separate state variable within the resources sector of the model.

3.4 The Overall Model

The overall WORLD3 model is depicted in Fig.8.

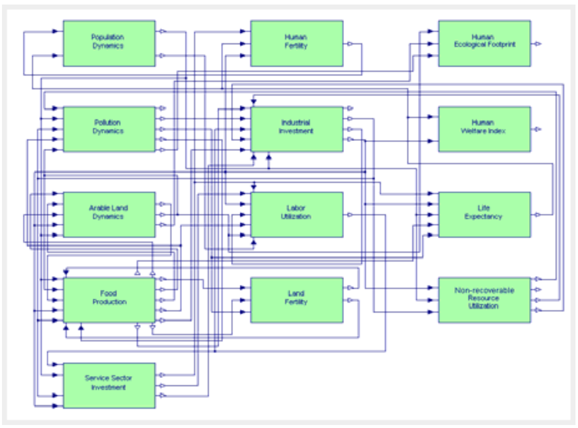


Figure 8: Overall WORLD3 model

I subdivided the WORLD3 model into 13 different sectors, capturing the dynamics of population, pollution, arable land development, food production, the service sector, human fertility, industrial investments, the work force, land fertility, the human ecological footprint, the human welfare index, life expectancy, and last but not least the utilization of non-recoverable natural resources. Three of those were presented in the previous sections of this paper. The overall model invokes one of each of the 13 sector models and connects the terminal variables of those sector models among each other.

We are now ready to simulate the model. The compiled model contains 41 state variables and 265 algebraic variables. A few simulation results are shown in Figs.9 and 10.

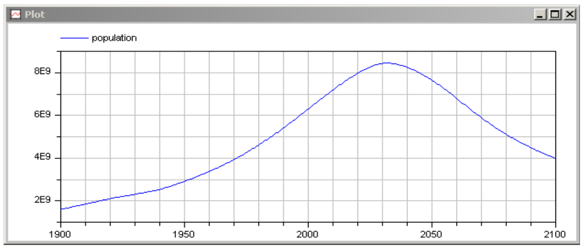


Figure 9: Population as a function of time

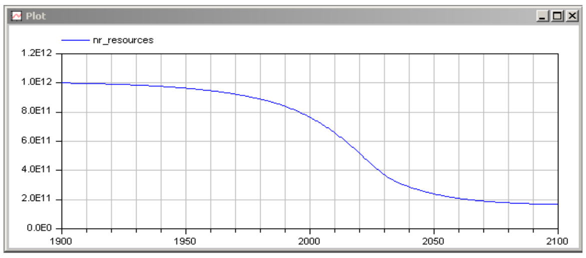


Figure 10: Natural resources as a function of time

The simulation results are identical to those shown in the book Limits to Growth [8]. The population grows until roughly 2030. At that time, the non-recoverable resources have been depleted to an extent where production can no longer proceed as before. In particular, less food gets produced, which leads to a decline in the population.

Can we trust these results? To answer this question, it may be useful to look at scenario #2. In this scenario, Meadows and his co-workers postulated that the amount of the remaining non-recoverable natural resources had been massively underestimated. The amount sill available in 1900 is thus doubled. Furthermore, it is proposed that, in 2002, money is being invested in producing the remaining resources more efficiently.

Some simulation results of this scenario are shown in Figs. 11 and 12. The results from scenario #1 are superposed for comparison. We would expect that, since resource depletion won't occur as quickly, the population can continue to grow for some time after 2030.

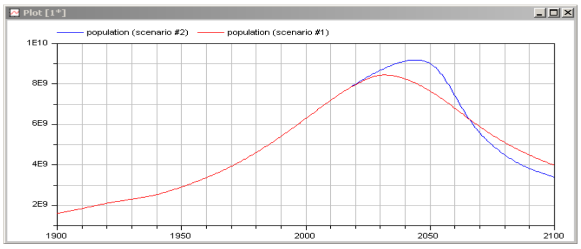


Figure 11: Population as a function of time (scenario #2)

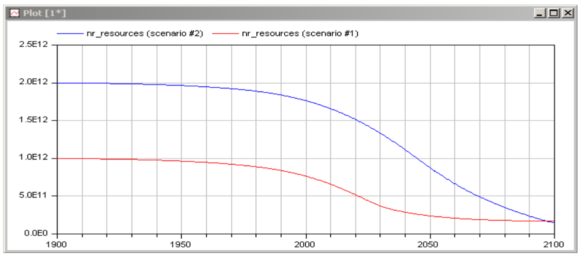


Figure 12: Resources as a function of time (scenario #2)

In this scenario, the population is indeed able to grow for a little while longer, but now it starts shrinking at 2045, although the resources aren't getting depleted until 2080. This time around, the cause of the die-off is the pollution. Pollution is allowed to continue to increase unabated, which eventually hampers our ability to grow food.

Whereas scenario #1 suffers (in a general sense) the effects of *Peak Oil*, scenario #2 is plagued by *Global Warming*. Similar results were shown in earlier editions of Limits to Growth [8]. The main difference between the models is the year, in which corrective action is being taken in the different scenarios. In the first edition of the book, corrective actions were taken in 1972. However, we already know that this didn't happen. Hence, the 3rd edition proposes corrective actions to take place in 2002 only. By postponing the intervention, the window of opportunity for still influencing the simulation results in a significant way shrinks.

Why do I believe these results? It is, because they aren't very sensitive to the scenario chosen. Whatever we do, if it is not one factor that brings us to the limits of growth, it is another … and irrespective of what we do, it always happens in the 21st century. It may happen a few years earlier or a few years later, but the general picture doesn't change at all.

Also the (much simpler) WORLD2 model that features a different set of state variables and different interactions between them essentially paints the same picture.

Since the 1980s, we are consuming more resources per time unit than the planet is able to regrow [1,12]. We are living beyond our means. This is not sustainable. It cannot continue indefinitely.

So, will the decline take place? Maybe it won't. Maybe the moon is made out of Swiss cheese.

3.5 Analysis of Simulation Results

Meadows and co-workers found two scenarios that look a bit more hopeful. These are scenarios #6 and #9. Let me analyze these two scenarios in more detail. To this end, we shall continue the simulation all the way until 2500.

In scenario #6, a whole palette of interventions was enacted in 2002. These include the interventions of scenario #2. In addition, money was invested in improved pollution control technology (scenario #3), in enhanced land yield (scenario #4), in increased land erosion control (scenario #5), and in augmented resource utilization efficiency (scenario #6).

Some simulation results are depicted in Fig.13.

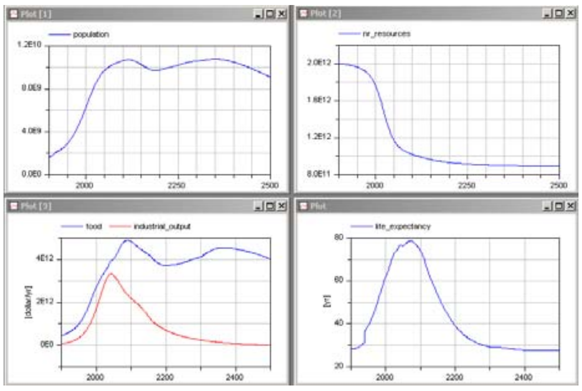


Figure 13: Simulation results of scenario #6

This scenario is indeed sustainable. The world population hovers at approximately 10 billion people. The remaining natural resources get no longer consumed.

Yet, humanity is paying a heavy price for insisting on maintaining such a large population. It spends all of its resources in producing food, and does so with the most primitive of means. The industrial output, and also the service sector output get reduced to almost zero. This is also why the remaining natural resources are no longer being consumed. Worst of all, the life expectancy is back at a value as it was experienced prior to the industrial age. The average human dies before age 30 due to huge infant mortality.

Let us now look at scenario #9. In that scenario, additional interventions are chosen. The scenario starts out with scenario #6, but in addition enforces strict population control (scenario #7), and products are being built that last 25% longer on average (scenarios #8 and #9).

Some simulation results are depicted in Fig.14.

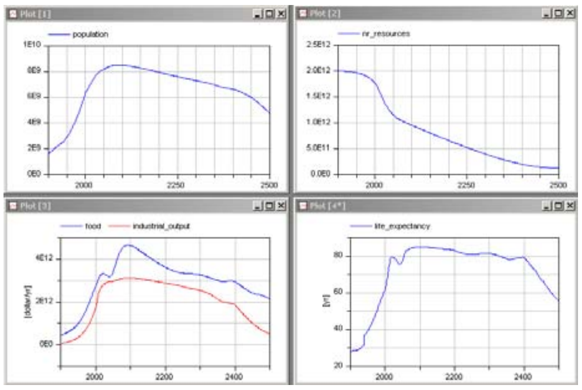


Figure 14: Simulation results of scenario #9

By enforcing strict population control, the world population is kept at a maximum value of 8 billion people. The scenario promises a golden age that will last for 400 years. Unfortunately, the scenario is not fully sustainable, as the natural resources continue to be used up, and by the year 2400, the industrial output, and with it also the population and life expectancy start declining again.

4 Dymola *vs.* Stella

What have we gained by offering a System Dynamics modeling capability in Dymola and by porting the WORLD3 model to that new environment?

Stella, contrary to Modelica, is not truly object-oriented. Large models are handled in Stella by supporting the concept of a virtual canvas. The physical screen can be scrolled over the virtual canvas, enabling the user to look at parts of the model separately. However, there is no feature available that would help a user find a particular spot, such as the population dynamics model, on the canvas.

Stella furthermore does not offer an icon editor. Stella only supports three types of icons that are all displayed in Fig.4. The square boxes represent levels (or "stocks," as they are being called in Stella); the circles with the tap on top denote the rates (or "flows," as they are being named in Stella), and the circles without a tap are everything else (linear and non-linear functions, tabular functions). For this reason, Stella diagrams don't offer mnemonic hints. They look all the same, irrespective of what they represent (just like a bond graph [2]).

The numerical ODE solvers offered by Stella are rather poor. Also, Stella computes internally with an accuracy of 2 digits after the comma only (triggered by the fact that Stella is frequently used by economists who think in terms of dollars and cents).

On the other hand, Stella offers better support in dealing with tabular functions. Each 1D table is immediately plotted in the parameter window, and the user can tweak the curve by moving supporting values around using the mouse.

Furthermore, Dymola forces the user to create a separate block for each non-linear function and program the non-linear relationship either graphically in its diagram window or alphanumerically in its equation window. In contrast, Stella offers a generic non-linear function block that enables the user to create the non-linear relationship interactively in the parameter window of that generic block. The user doesn't even need to retype the names of the input variables. The parameter window of that generic function offers a scroll-down list of the names of all input variables, and the user can simply click on any of those in order to get them included in the expression.

Finally, Modelica has been designed by engineers for engineers. It is based heavily on SI units. Whereas the user can declare types based on these units, he cannot declare new units. Whereas this works well for most engineering endeavors, it causes problems when dealing with soft science models.

Sometimes, new derived units are needed. For example, time in System Dynamics models is often measured in years rather than seconds. Whereas Dymola offers the possibility to declare new display units, the user cannot change the units used in computations. This is inconvenient. Of course, the types encoded in the SIunits library are based on SI units. Thus, if a user wishes to declare his own units, he will have to declare his own types based on these units also.

Even worse, however, are those units that cannot be expressed at all in terms of SI units. For example, many System Dynamics models operate on units of money. Dollars cannot be expressed in terms of SI units at all.

The most important advantage of Dymola is the fact that the entire System Dynamics knowledge is encoded at the level of Modelica. The interface can therefore be easily modified and enhanced by the user. In contrast, Stella's user interface is completely hard-wired. The user cannot modify the syntax or semantics of Stella in any way, and therefore, new ideas cannot be incorporated into the code except by talking the designers of the tool into including them with their next software release.

5 Conclusions

In this paper, a new release, or rather re-implementation, of the System Dynamics library of Modelica was presented.

The System Dynamics methodology is very easy to use, and consequently, does not really require much of an introduction. The most important value of a System Dynamics library is the knowledge encoded in its application examples. Currently, the by far most valuable part of the new library are its world models.

What future additions are in the works? In today's world of dwindling fossil fuel reserves, it becomes important to track how much energy we are actually using. Whereas classical System Dynamics is designed to track material flows, it does not track energy flows. This is a major drawback of the methodology.

For this reason, a second version of the System Dynamics library has also been released as a sub-library of BondLib [2], our bond graph library. In that version, all material flows are represented internally by bond graphs. A bond graph naturally tracks energy flows. Each energy flow, in that version of the library, is represented as the product of a specific enthalpy and a mass flow. Hence we can track material flows and energy flows simultaneously.

When I drive my car from home to work, I am not only spending energy in the form of the gas that my car consumes. Some energy was also spent in producing the car, and more energy will be spent in discarding it at the end of its lifecycle and in recovering those materials from it that can be recycled.

The accumulated energy that accounts for all of those indirect uses of energy is called *emergy* [9]. The specific enthalpy can be used to encode in the model the *specific emergy*, i.e., the emergy per unit of mass.

I plan on porting examples of emergy modeling, as described in the publications by Howard Odum, over to the bond graph implementation of the System Dynamics library, but this work has not yet been completed.

Acknowledgments

I wish to express my gratitude to Dennis Meadows and Jørgen Randers. Without their generous help and support, I would have been unable to port the WORLD3 model and all of its 10 scenarios over from Stella to Dymola (Modelica).

References

[1] Cellier, F.E.: Ecological Footprint, Energy Consumption, and the Looming Collapse. *The Oil Drum*, May 16, 2007

[2] Cellier, F.E., Nebot, A.: The Modelica Bond Graph Library. In: *Proceedings of the 4th International Modelica Conference*, Hamburg-Harburg, Germany (2005) Vol. 1, 57-65

[3] Fabricius, S.M.O.: *SystemDynamics Modelica Library; Brief Feature and Example Documentation*. Modelica Website, 2002

[4] Forrester, J.W.: *Industrial Dynamics*. M.I.T. Press, 1961

[5] Forrester, J.W.: *World Dynamics*. Pegasus Communications, 1971

[6] Lebel, J.D.: System Dynamics. In: *Progress in Modelling and Simulation* (F.E. Cellier, ed.), Academic Press, London (1982) 119-158

[7] Meadows, D.L., Behrens III, W.W., Meadows, D.H., Naill, R.F., Randers, J., Zah, E.K.O.: *Dynamics of Growth in a Finite World*. Wright-Allen Press, 1974

[8] Meadows, D.H., Randers, J., Meadows, D.L.: *Limits to Growth: The 30-Year Update*. Chelsea Green, 2004

[9] Odum, H.T.: *Environmental Accounting: Emergy and Environmental Decision Making*. John Wiley, 1995

[10] Richardson, G.P., Pugh III, A.L.: *Introduction to System Dynamics Modeling with DYNAMO*. M.I.T. Press, 1981

[11] Richmond, B., Peterson, S., Vescuso P.: *An Academic User's Guide to STELLA*. High Performance Systems, Inc., Lyme, N.H., 1987

[12] Wackernagel, M., Rees, W.: Our Ecological Footprint. *Green Teacher*, 45 (1995) 5-14

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.

Modelica as a host language for process/control co-simulation and co-design

Filippo Donida, Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio, 34/5 – 20133 Milano, Italy
{donida,leva}@elet,polimi.it

Abstract
The manuscript describes a project, currently under development at the Politecnico di Milano, the aim of which is to create an integrated environment for the modelling and simulation of process control systems, where the plant(s) are described according to the Modelica object-oriented paradigm, while the control systems are specified in an IEC 61131.3-compliant language, and automatically translated into algorithmic Modelica. Preliminary results will be reported, given the vast scope of the project, but even at the present stage, interesting discussions are possible on the potentialities and pitfalls of Modelica (and even of object-oriented modelling at large) when it comes to describe control algorithms of realistic complexity and size.

1. Introduction

A significant experience is nowadays available on the use of Modelica to model, simulate and assess control systems in the process domain [15, 16, 12, 13]. As witnessed by many references (samples will be given in the final manuscript, including some directly related to the authors' experience) there is a correspondingly vast *corpus* of libraries, models, and system studies [7, 6, 15, 16, 14, 4].

Based on that experience, a critical point when dealing with applications of realistic size is invariantly the "correct" representation of the control system. The object-oriented paradigm can be suitably exploited to allow for various, interchangeable control representations of different, scalable complexity, and such a possibility is definitely *a plus* of Modelica. However, when it comes the time to describe the control system in full detail, the most effective way to do so is not only algorithmic, but compliant with the industrial standard accepted in that domain, the IEC 61131-3 being the most important one [8, 18, 17, 9, 5, 2, 11]. Adhering to an industry standard is beneficial not only in terms of acceptability of the developed simulators on the part of people who know much more about their processes than about simulation (a problem worth addressing in any case, however) but also in terms of reduced ambiguity in the realisation of controller models [3, 4].

After several years of experience on the matter, the authors are strongly convinced that Modelica is very well suited as a host language for the representation of realistic-scale process controls, but that to do so it is highly desirable to allow for the specification of such controls in IEC-compliant languages.

Based on the above idea, the AutoEdit (the name may change in the future) project was started. The aim of the project is to set up a tool composed of

- a graphical Modelica editor, aimed at writing the "plant model",
- an editor for IEC 61131.3 languages (at present the Ladder Diagram, Sequential Functional Chart and the Functional Block Diagram are being considered), aimed at writing the "control model",
- a "compiler" capable of translating both the "plant" and "control" model in a single Modelica file, to be fed to any Modelica translator for simulation (the term "compiler" being used here for analogy and compatibility with the IEC terminology, albeit the Modelica jargon would most likely advise something like "pre-translator"),
- and a simulation output browser.

To the best of the authors' knowledge, such a tool is the only one allowing to couple Modelica process modelling with IEC (i.e., industry standard) control system representation, greatly facilitating the creation of simulators of process control systems.

AutoEdit is fully written in java (hence cross-platform), uses the XML language as internal data format for maximum openness and transparency, and is entirely free software, released under the terms of the GPL license. It is the authors' intention to allow AutoEdit to operate with any Modelica translator, so as to maximise its use and to have the maximum amount of feedback for improvement. At present, the AutoEdit site is hosted at the URL http://home.dei.polimi.it/donida/projects.php?project=AutoEdit

The paper organised as follows. First a minimal review of the background. Then, a discussion is carried out on the opportunity of generating event-driven Modelica code with an *ad hoc* tool, instead of describing control system components, as already attempted, with Modelica (continuous time based) models. The outcome of such discussion, as can be guessed, is that the "best" approach depends on the size of the considered application, direct generation of algorithmic code being preferable in the case of large (control) systems. The AutoEdit project is then described, illustrating its goals, structure, organisation, present state, and future developments.

2. Background

Recent advances in object-oriented modelling allow to tackle the simulation and the computer-aided control system design of industrial plants in a unified framework. Traditionally, however, the plant study and design, the following design assessment simulations, the control system design, the overall system validation, and the operator training, are not developed in a coordinate way within a single environment. By vastly acknowledged opinion, doing so is a waste of time and resources, not to say a possible source of errors, because the involved environments are frequently not compatible each other, requiring manual intervention to transfer information from one tool to another..

The Modelica multi-physics approach allows *per se* to perform a first integration of two of the involved frameworks: the plant model and its control are defined with an *equation* section for the plant and an *algorithm* section for the control code, and then the two sections are unified in a single model and simulated simultaneously.

In the present software engineering arena, translators and cross-compilers are well diffused, but basically such tools are available for the software development only. To the best of the authors' knowledge there are no similar examples in *simulation for control* area, except for some *ad hoc* solutions pertaining to the micro-controller real-time applications.

The AutoEdit is an attempt to fill the gap sketched above. It is in the first place an integrated IEC61131.3 compliant environment for the graphical development of the control programs, having (algorithmic) Modelica as the target language. Moreover, it proposes new standard for the Ladder Diagram (LD) and Sequential Function Chart (SFC) file representations, using the XML language and DTD validation. AutoEdit also encompasses a converter from the SFC XML to LD XML format, managing different level of variables' scope, as required to be compatible with the way IEC-compliant projects are organised. In one word, AutoEdit is an attempt to allow developing the model of a complete control application (process and control system) in a single environment, and having as final output a complete simulator of the overall application.

3. Modelling control code in Modelica

Consider the way a control application is typically developed in an IEC-compliant environment. The application is composed of programs, written in one or more of the supported languages, and linked together by the development tool. The programs of an application are organised into sub-applications, that in turn are deployed to one or more CPUs and arranged into threads, each one composed of programs that share the cycle time. i.e., the temporal cadence for the update of inputs and outputs.

The goal of AutoEdit is to take as inputs

- a model of the plant written in standard Modelica
- and some description of the control application (the term "application" being intended in the IEC sense summarised above,

producing as output a single Modelica model, to be fed to any Modelica translator for subsequent simulation.

The question, then, is how to describe the control application.

Basically, one can follow two strategies. One is to describe the IEC languages' elements as Modelica models: this is somehow tempting especially if one considers the graphical IEC languages (FBD, LD, and SFC). Doing so allows to take profit from the manipulation capabilities of the adopted translator, to the apparent advantage of simulation efficiency.

The other strategy is to translate the IEC programs into Modelica algorithms, to be assembled conveniently in blocks, and connected to the plant model in the usual way.

AutoEdit takes the second way, for the reasons summarised in the following. First, especially large can easily lead symbolic manipulator to deal with thousands and thousands of variables: many of them are managed trivially, but the overhead remains. Then, many problems in IEC-specified control systems reside in the incorrect synchronisation of control threads and applications, and therefore – for a credible validation of the control system – representing that timing (e.g., and typically, with when clauses) is very important; if this is done, given the limitations of when-equations, describing the code as algorithms starts looking advisable. In addition, the organisation of the code in threads and sub-applications is typically functional, thus better reflected in algorithms than in equations.

Finally, and in some sense as a by-product, if a tool like AutoEdit generates algorithmic Modelica code starting from an IEC source, then the same tool can easily be extended to generate – from the same source – code in virtually any procedural programming languages. Exploiting that possibility is in the future plans of the AutoEdit project, and will lead to a single tool for the simulation of a complete system (avoiding the "how-to-close-the-loop" problem of IEC development environments) and also for the generation of the control code to be actually deployed to the system's CPU(s).

4. An example

A very simple example is now reported to better illustrate the ideas of section 3. In this example, a home irrigation plant is introduced. The plant has an accumulating tank, a pump, two level sensors, and three valves, each one connected to an irrigation line. A schematic figure of the plant is reported as figure 1.

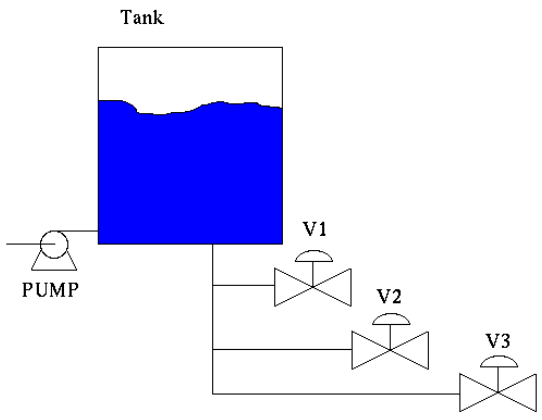


Figure 1: the example plant.

The pump starts pumping water in the tank when the level of the water is lesser than a OK_LEVEL (boolean sensor that returns true if covered by water) level since the water reaches the level FULL_LEVEL (similar boolean sensor). Everyday, say at 20:00 (event launched by a START_CYCLE variable, assumed here to be managed by some clock external to the program), each of the valves (V1 to V3) has to be opened. Each valve, one for each zone, remains opened for 10 minutes and then is closed. There is also a ON/OFF command: if ON is true then the plant works as described below, otherwise all the valves are opened, and the pump is stopped.

Figure 2 shows the overall control program, written in the SFC language, as is appears in the AutoEdit window. It is possible to recognise the various elements of the (very simple) control logic, and to appreciate the similarity of the user interface to that of the typical IEC-compliant environments (to the advantage of acceptability on the part of control system developers). We do not report simulations here since the plant and control operation in this example are very simple, and would not contribute to the purpose of this paper.

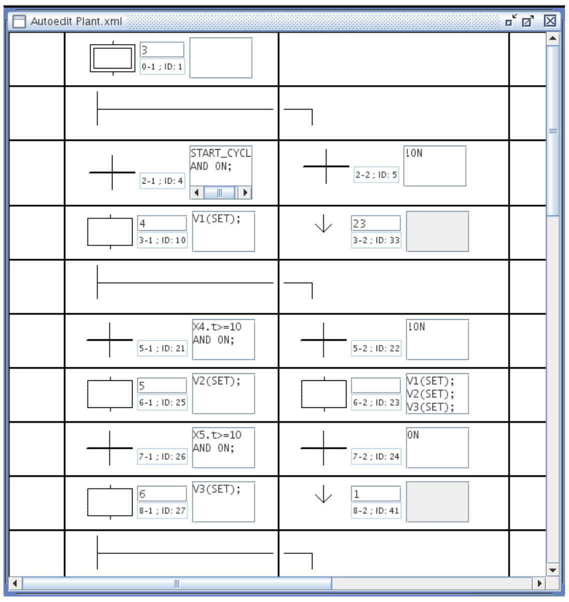


Figure 2: the example plant control in SFC.

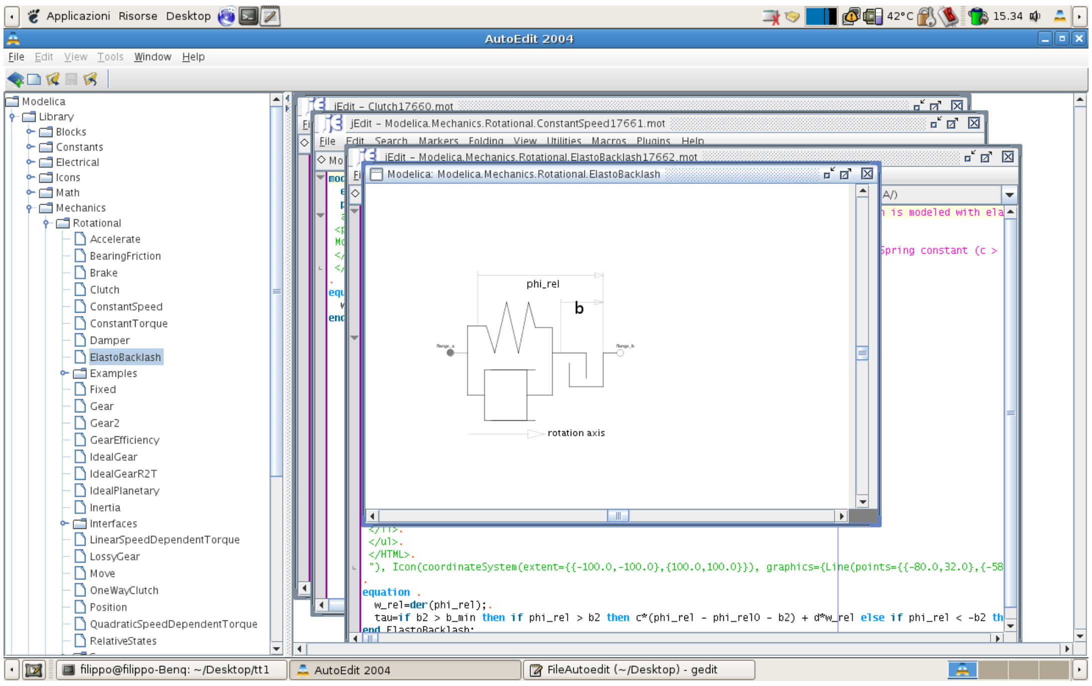
In the example the translation was very simple but, when considering industrial applications of realistic size, the number and length of the lines of control code would increase dramatically, and automatic generation of the algorithmic code would prove necessary. In addition to this if we consider the possibility to have heterogeneous IEC-compliant programs a mixture of ST, LD and SFC implementation, the complexity further increases.

Thanks to the AutoEdit conversion utility, it is possible to translate the SFC programs into LD and then, automatically, to Modelica algorithm-based models. The translation of an heterogeneous IEC control program is perfectly transparent to the AutoEdit user.

4. The AutoEdit project

The project started in the 2004 with the intent to realise a Java graphical application to support graphical programming for the LD, SFC and Structured Text (ST) languages.

From 2004 to 2006 a graphical application was therefore developed to graphically support the SFC and LD programming.



```
</ul>.
</HTML>.
"), Icon(coordinateSystem(extent={{-100.0,-100.0},{100.0,100.0}}), graphics={Line(points={{-80.0,32.0},{-58
.
equation .
  w_rel=der(phi_rel);.
  tau=if b2 > b_min then if phi_rel > b2 then c*(phi_rel - phi_rel0 - b2) + d*w_rel else if phi_rel < -b2 th
end ElastoBacklash;
```

Figure 3: The AutoEdit main window.

Starting from 2006, the target was widened as illustrated in section 3, so as to integrate the AutoEdit environment with a Modelica editor, and then (starting in 2007) to create a converter from SFC, LD XML and Modelica algorithmic-based .mo files. This is – more or less – the present state of the project. Notice that a high development effort is being spent on AutoEdit, so that the mentioned state is continuously changing. The reader is referred to the project site for up-to-date information.

Here, just some samples of the AutoEdit operation are given. Space limitations prevent from reporting here any technical detail, that can anyway be figured out from the site, and will also be available in the system documentation.

Figure 3 shows the main window of AutoEdit with a Modelica model open for editing. It is possible to see the multiple subwindows scheme, allowing simultaneous editing of multiple (process and/or control) models. The AutoEdit text editor, thanks to the integration of the JEdit software, offers many functionalities, among which syntax highlighting, bracket highlighting, text folding (also for annotations), word auto-completion, auto-indentation and many others utilities. of multiple models.

Figure 4, on the other hand, shows the conversion from SFC to LD, namely of the pump control program in the example introduced above. It is possible to appreciate the usefulness of having simultaneous views of the same code with different representations, a facility offered by several IEC-compliant environments, and of high usefulness according to the opinions of the industrial community.

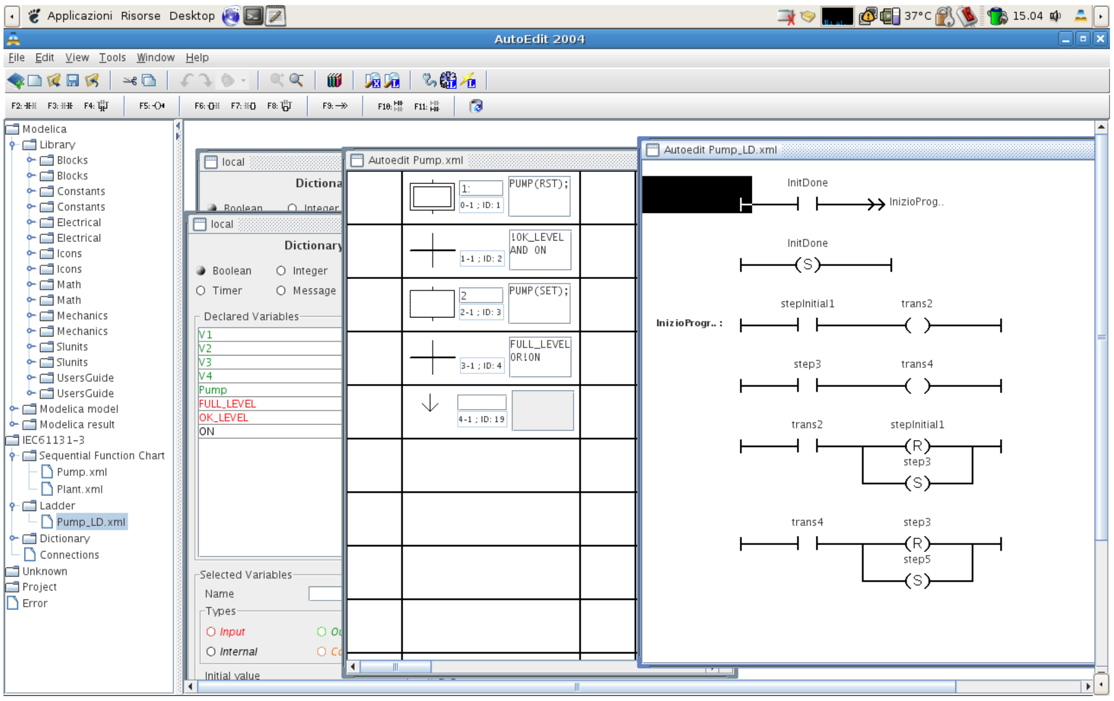


Figure 4: the pump control program converted from SFC to LD by AutoEdit.

5. Future developments

Many interesting “future works” arise for the AutoEdit project from the scenario synthetically described above. Among those possible developments, those that seem more promising, and are therefore scheduled as work to be done in the near future, are

- the development of a 3d viewer for the simulation data,
- the addition of other advanced editing functionalities,
- the exploitation of interaction/integration possibilities with other IEC-compliant tools,
- the output of *ad hoc* real-time code in several languages, the C languages being for obvious reasons the first to be considered,
- the addition of multitasking support.

6. Conclusions

A Java-based integrated environment for the development of complete object-oriented simulation models of controlled plants, namely the AutoEdit project, was presented.

The goal of AutoEdit is to allow the user to create both the plant model, using the power of the Modelica language, and an algorithmic model of the control program, adhering to the IEC61131.3 industry standard,

As such, AutoEdit not only proposes a software solution, but also tries to suggest new standards and ideas for unifying two of the most important activities of the computer-aided engineering tasks: model and control co-simulation.

References

[1] T. Sato, E. Yoshida, Y. Kakebayashi, J. Asakura, N. Komoda, Application of IEC61131-3 For Semiconductor Processing Equipment, Emerging Technologies and Factory Automation. Proceedings. 2001 8th IEEE International Conference on, 2001.

[2] J. Huang, Y. Li, W. Luo, X. Liu, K. Nan, The Design of New-Type PLC based on IEC61131-3, Proceeding of the Second International Conference on Machine Learning and Cybernetics, Xi, 2-5, November 2003.

[3] A. Leva, A. M. Colombo, Method for optimising set-point weights in ISA-PID autotuners, IEE Proc-Control The09 Appl., Vol. 146, No. 2, March 1999 .

[4] H. Takada, H. Nakata, S. Horiike, A Reusable Object Model for Integrating Design Phases of Plant Systems Engineering, Proceedings of the Fourth International Conference on Computer and Information Technology (CIT'04).

[5] H. Taruishil, S. Kajiharal, J. Kawamotol, M. Ono, H. Ohtani, Development of Industrial Control Programming Environment Enhanced by Extensible Graphic Symbols, SICE-ICASE International Joint Conference 2006 in Bexco, Busan, Korea, Oct. 18-2 1, 2006.

[6] Y. Qiliang, X. Jianchun, W. Ping, Water Level Control of Boiler Drum Using One IEC61131-3-Based DCS, Proceedings of the 26th Chinese Control Conference, Zhangjiajie, Hunan, China, July 26-31, 2007.

[7] M. Bonfe', C. Fantuzzi, L. Poretti, PLC Object-oriented programming using IEC61131-3 norm languages: an application to manufacture machinery, in Proc. of IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics, vol. 2, pp. 787-792, 2001.

[8] [Online]. Available http://www.plcopen.org.

[9] J. Roger Folch, J. Pérez, M. Pineda, R. Puche, Graphical Development of Software for Programmable Logic Controllers, 12th International Power Electronics and Motion Control Conference.

[10] [Misc]. DeltaV: Monitor and control software.

[11] [Misc]. Labview: http://www.ni.com/labview.

[12] [Online]. Dymola: http://www.dynasim.se

[13] [Online]. Openmodelica: http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html

[14] A. Nobuo, I. Kenichi, Y. Eiji, Application portfolios for stardom, 12th International Power Electronics and Motion Control Conference.

[15] M. Otter, K. E. Årzén, I. Dressler, StateGraph-A Modelica Library for Hierarchical State Machines, 4th International Modelica Conference, March 7-8, 2005.

[16] O. Johansson, A. Pop, P. Fritzson, Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools, 5th International Modelica Conference, September 4-5, 2006.

[17] E. Tisserant, L. Bessard, M. de Sousa, An Open Source IEC 61131-3 Integrated Development Environment, Industrial Informatics, 5th IEEE International Conference on, 2007.

[18] [Online]. ISaGRAF: http://www.icpdas.com/products/PAC/i-8000/isagraf.htm

Exception Handling for Modelica

Adrian Pop, Kristian Stavåker, Peter Fritzson
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{adrpo, krsta, petfr}@ida.liu.se

Abstract

Any mature modeling and simulation language should provide support for error recovery. Errors might always appear in the runtime of such languages and the developer should be able to specify alternatives when failures happen. In this paper we present the design and implementation of exception handling in Modelica. To our knowledge this is the first approach of integrating equation-based object-oriented languages (EOO) with exception handling.

Keywords: Exception handling, Modelica.

1 Introduction

According to the terminology defined in IEEE Standard 100 [9], we define an *error* to be something that is made by humans. Caused by an error, a *fault* (also *bug* or *defect*) exists in an artifact, e.g. a model. If a fault is executed, this results in a *failure*, making it possible to detect that something has gone wrong.

Approaches to statically prevent and localize faults in equation-based object-oriented modeling languages are presented in [16] and [17]. However, in this paper we focus on language mechanisms for dynamically handling certain classes of faults and exceptional conditions within the application itself. This is known as exception handling. An exception is a condition that changes the normal flow of control in a program.

Language features for exception handling are available for most modern programming languages, e.g. object oriented languages such as Java [15], C++ [14], and functional languages such as Haskell [3], OCaml [12], and Standard ML [13].

However, exception handling is currently missing from Object-Oriented Equation-Based (EOO) Languages like Modelica [2][6], VHDL-AMS [10], gPROMS [11].

A short sketch of the syntax of exception handling for Modelica was presented in a paper on Modelica Metaprogramming extensions [5], but the design was incomplete, not implemented, and no further work was done at that time.

The design of exception handling capabilities in Modelica is currently work in progress. The following constructs are being proposed:

- A `try...catch` statement or expression.
- A `throw (...)` call for raising exceptions.

We have tried to keep the design of syntax and semantics of exception handling in Modelica as close as possible to existing language constructs from C++ and Java, while being consistent with Modelica syntax style.

2 Applications of Exceptions

In this section we provide examples of exception handling usefulness. There are three contexts in which exceptions can be thrown and caught: expression level, algorithm level and equation level.

```
import Modelica.Exceptions=Exn;

function log
 input Real x;
 output Real y;
algorithm
 y :=
 if x <= 0
 then
  throw (Exn.InvalidArgumentException(
         message="Logarithm is undefined
                  for ..."))
 else
   Modelica.Math.log(x);
end log;
```

Function `log` defined above will throw an exception if it is provided with an invalid argument. This is not only useful for mathematical functions, but also for functions (i.e. like the ones in `Modelica.Utilities` package) that deal with errors due to the operating system. A common for all tools standard hierarchy of exceptions could be defined in the Modelica Standard Library for all the exceptions categories needed. Depending on the simulation runtime implementation (i.e. language of choice) of the Modelica tool exceptions could be translated from Modelica to the runtime and back.

A model that uses the try-catch construct in the expression and equation contexts is presented below:

```
model Test
 // try to read a value from file
 // and if it fails just give it
 // a default value.
 parameter Real p=
       try
        readRealParameter("file.txt","p")
       catch(Exn.IOException e)
        0
       end try;
 Real x;
 Real y;
equation
  try
    y = log(x);
  catch(Exn.InvalidArgumentException e)
    // terminate the simulation with
    // a message on what went wrong
    terminate(e.message);
  end try;
end Test;
```

In this model exception handling in expressions and equations are shown. In the case of exception handling in equations the example just terminates the simulation with an exception.

As one may have noticed the exceptions can be thrown during:

- Compilation time for expressions or functions that are evaluated at compile time
- Simulation time, due to exceptions raised into the solver, functions, expressions or equations.

All the exceptions raised during compile time are reported to the user. The exceptions which are caught are reported as warnings and the un-caught ones are reported as errors.

3 Exception Handling

In this section we present the design of the exception handling constructs. The grammar of the try-catch constructs is given below. The grammar follows the style from the Modelica Specification [6] and uses constructs defined there. Different try clauses for each of the expression, statements and equations contexts are defined.

```
exception_declaration:
  type_specifier IDENT
  ["(" exception_arguments ")"]

exception_arguments:
   expression
   [ "," exception_arguments ]
 | named_arguments

named_arguments:
  named_argument [ "," named_arguments ]

named_argument:
  IDENT "=" expression

name:
  IDENT [ "." name ]

throw_clause:
  throw ["(" name
  [ "(" exception_arguments ")"] ")" ]

try_clause_expression:
  try
    expression
  ( else_catch_clause_expression
    | catch_clause_expression
      { catch_clause_expresion }
      [ else_catch_clause_expression ] )
  end try

catch_clause_expression:
  catch "(" exception declaration ")"
    expression

else_catch_clause_expression:
  elsecatch
    expression

try_clause_algorithm:
  try
    { statement ";" }
    ( else_catch_clause_algorithm
      | catch_clause_algorithm
        { catch_clause_algorithm }
        [ else_catch_clause_algorithm ] )
  end try

catch_clause_algorithm:
  catch "(" exception declaration ")"
    { statement ";" }

else_catch_clause_algorithm
  elsecatch
    { statement ";" }

try_clause_equation
  try
    { equation ";" }
  ( else_catch_clause_equation
    | catch_clause_equation
      { catch_clause_equation }
      [ else_catch_clause_equation ] )
  end try

catch_clause_equation:
  catch "(" exception_declaration ")"
    { equation ";" }

else_catch_clause_expression:
  elsecatch
    { equation ";" }
```

Throwing via `throw;` without any parameter can only appear inside the catch clause and will throw the currently caught exception. This constraint is not specified in the above grammar to keep it simple. Of course, it could be also checked by the semantics phase.

The try-catch clauses shown here are part of the various contexts rules in Modelica grammar: expressions, algorithm and equation.

3.1 Exception Handling for Statements

The statement variant has approximately the following syntax:

```
try
  <statements1>
catch(<exception_declaration>)
  <statements2>
end try;
```

The semantics of a try-catch for statements is as follows: An exception generated from a failure during the execution of `statements1` will lead to the execution of `statements2` if the exception matches the catch clause.

3.2 Exception Handling for Expressions

The syntax of the expression variant is as follows:

```
try
  <expression1>
catch(<exception_declaration>)
  <expression2>
end try;
```

The semantics of a try-catch for expressions is as follows: An exception generated from a failure while executing `expression1` will lead to the execution of `expression2` if the exception matches the catch clause.

3.3 Exception Handling for EOO

What does it mean to have exception handling for equation-based models? For example, if an uncaught exception, e.g. division by zero, occurs in any of the expressions or statements executed during the solution of the equation-system generated from the model, the catch could handle this, e.g. by simulating an alternative model (providing alternate equations), or stopping the simulation in a graceful way, e.g. by an error-message to the user. *The number of equations within the try construct must be the same as the number of equations in the catch part. This restriction is needed because models must be balanced. Of course, the restriction does not apply for the catch parts that only terminates the simulation and reports an error.*

The syntax of the equation variant is as follows:

```
try
  <equations1>
catch(<exception_declaration>)
  <equations2> | <terminate(...)>
end try;
```

The semantics of a try-catch for equations is as follows: If a failure generating an exception occurs during the solution of the equations in the set of equations denoted `equations1`, then if the catch matches the raised exception, then instead the `equations2` set is solved.

The source of the exception can be in the expressions and functions called in `equations1`, which are evaluated during the solving process. Certain exceptions might originate from the solver. In that case, a few selected solver exceptions need to be standardized and predefined.

The semantics of try-catch for equations is similar to the one for if-equations, with the difference that the event triggering the catch block is when an exception is thrown.

There could be several semantics for try-catch in equation section and they are discussed in Section 8.

3.4 Exception Handling and external functions

The compiler should be able to check the exceptions in order to:

- Report an error if the catch part tries to catch an exception that will never be thrown.
- Report exceptions that are not caught anywhere
- Generate efficient code for exceptions

The compiler can find automatically at compilation time what exceptions are thrown from models and functions defined in Modelica. However, the compiler must be provided with additional help when it comes to external functions. Therefore, when declaring external functions, the exceptions that might be thrown by them have to be declared too.

We could model this additional information in two ways: directly in the grammar or as annotations.

Directly in the grammar as part of the `element_list` (check the Modelica grammar for the element list specification) of the function or model:

```
throws_declaration:
   throws name { "," name } ";"
```

Is not really needed to specify in the grammar the possible exceptions to be thrown, we could use annotations instead:

```
annotation(throws={name1, name2, ... };
```

Names used above are constructed according to `name` grammar rule specified in the beginning of this section.

In the literature this feature of the compiler (or the language) is called *Checked exceptions* [18].

4 Transforming matchcontinue Fail Semantics

The current MetaModelica language extension has a simple fail semantics: fail exceptions can be thrown explicitly (via a `fail()` call) or implicitly (e.g., via a failure due to no patterns matching in a called function), and be caught/handled within the subsequent case(s) in the `matchcontinue` construct matching the same pattern.

The `matchcontinue` construct can be transformed into a match-expression that does not have the continue semantics after a failure, however requiring that the fail exception is caught in the same case branch.

Example:

```
matchcontinue x local ...
case Plus(a,b) equation   // raise
    ...generateFailureException...
case Plus(a,b) equation   // Catch
    handleFailure(a,b)
case _ handle_All_Inclusive_case();
end matchcontinue
```

can be transformed into the following:

```
match x local ...
case Plus(a,b) equation
  try
    ...generateFailureException...
  catch(Fail fail)
    handleFailure(a,b)
  end try;
case _ handle_All_Inclusive_case();
end match;
```

This transformation will be supported by a refactoring tool to transform existing code based on `matchcontinue` constructs into faster and clenrer code based on the `match` construct combined with exception handling. Such transformation will speed up the OpenModelica compiler, by removing many uses of matchcontinue with repeated matching due to overlapping patterns.

5 Exception Values

In this section we discuss different ways of representing exception values in Modelica. In general exceptions are values of a user defined type. Certain exceptions, such as `DivisionByZero` or `ArrayIndexOutOfBounds` are predefined. The user should be able to define exceptions hierarchically (i.e. packages of exceptions) and use inheritance to add extra information (components) to existing exceptions, thus creating specialized exceptions.

5.1 Exceptions as Types

We can model exceptions as a built-in Modelica type Exception. A pseudo-class declaration of such a type and its usage would look like:

```
type Exception
  // the value of the exception is
  // a string, accessed directly
  StringType 'value'
end Exception;

// Defining a new exception
type E1
  extends Exception;
end E1;

// Instantiate new exception
E1 e1 = "exception E1";
// Raise new exception
throw e1;

// Adding more information to an exception
type E2
  extends E1;
  parameter String moreInfo;
end E2;

// Instantiate the exception
E2 e2(moreInfo="E2 add") = "exception E2";

 // Throw exception
throw(e2);

try
  ...
catch(E2 e2)
  // here you can access the
  // e2 value directly
  // but you cannot access e2.moreInfo

catch(E1 e1)
  // here you can access the
  // value of e1 directly
end try;
```

Because we extend a basic type, it is possible to add more information to the exception, but this information cannot be accessed via dot notation.

5.2 Exceptions as Records

Another way to model exceptions is as Modelica records.

```
record Exception
  parameter String message;
end Exception;

// defining a new exception
record E1
  extends Exception(message="E1");
  parameter String moreInfo;
end E1;

// instantiate new exception
E1 e1(moreInfo="More Info");

// raise new exception
throw(e1);
```

```
// Try and catch
try
  ...
catch (E1 e1)
  // here you can access e.message
  // and e.moreInfo
catch (Exception e)
  // here you can access e.message
end try;
```

Modeling exceptions as records has many of the desired properties that a user might want. The problems we see here are that:

- Is not very intuitive to throw and catch arbitrary records.
- The hierarchical structure is partly lost during flattening, which means that for the records used in the throw/try-catch constructs this information should be preserved.
- The inheritance hierarchy is flattened for records and one would like to keep it intact to be able to catch exceptions starting from very specific (at the bottom of the inheritance hierarchy) to more general (at the top of the inheritance hierarchy)

We think that a better approach is with a new restricted Modelica class called `exception`.

5.3 New Restricted Class: exception

We believe that the best way to model exceptions in Modelica is by extending the language with a new restricted class called exception. Moreover, similar design choices have been made in Java or Standard ML, with their predefined exception types. In Java one can only throw objects of the `java.lang.Throwable` and its superclass `java.lang.Exception`. The C++ language allows throwing of values of any type. In Standard ML and OCaml exceptions values and their type need to be defined using a special syntax.

Exceptions can be represented in Modelica as a new restricted class in the following way:

```
exception E1
  parameter String message;
end E1;

E1 e1(message="More Info");
throw(e1); // raise new exception

// defining a new exception
exception E2
  extends E1(message="E2");
  parameter String moreInfo;
end E2;

// instantiate new exception
E2 e2(moreInfo="More Info");
throw(e2); // raise new exception

try
  ...
catch(E2 e2)
  // here you can access e.message
  // and e.moreInfo
catch(E1 e1)
  // here you can access e.message
end try;
```

Having a specific restricted class for exceptions would have the following advantages:

- Throwing and catching only values of restricted class exception is more intuitive than using records.
- Both the structural hierarchy and the inheritance hierarchy of the exceptions can be kept during flattening and translated to C++, Java, Standard ML or OCaml code more easily.
- The type checking of throw and try-catch constructs would be more specific and straightforward.

6 Typing Exceptions

Modelica features a structural type system, which means that two structures can be in the subtype relationship even if they have no explicit inheritance specified between them.

The type checking procedure for exceptions has to be different than for all the other constructs, namely:

- Only restricted classes of type exception can be thrown.
- When elaborating declarations of restricted class exception the subtype relationship applies only if there is *specific inheritance relation* between exceptions. This is needed because the exceptions have to be matched by name and have to be ordered so that the most specific case (supertype) is first and the least specific (subtype) is last in a catch clause.
- When translation declarations of restricted class exception there will be *no flattening of the inheritance hierarchy*.
- When elaborating catch clauses the compiler has to: i) *match the exception by name*, ii) *reorder the catch clauses in the inverse order of the inheritance relation* between exceptions or give an error if the less specific exceptions are matched before the more specific ones.
- The compiler has to check if an exception specified in the catch clause will actually be thrown from the try body or not. If such exception is not thrown the compiler can either discard the catch clause or issue a warning/error at that specific point.

With these new rules the typing of exception declarations, exception values and catch clauses can be achieved. After the translation, the runtime system and the language in which was implemented (C++, Java, Standard ML) will provide the rest of the checking for exceptions.

7 Implementation

In this section we briefly present the OpenModelica implementation of exception handling. When referring to Exception Hierarchy we mean both the structural hierarchy and the inheritance hierarchy.

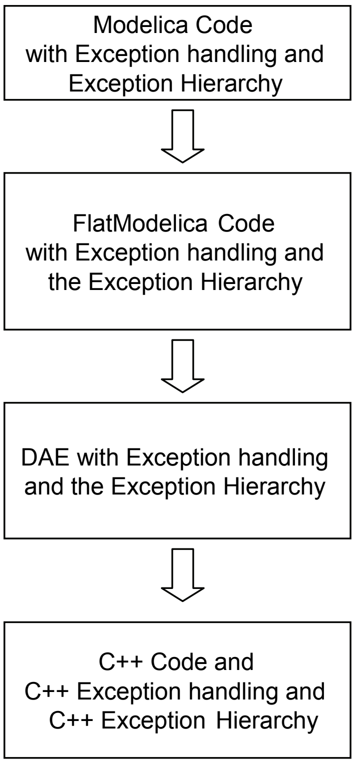

Modelica Code
with Exception handling and
Exception Hierarchy
FlatModelica Code
with Exception handling and
the Exception Hierarchy
DAE with Exception handling
and the Exception Hierarchy
C++ Code and
C++ Exception handling and
C++ Exception Hierarchy

Figure 1. Exception handling translation strategy.

7.1 Overview

The general translation of Modelica with exception handling follows the path described in **Error! Reference source not found.**. The exception handler and the exception hierarchy are passed through the compiler via the intermediate representations of each phase until the C++ code is generated (or any other language code used in the backends of different Modelica compilers).

The specific OpenModelica translation path for Modelica code with exception handling is presented in Figure 2.

Exception handling in OpenModelica required the following extensions:

- The parser was extended with the proposed exception handling grammar.
- Each intermediate representation of the OpenModelica compiler was augmented with support for exceptions.

Both the structural and the inheritance hierarchy of the exceptions are passed through the OpenModelica compiler until C++ code is generated.

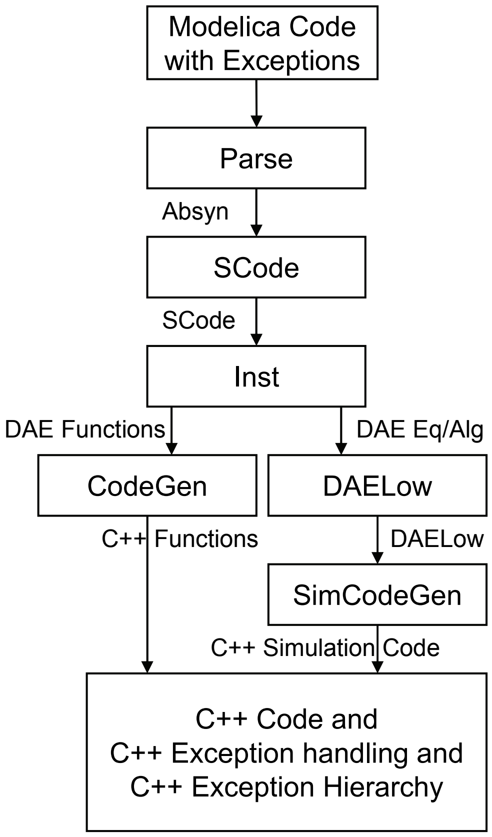


Figure 2. OpenModelica implementation.

7.2 Translation of Exception values

The translation from the internal representation to C++ code is straightforward: a Modelica exception maps to a C++ class. For example, the following Modelica code with exceptions:

```
exception E
  parameter String message;
end E;

exception E1
  extends E(message="E1");
  parameter Integer id = 1;
end E1;
```

is translated into the following C++ code:

```
class E
{
  public:
  modelica_string message;
  E(modelica_string message_modification)
  {
    message = message_modification;
  }
  E()
  { message = ""; }
}
```

```
class E1 : public E
{
  public:
  modelica_integer id;
  E1(modelica_string message_modification,
     modelica_integer id_modification)
  {
    message = message_modification;
    id = id_modification;
  }
  E1()
  {
    message = "E1";
    id = 1;
  }
}
```

The following Modelica code for exception instantiation and exception throwing:

```
E  e;  throw(e);
E1 e1; throw(e1);

E1 e2(message="E2", id=2);
throw(e2);

E1 e3(message="E3");
throw(e3);
```

is translated to the following C++ code:

```
E  *e  = new E();  throw e;
E1 *e1 = new E1(); throw e1;

E1 *e2 = new E1("E2", 2);
throw e2;

E1 *e3 = new E1();
e3->message = "E3";
throw e3;
```

Is also possible to represent exception values in C++ as objects allocated on the stack, i.e.: `E1 e2("E2", 2);`.

7.3 Translation of Exception handling

The C++ exception handling code follows the Modelica code. The table below defines the translation procedure for Modelica including the MetaModelica extensions.

Modelica Expressions	**C++**

`x :=`
`try`
`  exp1`
`catch``(E e)`
`  exp2`
`end try``;`
	`modelica_type temp;`
	
	`try`
	`{`
	`  temp = exp1;`
	`}`
	`catch``(E *e)`
	`{`
	`  temp = exp2;`
	`}`
	`x = temp;`

Modelica Statements	**C++**

`try`
`  <statements>`
`catch``(E e)`
`  <statements>`
`end try``;`
	`try`
	`{`
	` // Modelica`
	` // corresponding`
	` // C++ statements`
	`}`
	`catch``(E *e)`
	`{`
	` // Modelica`
	` // corresponding`
	` // C++ statements`
	`}`

Modelica Equations	**C++**

`try`
`  <eqnsA>`
`catch`` (Ex1 e1)`
`  <eqnsB>`
`end try``;`
 	
`try`
`  <eqnsC>`
`catch`` (Ex2 e2)`
`  <eqnsD>`
`end try``;`
	`event1=false;`
	`event2=false;`
	
	`while`` time < stopTime`
	`{`
	` `**`try`**`{`
	`  `*`call SOLVER for problem:`*
	`  `**`if`**` event1`
	`  `**`then`**
	`    eqnsB;`
	`  `**`else`**
	`    eqnsA`
	`  `**`end if`**`;`
	
	`  `**`if`**` event2`
	`    eqnsD;`
	`  `**`else`**
	`    eqnsC;`
	`  `**`end if`**`;`
	` }`
	` `**`catch`**`(Ex1 *e1)`
	` `**`{`**
	`  `*`discard posible`*
	`  `*`calculated current`*
	`  `*`step values;`*
	`  `*`reinit the solver`*
	`  `*`with previous step`*
	`  `*`values;`*
	`  event1 = true;`
	` }`
	` `**`catch`**`(Ex2 *e2)`
	` `**`{`**
	`  `*`discard posible`*
	`  `*`calculated current`*
	`  `*`step values;`*
	`  `*`reinit the solver`*
	`  `*`with previous step`*
	`  `*`values;`*
	`  event2 = true;`
	` }`
	`}`

8 Further Discussion

During the design and implementation of exception handling we have encountered various issues which we will present in this section. The exception handling in expressions and algorithm sections are straightforward. However when extending exception handling for equation sections there are several questions which influence the design choices that come to mind:

Questions: *Is the exception handling necessary for equation sections? If yes, what are the semantics that would bring the most usefulness to the language?*

Answers: We believe that exception handling is necessary in the equation sections at least to give more useful errors to the user (i.e. with `terminate(message)` in the catch clause) or to provide an alternative for gracefully continuing the simulation. Right now in Modelica there is no way to tell where a simulation failed. There are assert statements that provide some kind of lower level checking but they do not function very well in the context of external functions. As example where alternative equations for simulation might be needed we can think of the same system in different level of detail. Where the detailed system can fail due to complexity and numerical problem the simulation can be continued with the less complex system.

Semantics of try-catch in equation sections

Several semantics can be employed to deal with try-catch clauses in equation sections:

1. Terminate the simulation with a message (as we show in application section)
2. Continue the simulation with the alternative equations from the catch clause activated and the ones from the try-body disabled. When the exception occurs the calculated values in that solver step are discarded and the solver is called again with previous values and the alternative from the catch clause.
3. Signal the user that an exception occurred and restart the simulation from the beginning with the catch-clause equations activated.
4. When an exception occurs, discard the values calculated in the current step and activate the alternative equations from catch-clause. However, at the next step try again the equations from the try-body. This will make the catch-clause equation active only for the steps where an error might occur.

We think that the most useful design for exception handling in equation section is the one that has both features 1 and 2 active.

9 Conclusions

We have presented the design and the implementation of exception handling for Modelica. We strongly believe in the need for a well designed exception handling in Modelica. By adding exception handling constructs to the language we get a more complete language and provide the developer with means to better control exceptions. There are several issues that have to be considered when designing and implementing these constructs which we have discussed in this paper.

10 Acknowledgements

This work has been supported by Swedish Foundation for Strategic Research (SSF), in the RISE and VISIMOD projects, by Vinnova in the Safe and Secure Modeling and Simulation project.

References

[1] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. Simulation News Europe, 44/45, Dec 2005. http://ww.ida.liu.se/projects/OpenModelica

[2] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., Wiley-IEEE Press, 2004. See also: http://www.mathcore.com/drmodelica

[3] Paul Hudak. The Haskell School of Expression. Cambridge University Press, 2000.

[4] Kenneth C. Louden: Programming Languages, Principles and Practice. 2:nd edition, Thomson Brooks/Cole, 2003, (ISBN 0-534-95341-7)

[5] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005.

[6] The Modelica Association. The Modelica Language Specification Version 3.0, September 2007. http://www.modelica.org.

[7] Mikael Pettersson. Compiling Natural Semantics. PhD thesis, Linköping Studies in Science and Technology, 1995.

[8] Peter van Roy and Seif Haridi. Concepts, Techniques, and Models of Computer Programming. MIT Press, 2004.

[9] IEEE Standards Information Network. IEEE 100 The Authoritative Dictionary of IEEE Standards Terms. IEEE Press, New York, USA, 2000.

[10] Christen E. and K. Bakalar. VHDL-AMS-a hardware description language for analog and-mixed-signal applications, In 36th Design Automation Conference, June 1999

[11] Oh Min and C.C. Pantelides (1996) "A Modeling and Simulation Language for Combined Lumped and Distributed Parameter System." Computers & Chemical Engineering, vol 20: 6-7. pp. 611-633 1996.

[12] Xavier Leroy et al., The Objective Caml system. Documentation and user's manual, 2007, http://caml.inria.fr/pub/docs/manual-ocaml

[13] Robin Milner, Mads Tofte, Robert Harper and David MacQueen, The Definition of Standard ML, Revised Edition, MIT University Press, May 1997, ISBN: 0-262-63181-4

[14] Bjarne Stroustrup: The C++ Programming Language (Special Edition). Addison Wesley. Reading Mass. USA. 2000. ISBN 0-201-70073-5. 1029 pages. Hardcover.

[15] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. The Java™ Language Specification, Third Edition, ISBN-13: 978-0321246783, Prentice Hall, June 2005.

[16] Peter Bunus. Debugging Techniques for Equation-Based Languages. Ph.D. Thesis No. 873, Linköping University, 2004

[17] David Broman. Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments. Licentiate Thesis, Thesis No. 1337, Linköping University, December, 2007.

[18] Exception Handling: http://en.wikipedia.org/wiki/Exception_handling

Session 4b

Thermodynamic Systems & Applications

Modeling of the Gasification Island with Modelica

Julia Fahlke[1] Stephan Püschel[1] Frank Hannemann[2] Bernd Meyer[1]
[1]TU Bergakademie Freiberg, Institut für Energieverfahrenstechnik und Chemieingenieurwesen
Fuchsmühlenweg 9, Haus 1, 09596 Freiberg
[2]Siemens Fuel Gasification Technology GmbH
Halsbrücker Straße 34, 09599 Freiberg

Julia.Fahlke@iec.tu-freiberg.de

Abstract

For the modeling and simulation of the Gasification Island a new Modelica library *GasificationIsland* was developed. Therefore new components had to be generated, like the gasifier or the components of the pneumatic feeding system. The developed models are based on the Modelica_Fluid and the Modelica.Media libraries. In this paper the structure of the most important component models and the main modeling assumptions are illustrated.

Keywords: Gasification Island modeling; SFG

1 Introduction

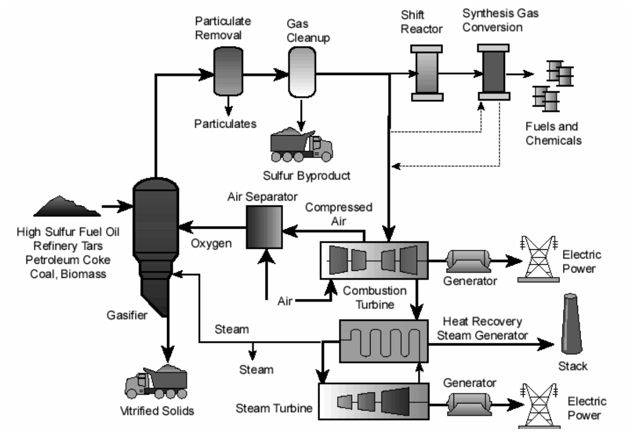


Figure 1: usage of the syngas from the gasification [1]

The gasification process is of great importance for the electrical and basic chemical industry as it converts any carbon-containing material into a synthesis gas (syngas) composed primarily of carbon monoxide and hydrogen. This syngas can be used as a fuel in a combined cycle to generate electricity (**I**ntegrated **G**asification **C**ombined **C**ycle). But it can also be used as a basic chemical for a large number of syntheses in the petrochemical and refining industry, like Methanol or Fischer Tropsch Synthesis (Figure 1). The modeling of the gasification process is till this day a great challenge.

1.1 The gasification process

Gasification means the thermo-chemical conversion of fuels with a reactant to a combustible gas, which is rich of the components CO, H_2 and CH_4. The most proceeded reactions are partial oxidation procedures, which take place with oxygen in free (elemental) or bounded form (H_2O, CO_2). These partial oxidations are interfered in dependence of the process and the process parameters with pyrolysis or devolatilization and hydrogenation processes [2].

The gasification process can be divided into different types according to the gasification agent/heat supply (autothermic, allothermic or hydrogenating gasification), the gas-solid-contacting (fixed/moving bed, fluidized bed or entrained flow gasification) or concerning the process temperature (above or below ash melting point).

This paper deals with the SFG gasification process, which is an autothermic, entrained flow gasifier with temperatures in the gasifier above the ash melting point.

In the gasification process a large number of reactions take place. Principle chemical reactions are those involving carbon, carbon monoxide, carbon dioxide, hydrogen, water (or steam) and methane [3]:

combustion reactions

$C + 0.5O_2 \rightarrow CO$	-111 MJ/kmol
$CO + 0.5O_2 \rightarrow CO_2$	-283 MJ/kmol
$H_2 + 0.5O_2 \rightarrow H_2O$	-242 MJ/kmol

Boudouard reaction

$C + CO_2 \leftrightarrow 2CO$ +172 MJ/kmol

water gas reaction

$C + H_2O \leftrightarrow CO + H_2$ +131 MJ/kmol

methanation reaction

$C + 2H_2 \leftrightarrow CH_4$ -75 MJ/kmol

CO shift reaction

$CO + H_2O \leftrightarrow CO_2 + H_2$ -41 MJ/kmol

steam methane reforming reaction

$CH_4 + H_2O \leftrightarrow CO + 3H_2$ +206 MJ/kmol

Most fuels contain additional components beside carbon, hydrogen and oxygen, e.g. sulfur, nitrogen or minerals. Sulfur in the fuel is converted into H_2S and COS and the nitrogen into elemental nitrogen, NH_3 or HCN.

1.2 The SFG Gasification Island

The SFG Gasification Island consists of the SFG gasifier itself, the pneumatic feeding system and the gas treatment system (Figure 2).

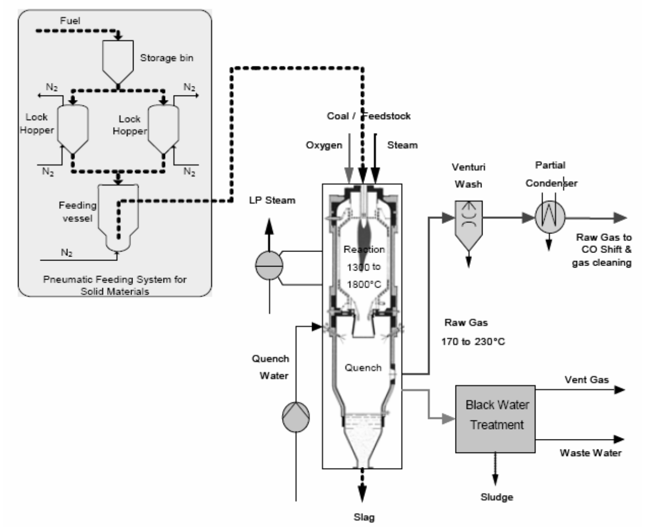


Figure 2: SFG Gasification Island [4]

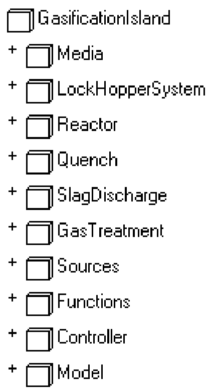
The solid fuel (e.g. coal) is fed into the SFG-Reactor through a pneumatic feeding system. In the reactor the carbon rich fuel will be partially oxidized under high pressure and under the addition of oxygen as gasifying agent and steam as temperature moderator into a raw gas. Minerals in the fuel are separated and leave the bottom of the gasifier as an inert glass-like slag. The raw gas is cooled down and saturated in the quench. Afterwards it flows in the venturi wash and in the partial condenser, where the raw gas is cooled down and the solid particles are separated from the raw gas.

2 *GasificationIsland* Library Overview

The *GasificationIsland* library is an in-house Modelica library for the transient simulation of the Gasification Island process. The library is designed in a joint project with Siemens Fuel Gasification Technology GmbH Freiberg. The intention of the project is to apply these models to analyze the behavior of the different sub-processes as well as the whole Gasification Island at load changes or disturbances and to test new control strategies (see chapter 4). Furthermore the library shall be utilized in the plant shop tests prior real commissioning on site.

The developed models are based on the Modelica_Fluid and the Modelica.Media libraries.

The library is divided into functional sub-packages. In Figure 3 a screen shot of the first hierarchical level of the library is shown.



GasificationIsland
+ Media
+ LockHopperSystem
+ Reactor
+ Quench
+ SlagDischarge
+ GasTreatment
+ Sources
+ Functions
+ Controller
+ Model

Figure 3: screen shot of the GasificationIsland library in the package browser

The **Media** package contains all the used media models like raw gas, slag or coal. The solids are simulated as media with constant properties, like the *ConstantPropertyLiquidWater* in the Modelica.Media library.

The package **LockHopperSystem** includes all component models of the pneumatic feeding system, e.g. lock hopper, storage bin or feeding vessel.

The packages **Reactor/Quench** comprise the models of the gasification reactor and the quench.

The **SlagDischarge** package includes the models of the slag hopper and the flushing tank.

The **GasTreatment** package contains models of the venturi scrubber, partial condenser and drums.

The **Controller** package includes the sequence control of the batch processes for the pneumatic feeding system and the slag discharge system. The controllers were modeled by the Modelica.StateGraph library.

The package **Model** comprises the different simulated sub processes of the gasification island and a model of the complete process.

In the following section the structure and the main modeling assumptions for some selected components will be illustrated.

3 Developed Models

3.1 Lock Hopper System

3.1.1 Storage bin

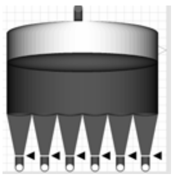


Figure 4: screen shot of the storage bin icon

The storage bin should ensure the uninterrupted service of the gasification reactor with coal. The storage bin works at ambient pressure.

In practice the pneumatic feeding system consists of more then one lock hopper, so the storage bin needs as many outlets as lock hoppers exist. For solid flow the outlet form is conical (Figure 4). To simulate the filling level of this geometrical form the storage bin was divided into segments. The single segments are connected through valves. These valves have a huge *Kv* flow coefficient and ensure the mass flow between the segments. So it can be guaranteed that the filling level in each storage bin segment is the same. There is only one exception: if the filling level is lower than the high of the cone no more mass transfer between the segments occurs. This is realized by setting the outlet pressure of the segment connections to a defined minimum value. Furthermore the inlet flow is split to the segments. Figure 5 shows the implementation of a storage bin with six outlets in the Dymola Diagram Layer.

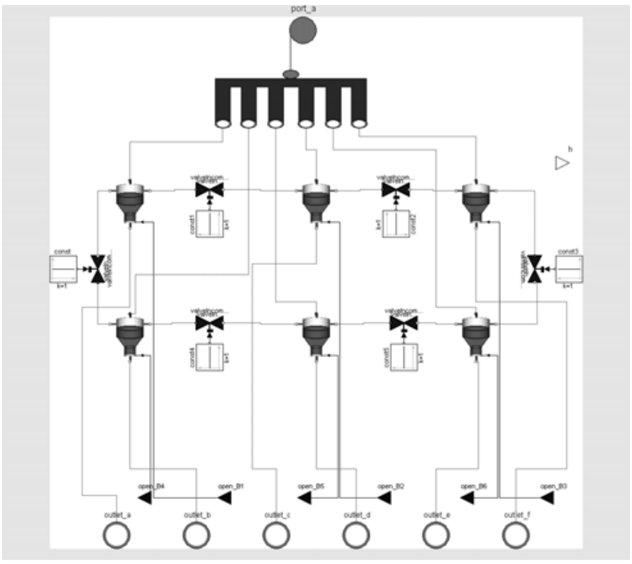


Figure 5: screen shot of the Implementation of a storage bin with 6 outlets in the Dymola Diagram Layer

For the outlet ports the coal mass flow is defined. The following function is used [5]:

$$\dot{m}_{c,out} = 0.3 \cdot \left(1 - e^{-0.1 \cdot \frac{d_A}{d_P}}\right) \cdot \frac{\rho_P \cdot g^{0.5} \cdot d_A^{\;2.5}}{\tan(\gamma) \cdot \beta^{0.36}} \cdot h^{0.5} \cdot \mu$$

There d_A is the diameter of the lock hopper outflow, d_P is the mean diameter of the coal particle, ρ_P is the particle density, γ is the repose angle, β is the cone angle and h is the fill level of the coal.

3.1.2 Lock Hopper

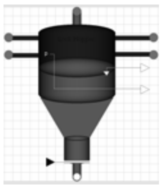


Figure 6: screen shot of the lock hopper icon

For dosing of the pulverized coal into the reactor it is necessary to bring it into a pressure system that operates at a pressure level higher than the reactor pressure. This is fulfilled by the lock hoppers. Therefore 4 sequences appear:

- filling of the lock hopper with coal until the maximum level is reached
- pressurizing of the coal. Therefore a pressurized inert gas is fed into the lock hopper
- discharging of the lock hopper into the feeding vessel
- depressurizing of the gas

In the lock hopper are two different media: the gas and the coal medium. For each a mass balance is considered but only one energy balance is implemented. Furthermore the wall material is regarded as a heat storage system and convective heat transfer between the gas and the wall is implemented.

The level of the coal is determined by the fixed bulk density.

```
model LockHopper
   …
  // Total quantities
  m_coal = V_coal*coal.d;
  m_gas = (V-V_coal)*gas.d;
  U = coal.u*m_coal + gas.u*m_gas;
  U_wall = m_wall*cp_wall*T_wall;
  V_bulk = m_coal/rho_bulk;
  Q = alpha*A*(T_wall – gas.T);

  //Mass balances
  der(m_coal) =  in_c.m_flow + out_c.m_flow;
  der(m_gas) = sum(in_g.m_flow) + sum(out_g.m_flow);

  //Energy balances
  der(U) =  in_c.H_flow + out_c.H_flow + sum(in_g.H_flow)
  + sum(out_g.H_flow) + Q;
  der(U_wall) = -Q;
  …
end LockHopper;
```

3.1.3 Feeding Vessel

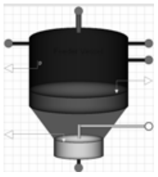


Figure 7: screen shot of the feeding vessel icon

The pulverized coal is fed from the feeding vessel into the gasification reactor. Therefore the feeding vessel remains a constant level of operating pressure above the reactor pressure. This is done through pressurizing and depressurizing of the feeding vessel with inert gas.

In the feeding vessel exist three layers: the fluidized bed, the bulk and the gas layer (Figure 8).

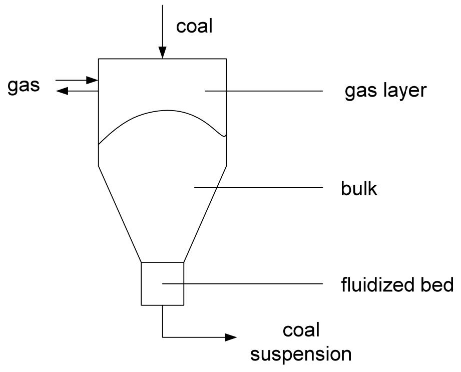


Figure 8: layers in the feeding vessel

From the fluidized bed the coal suspension is fed to the reactor. It is assumed that the height of the fluidized bed is fixed. Furthermore a functional correlation among the pressure drop between the feeding vessel and the reactor and the coal mass flow to the reactor exists. This functional correlation can be lodged.

In the feeding vessel only one energy balance is considered. The wall material as heat storage system is neglected because the appeared temperature fluctuations are only small.

3.2 Reactor

3.2.1 Gasification Reactor

In the gasification reactor the conversion of the coal into a combustible raw gas occurs.

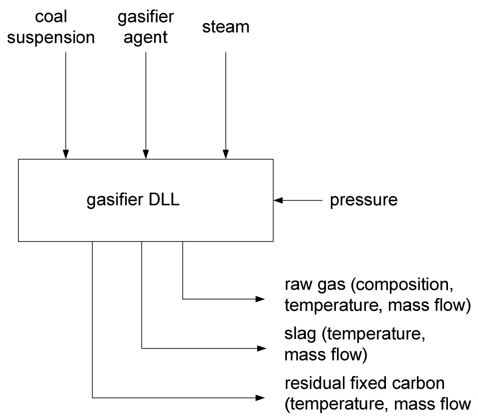

coal suspension
gasifier agent
steam
gasifier DLL
pressure
raw gas (composition, temperature, mass flow)
slag (temperature, mass flow)
residual fixed carbon (temperature, mass flow

Figure 9: calculation of the gasifier DLL

The gasifier is implemented in a Dynamic Link Library (DLL), which was developed by the Siemens Fuel Gasification Technology GmbH Freiberg. In the DLL the thermodynamic equilibrium is calculated. Therefore the equilibriums for the reversible reactions mentioned in chapter 1.1 are calculated.

Figure 9 shows the in- and outputs of the gasifier DLL.

3.2.2 Quench

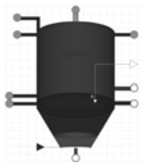


Figure 10: screen shot of the quench icon

In the quench the raw gas from the reactor is cooled down and saturated.

The quench consists of two zones the gas space and the sump. For each zone own mass and energy balances are considered. However convective heat transfer between the raw gas in the gas space and the water in the sump is assumed. In the gas space the raw gas from the reactor is saturated and therefore cooled down. This is done by the injection of fresh water at the top of the quench (Figure 11).

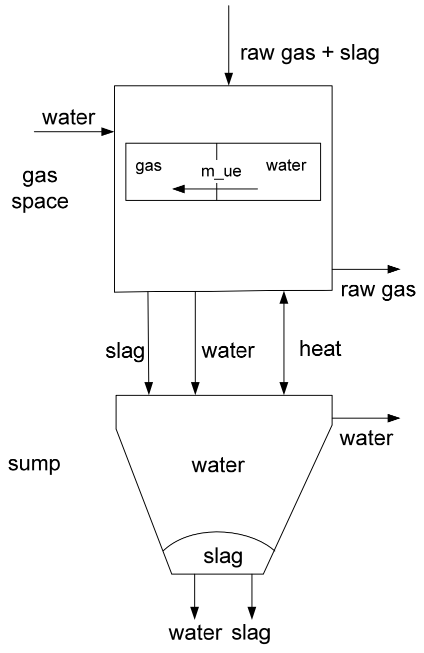


Figure 11: mass flows at the quench

In the gas space one energy balance for the media water, slag and gas is regarded.

For the calculation of the saturated steam fraction the following equation is used:

$$\varphi_{Steam} = \frac{p_{S,S}(T_G)\cdot M_s}{p_G \cdot M_G}$$

φ_{Steam} is the mass fraction of steam, $p_{S,S}(T_G)$ is the saturation vapor pressure at the temperature T_G, p_G is the gas pressure and $M_{S,}\, M_G$ are the molar masses of steam and the gas.

The vaporization flow has to be regarded in the mass balances of the water and the gas in the gas space. Furthermore the heat of vaporization has to be taken into account in the energy balance of the gas space.

```
model quench
  …
  //mass balances gas space
  der(m_gas) = in_g.m_flow + out_g.m_flow + m_ue;
  der(mXi[s]) = in_g.mXi_flow[s] + out_g.mXi_flow[s] + m_ue;
   0 = in_w.m_flow + out_w.m_flow – m_ue;
  …
  //energy balance gas space
  der(U) = … - delta_hv*m_ue;
  …
  //calculation of the saturated steam fraction
  gas.Xi[s] = p_steam*M_s/gas.p*gas.MM;
  p_steam = saturationPressure(gas.T);
  …
end quench;
```

3.3 Gas Treatment

3.3.1 Venturi Scrubber System

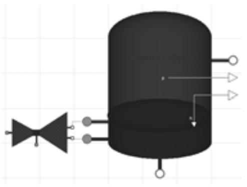


Figure 12: screen shot of the venture scrubber icon

The venturi scrubber system is located between the quench and the partial condenser. It consists of a venturi jet and a drum. The venturi jet is a pressure drop component. There are two different types of venturi’s: controlled and uncontrolled.

The raw gas, which leaves the venturi scrubber system, is saturated. For the calculation of the saturated gas properties the same equations as in the quench are used.

3.3.2 Partial Condenser

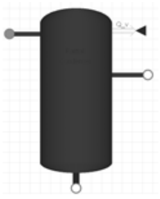


Figure 13: screen shot of the partial condenser icon

The partial condenser is located between the venturi scrubber system and the synthesis gas system. There the raw gas is cooled down and the condensate is separated. The raw gas leaving the partial condenser is saturated.

For the calculation of the gas properties the same equation as in the quench were used. Furthermore the energy balance equation is enlarged by the heat loss flux $\dot{Q}$. This heat flux is a real input value. It should be so adjusted that the temperature difference between the inlet and the outlet of the partial condenser reaches a given value.

3.4 Slag Discharge System

3.4.1 Slag Hopper

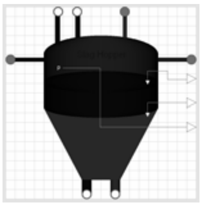


Figure 14: screen shot of the slag hopper

The function of the slag hopper is to discharge the slag from the pressurized system of the quench into the atmospheric pressure environment. Therefore 5 steps appear:

- filling of the slag hopper with water
- pressurizing until the pressure of the quench is reached
- filling of the slag hopper with slag (thus lead to a displacement of water from the slag hopper into the quench)
- depressurizing of the slag hopper
- drawdown of the slag hopper

As in the below explained components only one energy balance is considered.

3.5 Initialization

For every component the temperature and the filling levels can be defined. For the components which cover to the saturation of gas the dry gas composition and for all other components the gas composition have to be deposited.

Furthermore the user can decide for each component if the pressure should be initialized or not.

4 Simulation Results

As mentioned in the introduction the developed models shall be used to enhance process control. Therefore existing control methods can be verified and in addition tests of advanced process control conceptions like kinds of MPC (**M**odel **P**redictive **C**ontrol) and virtual sensors are allowed.

The Gasification Island contains a multitude of control systems:

- level control systems
- temperature control systems
- pressure control systems
- mass flow rate control systems

In cooperation with the Siemens Fuel Gasification Technology GmbH Freiberg, analyses were carried out on how far advantages appear by applying the advanced process control strategies in comparison to the accepted PID controllers. As an example of use the coal mass flow control system from the feeding vessel to the reactor was chosen due to the occurred dead times. Furthermore this mass flow control system is of great importance to the gasification process, because of its impact to the quality (temperature, composition) of the formed raw gas.

The following two figures show the results of some simulations. Figure 15 shows the step response of the controlled coal mass flow for an accepted PI controller and a model based controller. To avoid overshooting both, controllers were designed for aperiodic transient behavior (this is an arbitrary chosen design case and doesn't reflect the behavior of the implemented control system in the gasification plant). The response of the system for a ramp like change of the coal mass set point is given in Figure 16.

Both figures show that the coal mass flow can be significantly enhanced for this control system design case by using advanced process control concepts.

The control tests were done in Matlab. Therefore, the developed Modelica/Dymola models were converted into a Simulink model.

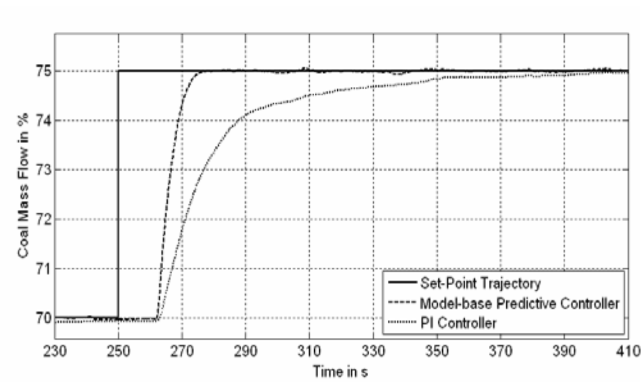


Figure 15: step response of the coal mass flow at sudden change of the coal mass flow set point value (arbitrary chosen design case: aperiodic transient behavior of the controllers)

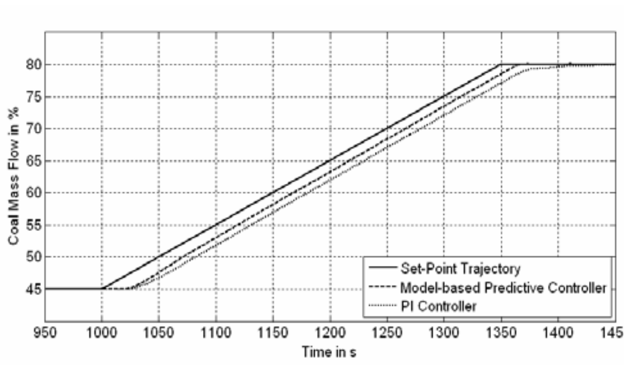


Figure 16: ramp like change of the coal mass flow set point (arbitrary chosen design case: aperiodic transient behavior of the controllers)

5 Conclusion and future work

The Gasification Island was developed in the Modelica language. Therefore new components were designed which are based on the Modelica_Fluid and Modelica.Media libraries. First analyses were done to enhance the process control of the coal mass flow from the feeding vessel to the reactor.

The further step is to enhance the gasifier model. Therefore the reaction kinetics of the reactions listed in 1.1 and the complex heat balance (heat radiation, convective heat transfer …) will be implemented.

Abbreviations

SFG	Siemens Fuel Gasifier

References

[1] http://www.gasification.org

[2] Klose, E.; Toufar, W.: Grundlagen der Vergasung, 1. Lehrbrief. Lehrbriefe für das Hochschulfernstudium, 1985

[3] Higman, C.; van der Burgt, M.: Gasification. Gulf Professional Publishing, Amsterdam, 2002

[4] Hannemann, F.: Siemens Fuel Gasification Technology at a Glance. Virtuhcon Workshop 2007, Freiberg, Germany

[5] Heyde, M.: Fluidisieren von Schüttungen.

[6] http://www.fossil.energy.gov

[7] Casella, F.; Otter, M.; Proells, K.; Richter, C.; Tummescheit, H.: The Modelica Fluid and Media Library for Modelling of Incompressible and Compressible Thermo-Fluid Pipe Networks. 5th International Modelica Conference Proceedings, 2006

Transient Modelling of a Controllable Low Pressure Accumulator in CO_2 Refrigeration Cycles

Marcos Bockholt[1] Wilhelm Tegethoff[1] Nicholas Lemke[2]
Nils-Christian Strupp[1] Christoph Richter[1]

[1]Technical University Braunschweig, Institute of Thermodynamics
Hans-Sommer-Str. 5, 38106 Braunschweig
Email: m.bockholt@tu-bs.de

[2]TLK-Thermo GmbH
Hans-Sommer-Str. 5, 38106 Braunschweig
Email: n.lemke@tlk-thermo.de

Abstract

Low pressure accumulators are usually employed in mobile R744 HVAC units to assure reliable operating conditions and consequently to extend equipment life. Furthermore, the design parameters of accumulator, e.g. the oil bleed hole, influence the coefficient of performance (COP) of the refrigeration cycle. A poorly designed accumulator may lead to inefficient refrigeration cycles. Thus, accumulators with a variable oil bleed hole, also called controllable accumulators, may be employed to bring the system to optimal operating condition assuring good performance. The aim of this work is to implement a semi-empirical physically based transient Modelica controllable accumulator model, which is part of TIL (the TLK-IfT-Library). Transient simulations are carried out to evaluate the impact of a controllable accumulator in an automotive refrigeration system.

Keywords: controllable accumulator; refrigeration cycle control; COP optimisation; fluid systems

1 Introduction

On January 2006 the EU agreed to vanish HFC-134a from air conditioning systems of new vehicle models from 1 January 2011. The natural refrigerant R744 is one of the promising candidates to replace R134a. Therefore, the actual vehicle refrigeration technology has to be optimized to reach the efficiencies using R134a. In fixed orifice tube R744 air-conditioning systems a low pressure accumulator is usually placed at the compressor inlet in order to store excess refrigerant, allowing an optimum system performance under various ambient conditions and compensating refrigerant loss through leakage along the life cycle. The refrigerant quality at the accumulator inlet is also influenced by the oil bleed hole located in the “J” tube of the accumulator, see e.g. Fig. 3. The size of the oil bleed whole is an optimization parameter in accumulator design and should be variable to attend optimum performance for different operating conditions and avoid high compressor outlet temperatures. This variability of the oil bleed hole can be put into practice by building a controllable accumulator.

2 TIL

TIL is a new component model library for thermodynamic systems that was developed by the Institute for Thermodynamics (IfT) and the TLK-Thermo-GmbH and that allows for the steady-state and transient simulation of thermodynamic systems. The underlying design principles as well as a detailed description of selected component models is given by [4].

TIL provides component models for the simulation of refrigeration, air-conditioning, and heat-pump systems. Many component models use a formulation of the balance equations that is similar to the balance equations for the accumulator as presented in the following section. TIL uses the object-based fluid property library TILFluids for the computation of fluid properties. This fluid property library uses a generalized approach to include external fluid property computation codes (e.g., REFPROP) in Modelica and a number of software tools.

3 Mathematical Modelling

Fig. 1 shows the control volume of the semi-empirical accumulator model V_{kv}. The following assumptions are made:

- The accumulator has adiabatic walls.
- The control volumes V_{kv}^1 are constant in time.
- Changes of kinetic and potential energy are not taken into account.
- The accumulator characteristics regarding the accumulated mass is modeled according to steady-state characteristic curves. The characteristic diagram determines the outlet enthalpy depending on the filling level.
- The accumulator outlet enthalpy may be changed by opening the oil bleed hole.
- Oil fraction in the liquid phase is ignored.

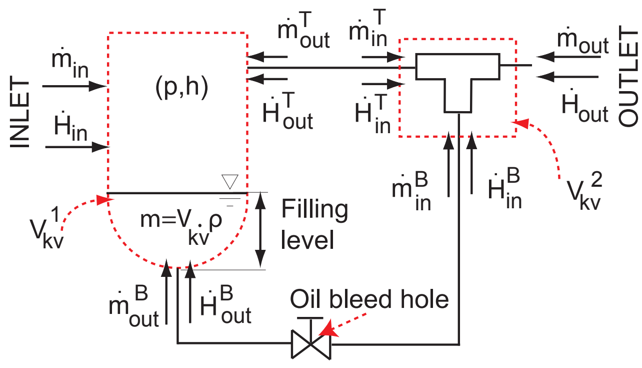


Figure 1: Controlled accumulator model.

3.1 Conservative Laws

3.1.1 Mass Balance

The transient mass balance equation for the control volume V_{kv}^1 is stated as follows:

$$\frac{dm}{dt} = \dot{m}_{in} + \dot{m}^T_{out} + \dot{m}^B_{out} \qquad (1)$$

Eq. 1 can be stated as:

$$\begin{aligned} \frac{d}{dt}(\rho\cdot V_{kv}) &= \dot{m}_{in} + \dot{m}^T_{out} + \dot{m}^B_{out} \\ V_{kv}\cdot\frac{d\rho}{dt} &= \dot{m}_{in} + \dot{m}^T_{out} + \dot{m}^B_{out} \end{aligned}$$

Using the Bridgmann's table the derivative $\frac{d\rho}{dt}$ above can be split into:

$$\frac{d\rho}{dt} = \left|\frac{d\rho}{dh}\right|_p \cdot \frac{dh}{dt} + \left|\frac{d\rho}{dp}\right|_h \cdot \frac{dp}{dt}$$

The partial derivatives $\left|\frac{d\rho}{dh}\right|_p$ and $\left|\frac{d\rho}{dp}\right|_h$ are modeled in TILFluids for the one phase and two phase regions.

3.1.2 Energy Balance

The 1st. Law of Thermodynamics for an opened system in its transient form is applied to the control volume V_{kv}^1 resulting in the following differential equation:

$$\begin{aligned} \frac{dU}{dt} &= \dot{m}_{in}\cdot h_{in} + \dot{m}^T_{out}\cdot h^T_{out} + \dot{m}^B_{out}\cdot h^B_{out} \\ \frac{d}{dt}\left(H - p\cdot V^1_{kv}\right) &= \dot{H}_{in} + \dot{H}^T_{out} + \dot{H}^B_{out} \\ \frac{d}{dt}(m\cdot h) &= \dot{H}_{in} + \dot{H}^T_{out} + \dot{H}^B_{out} + V^1_{kv}\cdot\frac{dp}{dt} \\ m\frac{dh}{dt} + h\frac{dm}{dt} &= \dot{H}_{in} + \dot{H}^T_{out} + \dot{H}^B_{out} + V^1_{kv}\cdot\frac{dp}{dt} \qquad (2) \end{aligned}$$

Using the mass balance Eq. 1, the Eq. 2 is rewritten into:

$$\begin{aligned} \frac{dh}{dt} = \frac{1}{m}\cdot[&\dot{m}_{in}\cdot(h_{in}-h) + \dot{m}^T_{out}\cdot(h^{out}_T - h) + \\ &+\dot{m}^B_{out}\cdot(h^B_{out}-h) + V^1_{kv}\cdot\frac{dp}{dt}] \qquad (3) \end{aligned}$$

where h is the enthalpy of the in the accumulator accumulated refrigerant. The system of differential equations (see Eqs. 1 and 2) is reduced from $\left(\frac{dh}{dt},\frac{dp}{dt},\frac{dm}{dt}\right)$ to $\left(\frac{dh}{dt},\frac{dp}{dt}\right)$ using the Bridgmann's table. This formulation has been shown to be very efficient for transient simulations (see [2, 7] for further details).

3.2 Accumulator

The semi-empirical accumulator model V_{kv}^1 in Fig. 1 is treated here in detail. Different from the existing TIL accumulator model, the model extended here is able to influence the outlet enthalpy and hereby to increase or decrease the level of accumulated liquid refrigerant by opening the oil bleed hole. Basically, the accumulator's physical behavior is characterized by its filling level, a phase separation efficiency and an empirical characteristic diagram.

3.2.1 Filling level

The filling level is defined as the liquid fraction of the accumulated refrigerant:

$$\delta = 1 - \left(\frac{h - h_{liq}}{h_{vap} - h_{liq}} \right). \tag{4}$$

where h is the enthalpy of the accumulated refrigerant. The outlet enthalpy at the accumulator top h_{out} depends on this variable as shown in Sec. 3.2.3.

$$\delta = \begin{cases} \delta_{min} = 0 & \text{if the accumulator is empty.} \\ \delta_{drop} & \text{if fluid droplets occur} \\ & \text{at the accumulator outlet.} \\ \delta_{max} = 1 & \text{if the accumulator is flooded} \\ & \text{with saturated liquid.} \end{cases}$$

A detailed investigation is described in [5] for different accumulator geometries and operation conditions using a transparent accumulator.

3.2.2 Separation efficiency η_S

The accumulator separation efficiency η_S describes the ability of an accumulator in separating the refrigerant phases. It is defined as as follows:

$$\eta_S = x_{out} \tag{5}$$

where x_{out} is the accumulator outlet quality when the accumulator filling level is between the droplet filling level and the minimum filling level, $\delta_{min} \leq \delta \leq \delta_{drop}$. An ideal accumulator without an oil bleed hole would have a separation efficiency of 100%, i.e. only refrigerant vapor h_{vap} leaves the accumulator. The separation efficiency is strongly dependent on the oil bleed hole diameter and is estimated from steady-state measurement data, see [3, 5]. These data show that the separation efficiency may range from 75% up to 98%.

3.2.3 Characteristic diagram

The accumulator characteristic diagram is divided in four different operating conditions according to its filling level. These operating conditions are drawn as follows:

I) **Accumulator is nearly empty** ($\delta \leq \delta_{min}$): in this operation point no liquid droplet occurs from eventually accumulated refrigerant in the receiver and the refrigerant phases are separated. This means that if the refrigerant enters the accumulator with quality $x_{in} = 0.7$ it will leave it with quality $x_{out} = \eta_S = 1$. The liquid part begins to accumulate.

II) **Accumulator has a filling level with few liquid droplets at the outlet** ($\delta_{min} < \delta \leq \delta_{drop}$): if the droplet filling level δ_{drop} is not reached, a very small amount of liquid droplets from the accumulated refrigerant occurs at the outlet of the accumulator. The refrigerant phases are still separated and the accumulator outlet quality is the separation efficiency $x_{out} = \eta_S$. This is the most common operating condition and will be treated in the steady state simulation presented in a further section.

III) **Accumulator has a filling level with excess of liquid droplets at the outlet** ($\delta_{drop} < \delta \leq \delta_{max}$): for this operation point, the accumulated refrigerant in the accumulator has reached a level in that large amount of accumulated liquid starts to leave the accumulator. The refrigerant phases cannot be clearly separated. The outlet enthalpy starts to decrease and enters the two phase area ($x_{liq} < x_{out} < \eta_s < x_{vap}$).

IV) **Accumulator is full** ($\delta > \delta_{max}$): if the receiver reached this filling level, it is then flooded and there is no separation of the refrigerant phases. The liquid phase dominates in the receiver and the outlet enthalpy equals or is small than the saturated liquid enthalpy, i.e. $\eta_S = 0$. This is a vary rare operation condition and is out of the scope of this work.

3.3 Controllable accumulator

The controllable accumulator, is shown as an extension of the ideal accumulator of TIL. A prototype of a controllable accumulator is presented in Fig. 3. The "J"-tube with the oil bleed hole may be modeled by correlating the oil bleed hole in the "J"-tube with the separation efficiency stated in Eq. 5.

3.3.1 Oil bleed hole ø

To verify the effect of changing the oil bleed hole diameter in a standard accumulator a measurement

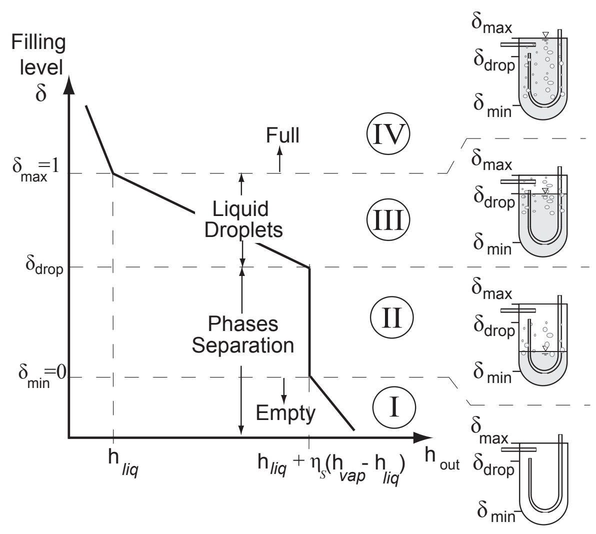


Figure 2: Accumulator characteristic diagram.

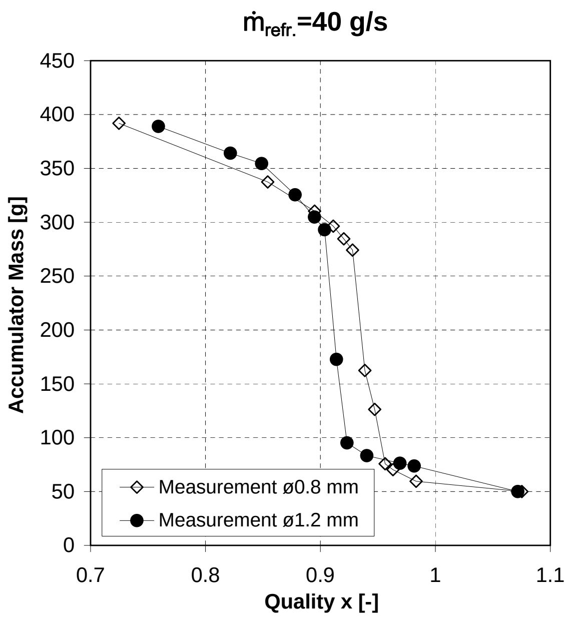


Figure 4: Accumulated mass dependence on the oil bleed hole, from [5].

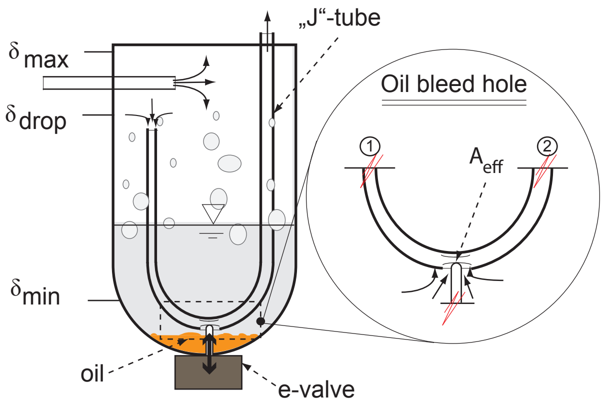


Figure 3: Accumulator prototype, from [1].

4 Steady-state simulation results

To investigate the impact of changing the accumulator efficiency in cycle behavior, a high ambient temperature and idle compressor speed condition for an automotive air-conditioning is applied. The refrigeration cycle characteristics and boundary conditions are summarized in the Tabs. 1 and 2. As a first approximation, the compressor volumetric and isentropic efficiencies as well as the heat transfer coefficients in the heat exchangers are kept constant for the cycle. The first step in this analysis is to find out the optimum operation pressure for the chosen boundary conditions and different accumulator efficiencies. Fig. 5 shows how the optimal high pressure varies with the accumu-

Total CO_2 mass $[kg]$	Cycle internal volume $[l]$	Compressor displacement $[cm^3]$
0.5	1.5	28

Table 1: Cycle characteristics

Compressor speed $[rpm]$	$\dot{m}_{air}$ evaporator $[g/s]$	$\dot{m}_{air}$ gas cooler $[g/s]$	Ambient Temperature $[^\circ C]$
780	140	600	40

Table 2: Boundary conditions for an automotive application

configuration developed by the Institut für Thermodynamik in Braunschweig, for the purpose of determining the liquid level in low-pressure accumulators with carbon dioxide as refrigerant is used. The measurements are performed varying the gas cooler outlet temperature yielding a variation of the accumulator outlet quality, see [5] for further details. In Fig. 4 it is observed that the accumulator outlet quality decreases by increasing the oil bleed hole diameter, i.e. reducing the separation efficiency. Thus, the separation efficiency will be used as a variation parameter to control the accumulator. A physical correlation between the efficiency and oil bleed hole should be further investigated in future works.

lator separation efficiency. The optimal high pressure is reached by setting the valve flow area 0.35 mm^2. The result of the COP-optimized cycles for three different accumulator separation efficiency are summarized in Tab. 3. In Fig. 6 the COP-optimized cycle is shown in the pressure-enthalpy diagram for the accumulators with small, medium and large-sized oil bleed hole. The increase in the compressor suction density is observed in Fig. 6 at point 1, which is shifted to the two phase region when decreasing the accumulator separation efficiency. In order to keep the same suction density at the compressor inlet, an enhanced internal heat exchanger (IHX) with maximal thermodynamic efficiency is used. The result is shown in the pressure-enthalpy diagram in Fig. 7. Now, a change on the accumulator efficiency has neither effect on the system Coefficient of Performance (COP) nor changes its cooling capacity. The points 4 and 5 are shifted to the left at the same amount as the point 6. This fact evidences a dependence between the accumulator separation efficiency and the IHX heat transfer two-phase heat transfer effects.

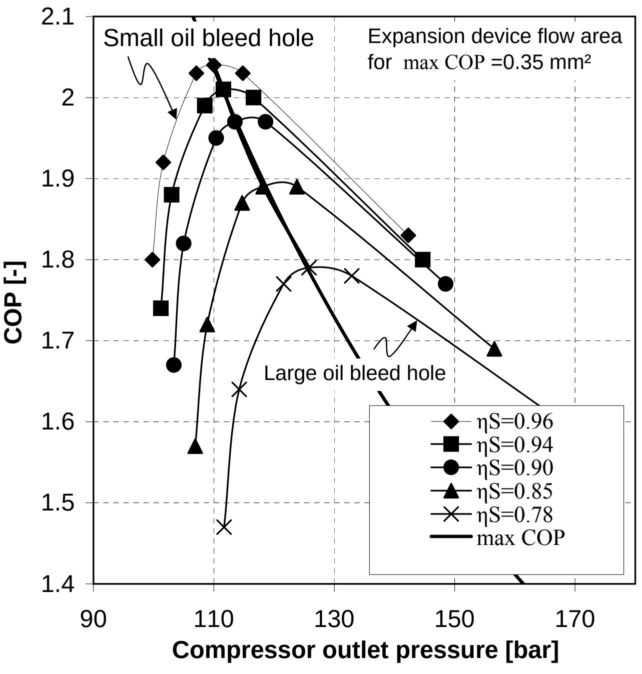


Figure 5: Coefficient of Performance (COP) sensitivity analysis for different accumulator separation efficiencies.

		Separation efficiency η_S		
State variable	Unit	0.78	0.85	0.96
$m_{CO_2\ cycle}$	[kg]	0.5	0.5	0.5
$m_{CO_2\ accu}$	[kg]	0.163	0.185	0.214
$x_{accu\ out}$	[-]	0.78	0.85	0.96
$T_{comp\ out}$	[°C]	91.4	94.5	100.9
$\dot{m}_{CO_2\ cycle}$	[g/s]	37.7	35.1	31.6
$p_{comp\ out}$	[bar]	125.8	118.2	110.0
$\Delta\dot{H}_{air}$	[kW]	3.38	3.34	3.26
IHX $\Delta\dot{H}_{ref}$	[kW]	1.4	1.3	1.1
COP	[-]	1.79	1.89	2.04

Table 3: Impact of the separation efficiency in steady-state cycle simulation.

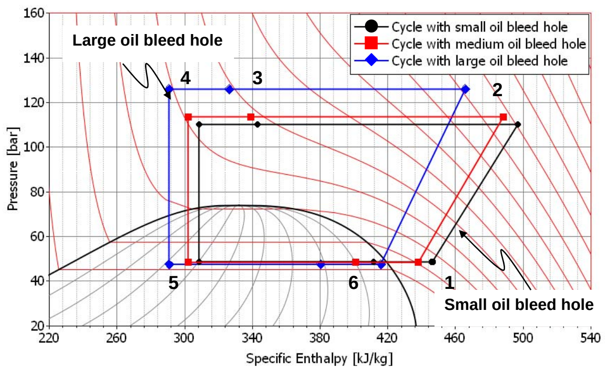


Figure 6: Pressure-enthalpy (p-h) diagram for COP-optimized cycles with large, medium and small oil bleed hole.

5 Transient simulation of a CO_2 refrigeration cycle with a controllable accumulator

In this application, a controllable accumulator is used to avoid that the temperature at the compressor outlet $T_{comp\ out}$ exceeds the oil decomposition temperature, e.g 160 °C. The cycle used previously for the steady state simulation, see Fig. 8, is now used in a transient simulation, where the compressor speed is set to $n = 2100\ rpm$ and the gas cooler and evaporator air inlet temperature $T_{evap\ in}$ are assumed to be 40°C. Fig. 9 shows the results of the transient simulation for some of the state variables. At time $t = 50\ s$ the

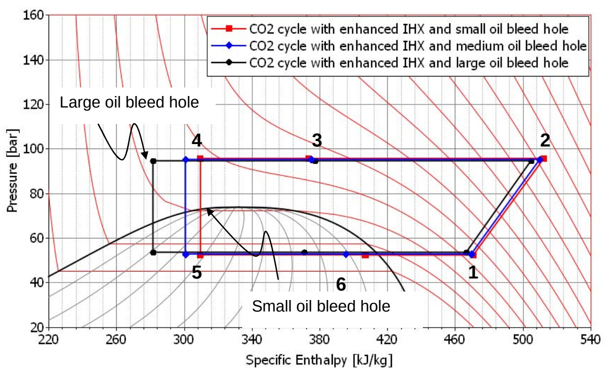


Figure 7: Pressure-enthalpy (p-h) diagram for COP-optimized cycles with large, medium and small oil bleed hole and enhanced IHX, with heat exchange area ($A_{IHX} \approx \infty$).

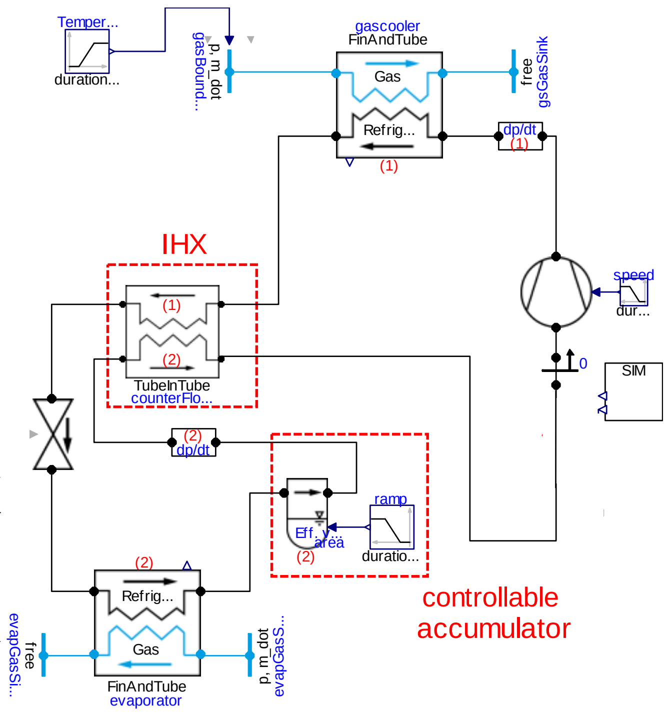


Figure 8: CO_2 refrigeration cycle with controllable accumulator using component models from TIL.

separation efficiency of the accumulator is decreased from η_S=96% to η_S=78%, compare $\eta_S = x_{out}$ in Fig. 9. Some refrigerant mass in the accumulator m_{accu} is moved to the cycle high-side pressure. The suction density at the compressor inlet increases yielding a higher compressor shaft power P_{comp}. An increase in P_{comp} means a decrease in the system coefficient of performance COP=$\Delta\dot{H}_{air}/P_{comp}$ as observed in Fig. 9. The compressor outlet temperature is decreased to a value smaller than the maximum oil working temperature. The increase in the cycle refrigerant mass flow rate due to higher compressor suction densities causes an insignificant increase in the cooling capacity for this modeling assumptions. The evaporator air outlet temperature $T_{air\ evap\ out}$ increases slightly.

6 Conclusion

The transient model of a controllable accumulator is presented to investigate the effects of varying the separation efficiency in an automotive CO_2 refrigeration. The model consists of simple models from the new component model library for thermodynamic systems that was developed by the Institute for Thermodynamics (IfT) and the TLK-Thermo-GmbH. The mathematical formulation used in the modeling allows an accelerated analysis of the parametric variation.

The results from the steady state simulation show a strong dependency between the accumulator separation efficiency and the internal heat exchanger (IHX) efficiency if the system Coefficient of Performance (COP) is considered. In a first simulation run with constant heat transfer coefficients in the heat exchangers it was observed an increase in the system COP when closing the oil bleed hole. Otherwise, if the oil bleed hole is opened the compressor outlet temperature decreases avoiding the oil temperature to reach critical limits. A second simulation run showed that using a nearly optimal IHX the oil bleed hole variation has no effect in the cycle COP and cooling capacity.

A transient simulation is carried out for a an automotive air-conditioning boundary condition. As a first application, it is shown that the compressor outlet temperature may be kept under the oil critical limit without loss of cooling capacity.

Future work will concentrate on finding an optimal relationship between IHX efficiency and accumulator separation efficiency as well an optimal control strategy for a CO_2 refrigeration cycle using this innovative component. Two-phase heat transfer effects in the IHX and in the other cycle heat exchangers should be taken into account in order to predict the cycle behavior more accurately when varying the accumulator oil bleed hole. The isentropic and volumetric compressor efficiencies should also be mapped more accurately so that the cycle mass flow rate and compressor outlet temperature can be precisely estimated.

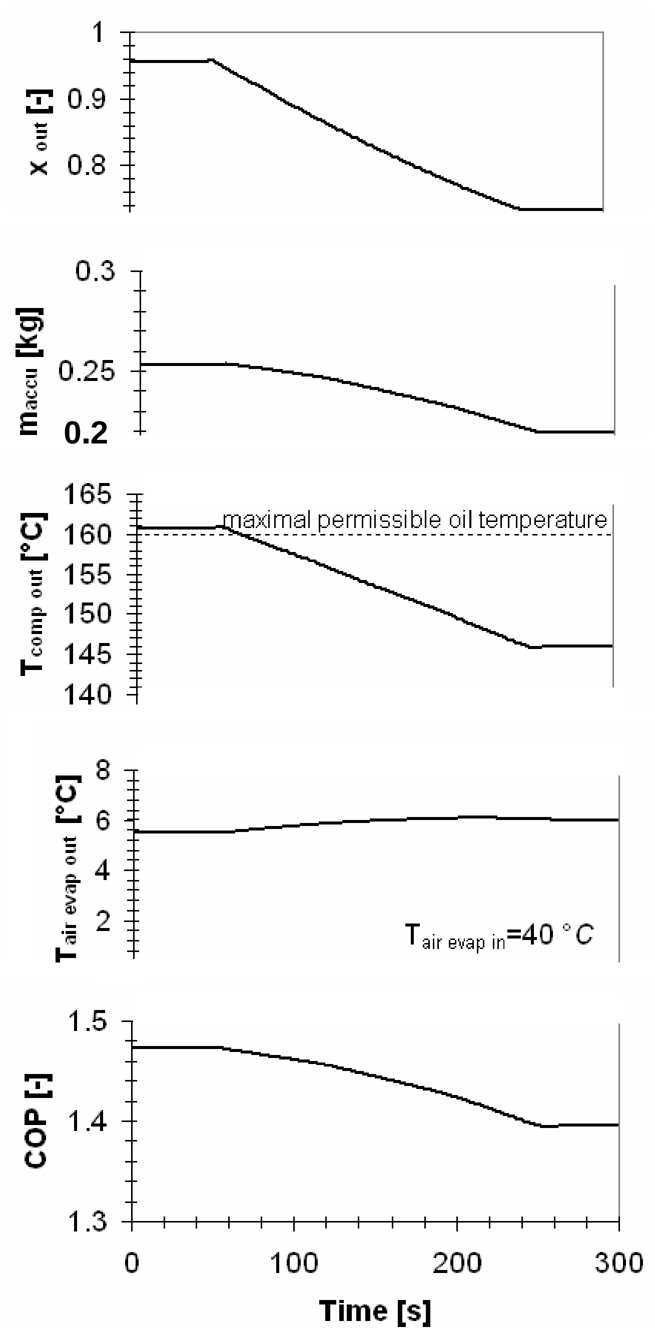


Figure 9: Simulation results of the transient controllable accumulator model in a CO_2 refrigeration cycle

References

[1] Hirota, H.: *Refrigertion cycle*, European Patent Application EP 1607698 A2, TGK Company Ltd., Tokyo, 2005

[2] Lemke N.: *Untersuchung zweistufiger Flüssigkeitskühler mit dem Kältemittel CO_2*, PhD thesis, TU-Braunschweig, Intitute of Thermodynamics, 2005.

[3] Raiser H., Heckenberger T., Tegethoff T., Försterling S.: *Transient Behavior of R744 Vehicle Refrigeration Cycles and the Influence of the Suction Side Accumulator Design*, SAE International, Nr. 2006-01-0162, 2006.

[4] Richter C.: *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems*, TU Braunschweig, PhD-thesis, 2008

[5] Strupp C., Lemke N., Tegethoff T., Köhler J.: *Investigation of Low Pressure Accumulators in CO_2 Refrigeration Cycles*, International Congress of Refrigeration, Beijing, China, ICR07-E1-1480, 2007.

[6] Tegethoff W.: *Eine objektorientierte Simulationsplattform für Kälte-, Klima- und Wärmepumpensysteme*, PhD thesis, TU-Braunschweig, Intitute of Thermodynamics, 1999.

[7] Tegethoff W., Lemke N., Correia C., Cavalcante P., Köhler J.: *Component modelling and specification using a new approach for transient simulation*, VDA Alternate Refrigerant Refrigerant Winter Meeting- Automotive Air-Conditioning and Heat Pump Systems, 2004

Modeling and Simulation of a Thermoelectric Heat Exchanger using the Object-Oriented Library TIL

Christine Junior, Christoph Richter, Wilhelm Tegethoff, Nicholas Lemke, Jürgen Köhler
Institut für Thermodynamik, TU Braunschweig, Germany
c.junior@tu-bs.de ch.richter@tu-bs.de

Abstract

Thermoelectric technology allows for the direct conversion of a temperature difference into an electric potential and vice versa. Thermoelectric devices can act as coolers, heaters, or power generators and applications of small capacity thermoelectric modules are widespread. Applications of large capacity thermoelectric devices have been limited for decades by their low efficiency. New environmental regulations regarding the manufacture and release of CFCs have revived the interest in this area. Recent investigations on thermoelectric materials promise that their thermoelectric efficiency can be improved dramatically. This would mean a breakthrough for new fields of applications for thermoelectric modules. A new Modelica model of a Peltier water-water heat exchanger was developed for transient simulations. The new model uses component models from the object-oriented Modelica library TIL. The new model was used to simulate the transient behavior of a Peltier heat exchanger during a sudden reversion of the applied voltage. The numerical results were compared to measurement results from a prototype.

Keywords: heat exchanger; simulation; thermoelectrics; Peltier element

1 Introduction

Thermoelectric technology allows for the direct conversion of a temperature difference into an electric potential and vice versa. The French physicist Jean Peltier discovered in 1834 that an electric current sent through a circuit made of dissimilar conducting materials yields heat absorption at one junction and heat rejection at the other. Standard thermoelectric modules utilize doped bismuth telluride as semiconductor and achieve moderate performance. They can act as coolers, heaters, or power generators and applications of small capacity thermoelectric modules are widespread. However applications of large capacity thermoelectric devices have been limited in the past by the low efficiency of thermoelectric modules. Recent scientific advances regarding new materials and assembly methods for thermoelectric modules as well as the increasing concerns about fuel economy, harmful emissions of particulate matter, and chemical refrigerants revived the interest in thermoelectric technology. The inherent advantages of thermoelectric systems such as the absence of moving parts, quiet operation, and environmental friendliness of the module itself have further increased the interest. Several investigations for applications of large capacity thermoelectric modules in the fields of refrigeration and air-conditioning [1], waste heat recovering [2], or superconduction [3] have been carried out with promising results.

This paper describes the development of a Modelica model that allows the transient simulation of thermoelectric devices to determine their performance potential. The model for the thermoelectric devices was developed as an add-on for the object-oriented Modelica library TIL (TLK-IfT-Library) described in [4] that allows for the simulation of thermodynamic systems such as air-conditioning and heat-pump systems.

2 Thermoelectric Refrigeration

Thermoelectric refrigeration is achieved when a direct current I is passed through one or more pairs of n-type and p-type semiconductors connected with a metal with high conductivity such as copper as sketched in Figure 1.

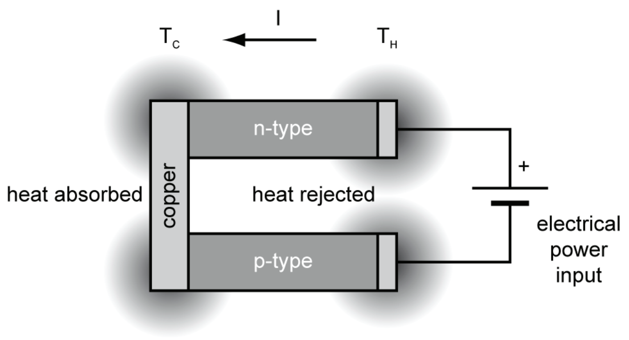


Figure 1: The Peltier effect (thermoelectric cooling) from [5].

If the electric current passes from the n-type to the p-type semiconductor, electrons pass from a low energy level in the p-type material through the interconnecting conductor to a higher energy level in the n-type material. Thus the temperature T_C of the interconnecting conductor decreases and heat is absorbed from the environment. The absorbed heat is transferred by electron transport through the semiconductors to the other end of the function. It is liberated as the electrons return to a lower energy level in the p-type material yielding an increased temperature T_H.

This phenomenon is known as the Peltier effect and is described by the Peltier coefficient π, defined as the product of the Seebeck coefficient α of the semiconductor material and the absolute temperature. The Peltier coefficient relates to a cooling effect as the electric current passes from the n-type to the p-type semiconductor and a heating effect as the polarity of the power supply is changed. Reversing the direction of the electric current also reverses the temperatures of the hot and cold ends.

The amount of heat absorbed at the cold end not only depends on the product of the Peltier coefficient and the electric current flowing through the thermoelectric module but also on two other effects: Due to the temperature difference between the cold and the hot ends of the semiconductors, heat is conducted through the semiconducting material from the hot to the cold end. The amount of conducted heat depends on the thermal conductance κ of the material as well as on the temperature difference. The second effect occurs when the electric current is passing through the semiconductors. The electrical resistance R causes the generation of the so-called Joule heat in equal shares at the cold and the hot side of the thermoelectric device. The Joule heat is dependent on the electrical resistance and proportional to the square of the electric current and therefore becomes eventually the dominant factor.

The heat absorption rate at the cold side of the thermoelectric module can be described taking into account the three different effects mentioned above

$$\dot{Q} = \alpha T_C I - \frac{1}{2} I^2 R - \kappa (T_H - T_C)$$

where α is the differential Seebeck coefficient sometimes referred to as α_{pn}, R the electrical resistance of the thermoelements in series, and κ the thermal conductance of the thermoelements in parallel. The energy efficiency of the thermoelectric device is described by its coefficient of performance (COP) defined as the net heat absorbed at the cold junction divided by the electric power input

$$COP = \frac{\dot{Q}}{P_{el}} = \frac{\alpha T_C I - \frac{1}{2} I^2 R - \kappa \Delta T}{\alpha \Delta T I + I^2 R}$$

The refrigeration capability of a semiconductor material depends on a combined effect of the Seebeck coefficient α, the electrical resistivity ρ, and the thermal conductivity κ of the material over the operational temperature range between the cold and the hot junctions. The electrical resistivity is defined as

$$\rho = R \frac{A}{l}$$

where A is the cross-sectional area of the resistive material and l its length. The three material properties are combined in the thermoelectric figure of merit Z defined as

$$Z = \frac{\alpha^2}{\kappa \rho}$$

The figure of merit is used by material scientists to describe the efficiency of semiconductor materials for thermoelectric applications.

3 Prototype Peltier Heat Exchanger

The Peltier effect can be used for heating and cooling in practical applications by combining thermoelectric modules with conventional heat exchangers. The fluid flowing through the heat exchanger acts as a heat sink at the hot side of the thermoelectric module and as a heat source at the cold side. Figure 2 shows the assembly of the prototype Peltier heat exchanger used for all measurements.

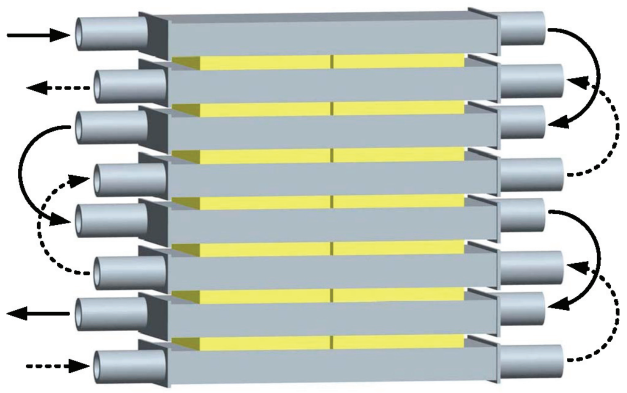


Figure 2: CAD drawing of the prototype Peltier water-water heat exchanger. The Peltier elements are the flat cuboids between two aluminum channels. The orientation of the Peltier elements changes successively between the rows of channels.

Because of the consolidated design and small size of the prototype heat exchanger, water was chosen as coolant at both sides. The heat exchanger consists of rectangular aluminum channels whose endings are covered by plates. Aluminum cores act as connecting tubes. The prototype heat exchanger is assembled so that both sides of the thermoelectric module are in contact with a channel. The arrangement of the thermoelectric modules has to be taken into account for an efficient utilization of the Peltier effect. It is necessary to either heat or cool the channels. A combination of heating and cooling does not yield a reasonable application.

To increase the flow velocity and the heat exchange between the fluid and the wall, three barriers were installed in each channel. A CFD simulation was carried out to determine the flow situation in the channel. The simulations results proved that the fluid meanders through the channel and showed that fluid circulation caused by the barriers leads to a decrease in dead storage capacity and thus to an improvement in the heat exchange between fluid and wall. Figure 3 shows a single channel and the corresponding flow path.

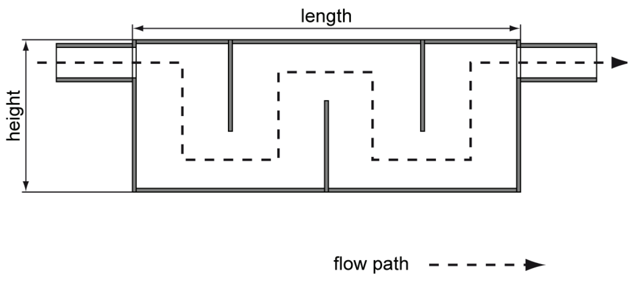


Figure 3: Single channel element of prototype Peltier heat exchanger.

4 Heat Exchanger Model

In order to model the prototype Peltier heat exchanger, a model for a Peltier element had to be developed. The new model was developed based on the component model library TIL (TLK-IfT-Library) that contains models for a steady-state and transient simulation of thermodynamic systems (see [4] for more information).

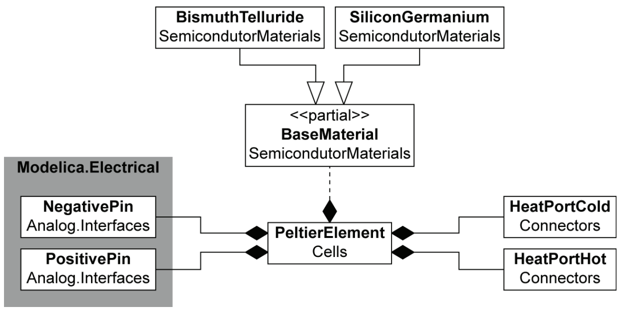


Figure 4: UML class diagram of PeltierElement.

Figure 4 shows a class diagram of the new model PeltierElement. The material properties of the semiconductor material are stored in a record extending from BaseMaterial. Two heat ports derived from the HeatPort connector defined in TIL and two electric pins defined in the Modelica Standard Library are the interface of the PeltierElement. Based on the equations presented in Section 2, the following set of equations is used to describe the Peltier element

$$I_{negative} + I_{positive} = 0$$

$$U_{positive} - U_{negative} = RI_{negative}$$

$$P_{el} + \dot{Q}_C + \dot{Q}_H = 0$$

$$COP = \frac{\alpha T_C I - 1/2\, I^2 R - \kappa(T_H - T_C)}{\alpha \Delta T I + I^2 R}$$

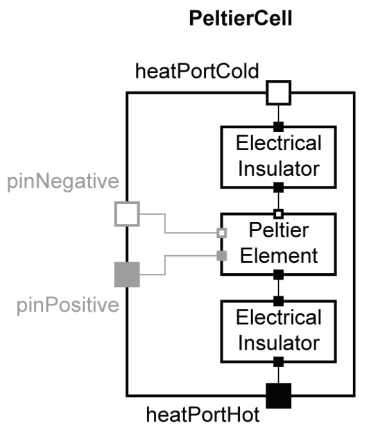


Figure 5: PeltierCell model as defined in TIL_Add-On_ThermoElectrics.

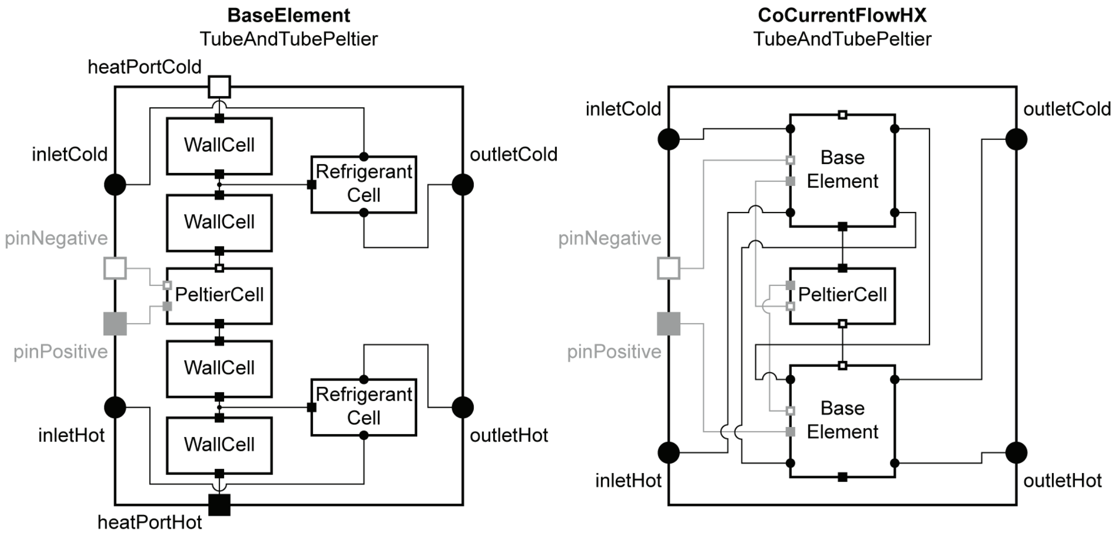


Figure 6: BaseElement and its usage in a Peltier water-water heat exchanger model from TIL_AddOn_ThermoElectrics. The PeltierCell is shown in Figure 5.

$$\dot{Q}_C = -COP \cdot P_{el}$$
$$\dot{Q}_H = (1 + COP) \cdot P_{el}$$

The PeltierElement is instantiated in the PeltierCell model along with two models for electrical insulators as shown in Figure 5. The electrical insulators prevent a short circuit between the Peltier elements and the aluminum channels. Note that the naming of the heat ports in Figure 5 is chosen for the default case that is a positive electric current in the conventional current notation. The hot side eventually becomes the cold side and vice versa if the direction of the current is reversed. The swapping of the corresponding temperatures T_C and T_H is implemented using a smooth transition function with a very short transition period.

In order to model the prototype Peltier heat exchanger shown in Figure 2 in a flexible way, an additional model called BaseElement is introduced that models a single layer of the heat exchanger.

A layer consists of two aluminum channels as sketched in Figure 3 and the Peltier element in between those two channels. The model is illustrated in the left picture in Figure 6. A refrigerant cell and two wall cells from TIL are combined to model a single channel. The reason for using a RefrigerantCell instead of a LiquidCell is that the new heat exchanger model was developed to cover cases of evaporating

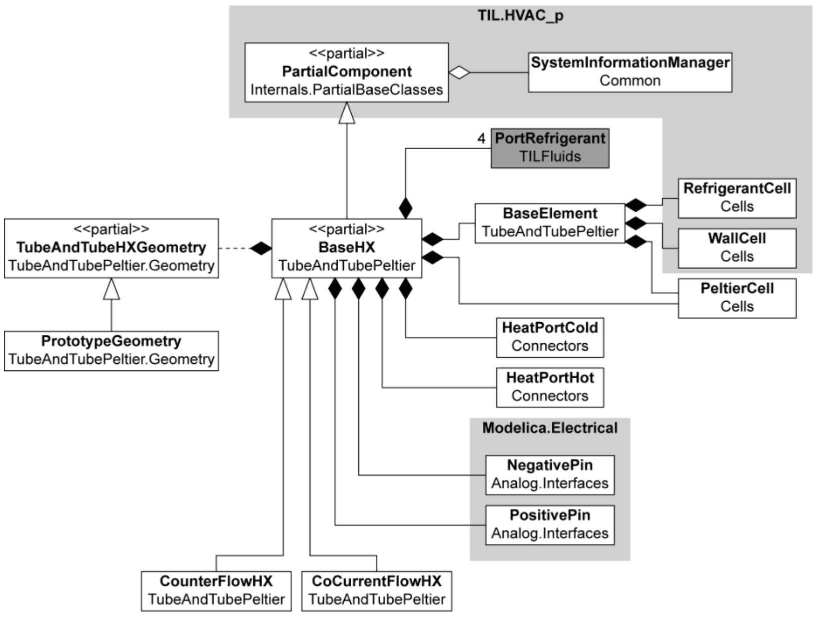


Figure 7: UML class diagram of TubeAndTubePeltier heat exchanger in TIL_AddOn_ThermoElectrics. The wall material model and all heat transfer and pressure drop models are skipped for simplicity.

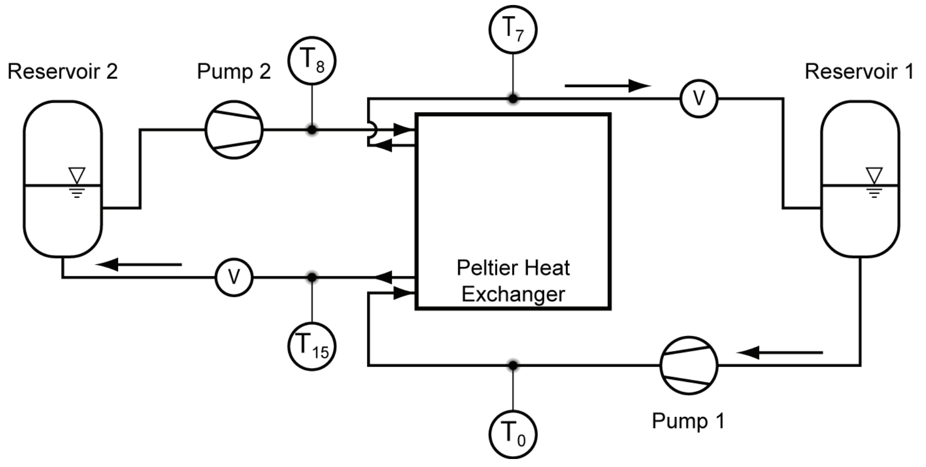


Figure 8: Schematic diagram of Peltier heat exchanger test stand.

and condensing fluids in both fluid paths. The two channels are connected using a PeltierCell as shown in Figure 5. Note that the BaseElement model in Figure 6 can directly be used as a single cell heat exchanger model.

The model for the Peltier heat exchanger assembles instances of BaseElement and PeltierCell as shown in the right picture in Figure 6. The prototype heat exchanger shown in Figure 2 for example is composed of four base elements and three Peltier cells in between. Figure 7 shows the class diagram of the new TubeAndTubePeltier heat exchanger model. Note that the wall material model and all heat transfer and pressure drop models are skipped for simplicity. A more detailed description of the structure of heat exchanger models in TIL is given in [4].

5 Measurements

A series of measurements was carried out with the prototype Peltier water-water heat exchanger presented in Section 3. Figure 8 shows a schematic diagram of the test stand used for all measurements.

To ensure a constant temperature at the water inlet of the prototype, a reservoir was used in both cycles. Water was pumped from the reservoirs into the prototype and flowed back after running through the heat exchanger. The reservoirs were chosen large enough to prevent significant temperature changes during operation. The volume flow rates were regulated with appropriate throttling devices and measured by using conventional water meters.

Besides the volume flow rates characteristic parameters such as the water temperatures at the inlet and outlet of each aluminum tube or the electric current and voltage dropping out over every Peltier element were taken up. The boundary conditions for the measurements were selected in consideration of showing the applicability of the simulation for different premises. Therefore a low, a medium and a high water inlet temperature were chosen and each condition measured by using a low and a high volume flow rate respectively. Each measurement was carried out at a working-voltage of 10 V. A summary of the boundary conditions for all measurements is given in Table 1.

	Water Stream 1		**Water Stream 2**	
#	V_1 [l/min]	T_0 [°C]	V_2 [l/min]	T_8 [°C]
1	2.05	4.00	2.00	4.00
2	0.90	4.00	0.85	4.00
3	2.20	18.00	2.10	18.00
4	0.85	18.00	0.80	18.00
5	2.35	30.00	2.40	30.00
6	1.00	30.00	1.10	30.00

Table 1: Measurements with prototype Peltier water-water heat exchanger.

All measurements were carried out in the same way: After reaching a stationary point for the boundary conditions listed in Table 1, the direction of the electric current was changed from positive to negative in the conventional current notation. The resulting change in temperature was detected until the values became stationary again.

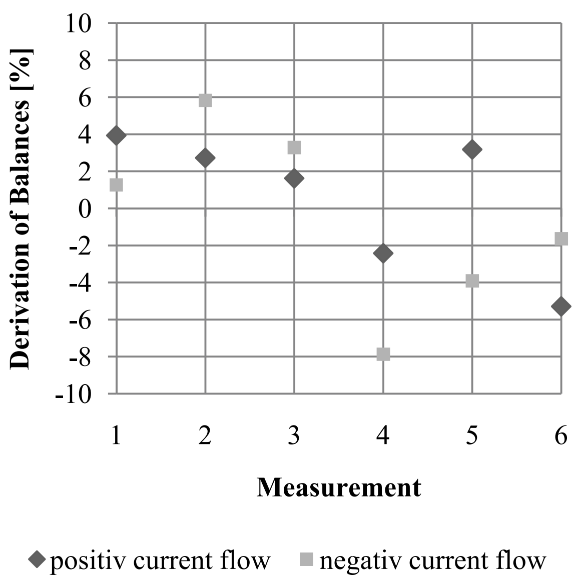


Figure 9: Deviation of electrical and thermal balances for all measurement points.

An evaluation of the quality of the measurements was carried out by comparing the sum of the input power and the gained cooling capacity to the achieved heating capacity according to

$$P_{el} + \dot{Q}_{cooling} = \dot{Q}_{heating}$$

The cooling capacity as well as the heating capacity was calculated from

$$\dot{Q} = \dot{m} c_p \Delta T$$

and the electric power from

$$P_{el} = U \cdot I$$

The deviation within the balance has to be zero for the ideal case. The deviation of the two balances for each measurement is shown in Figure 9. It can be seen that the deviation lies between 1% and 8%, and that the average value lies around 4%. A connection between the direction of the electric current and the resulting deviation can not be identified.

To exclude the existence of a statistical error and to confirm that the deviations of the balances are lying within the measuring accuracy an error analysis was carried out. Therefore, Gauss' error propagation law was used according to

$$\Delta\bar{F} = \sqrt{\left(\frac{\partial F}{\partial x}\Delta\bar{x}\right)^2 + \left(\frac{\partial F}{\partial y}\Delta\bar{y}\right)^2 + \cdots}$$

Measurement 4 from Table 1 was selected for an error analysis exemplarily. A variation of relevant measurands was carried out to find out the impact of these measurands on the total error and to identify possible potentials for further optimization.

Figure 10 shows the impact of the error occurring during the measurement of the temperature difference ΔΔT between the inlet and outlet of the Peltier prototype heat exchanger and during the estimation of the volume flow rate ΔV on the resultant heating or cooling capacity.

Due to the fact that the measuring accuracy of a thermocouple lies at about 0.3 K, the maximum error for the mathematical calculation of the temperature difference can be expected to be 0.6 K when using temperatures measured with two independent thermocouples. This error can be reduced to 0.1 K if the temperature difference is measured using two thermocouples connected in series which was done for all measurements presented in Table 1.

In consideration of the volume flow rate, measurements the deviation of the values estimated with conventional flow meters and the actual values lies between 4% and 9% which results in a maximum deviation of 0.09 l/min. The concluding summation yields - under consideration of these conditions - to

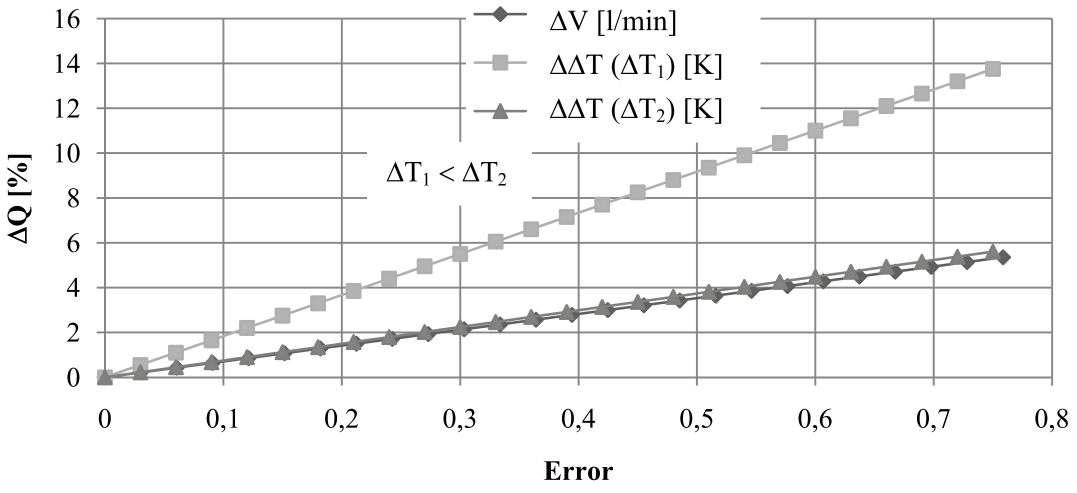


Figure 10: Error for Measurement 4 from Table 1. The corresponding units are given in the key.

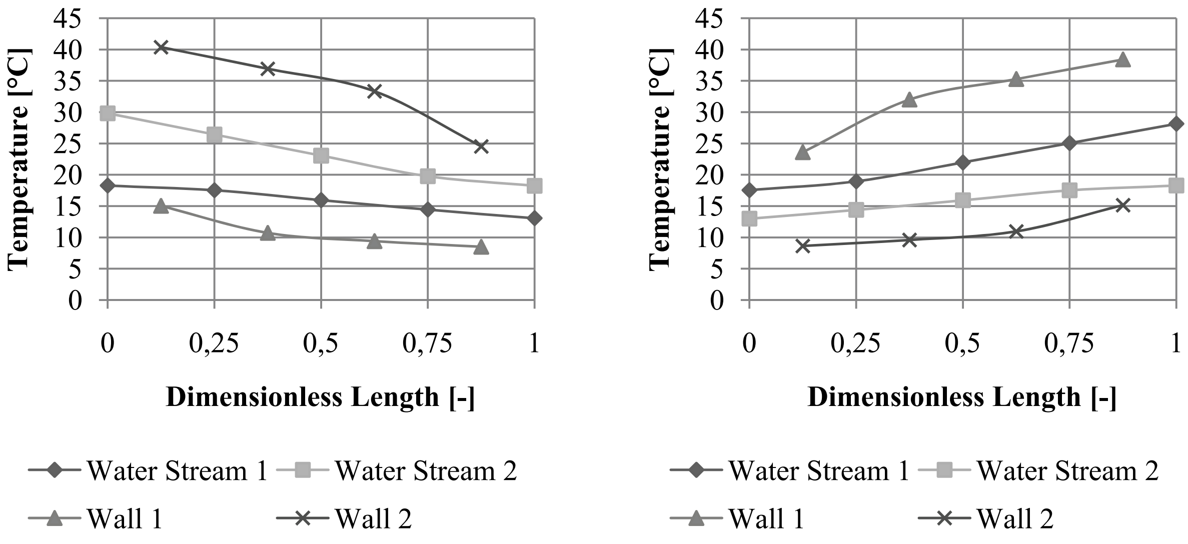


Figure 11: Temperature distribution in prototype Peltier heat exchanger before and after reversion of the applied voltage for Measurement 4 from Table 1.

the result that even the measurements with a deviation of balances of 8% are lying within measuring accuracy.

6 Simulation

Simulations were carried out for all measurements listed in Table 1. Measurement values were used for the electric current, for the two volume flow rates, and for the water temperatures T_0 and T_8 at the two heat exchanger inlets. The Peltier modules used in the prototype Peltier heat exchanger are standard bismuth telluride modules without any further specification from the manufacturer. Constant properties for the Seebeck coefficient α and the thermal conductance κ taken from Rowe [5, Table 9.1] were used in the Peltier element model. The electrical resistance R of the thermoelectric module was not specified by the manufacturer and had to be determined from the measurements. The reversion of the applied voltage was implemented using a smooth transition function with a period of Δt = 1s. This section describes the results obtained for the simulation of Measurement 4 from Table 1. A constant coefficient of heat transfer α = 4,100 W/m^{2}K was used. This coefficient of heat transfer was determined based on a CFD simulation of the flow through a single aluminum channel.

Figure 11 shows the temperature distribution in the prototype Peltier heat exchanger before and after the reversion of the applied voltage. The numbering of the water streams and of the walls refers to the numbering of the two independent water circuits as presented in Figure 8. The water temperatures are shown for the inlet of each channel and for the outlet of the last channel for both water streams. The wall temperatures are averages of the temperatures in the center of both wall cells connected to the same refrigerant cell as shown in Figure 6.

Figure 11 shows that the temperature change in the entrance channel of each water stream is smaller than in all other subsequent channels. This is caused by the fact that the entrance channels are insulated at one side and connected to a Peltier element at the other side whereas all other channels are connected to a Peltier element at both sides. The two diagrams shown in Figure 11 are mirror-symmetrical which demonstrates the reversibility of the process.

Figure 12 shows a comparison of the measured outlet temperature for each water stream with the values obtained from the transient simulation. The top picture shows the change in the electric current I caused by the reversion of the applied voltage.

Figure 12 illustrates that the simulated start and end temperatures differ from the measured temperatures. The simulated system also reacts slower to the sudden reversal of the applied voltage than the real system. Further Measurements are required to improve the model of the Peltier element that is currently based on material constants taken from the literature and the measured electrical resistance as explained in the beginning of this section.

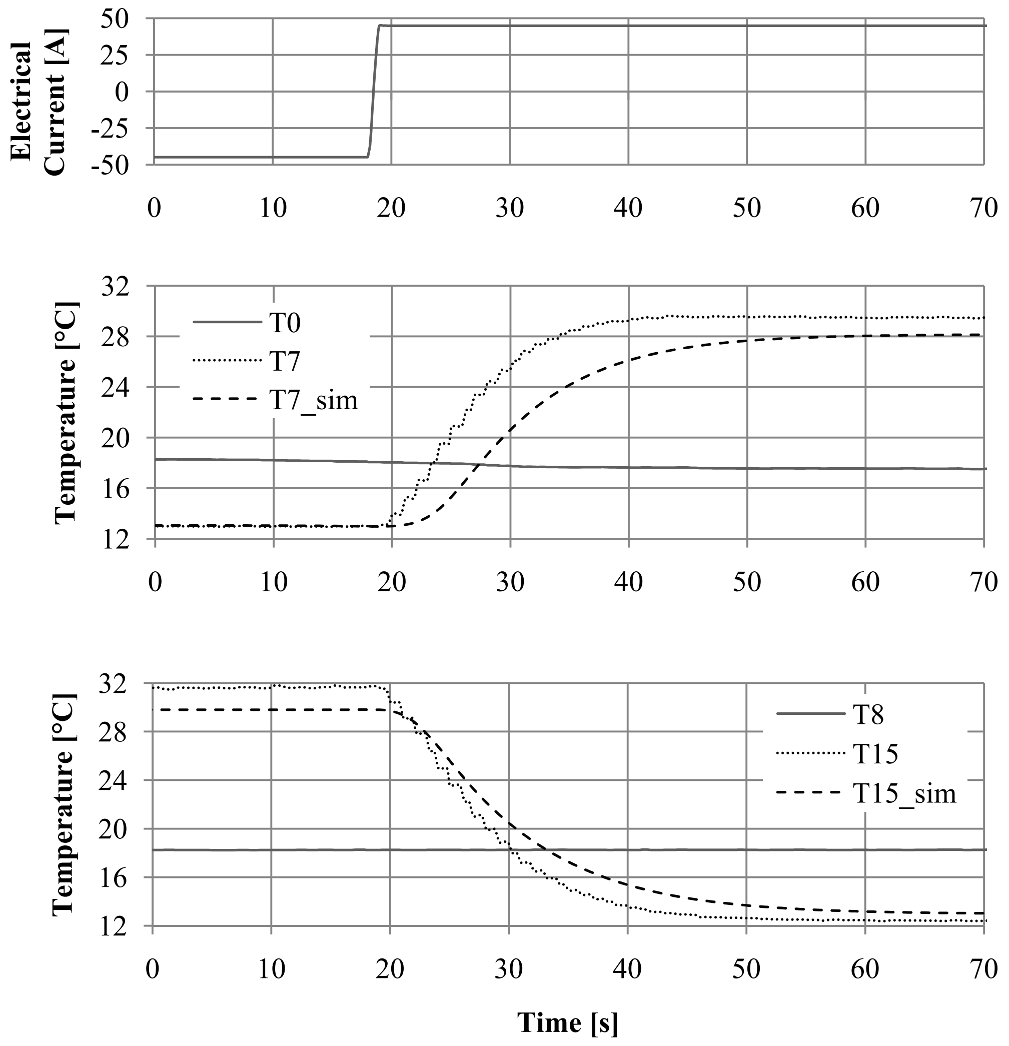


Figure 12: Measured and simulated water temperatures at inlets and outlets of prototype Peltier heat exchanger for Measurement 4 from Table 1.

7 Conclusions and Outlook

A new model for a Peltier water-water heat exchanger was presented that can be used in transient system simulations. Results from measurements with a prototype heat exchanger were used to validate the new model. Models from the new component mode library TIL [4] were used for many components of the new Peltier heat exchanger model and the new object-based fluid property library TILFluids was used to compute all fluid properties. A new model for Peltier cells was presented that was used to assemble the heat exchanger. The new heat exchanger model demonstrates that TIL can easily be extended to cover a wide range of thermodynamic systems. The presented model can be extended to cover other Peltier heat exchangers. A very interesting alternative concept to be analyzed in the future using simulations and experiments is a refrigerant-air heat exchanger with Peltier modules in between.

References

[1] J. Winkler, V. Aute, B. Yang, and R. Radermacher. Potential benefits of thermoelectric elements used with air-cooled heat exchangers. In Proc. of 2006 International Refrigeration and Air Conditioning Conference at Purdue, volume 1, pages R091.1-R091.8, West Lafayette, July 2006.

[2] K. Zorbas, E. Hatzikraniotis, and K. Paraskevopoulos. Power and Efficiency Calculation in Commercial TEG and Application in Wasted Heat Recovery in Automobile. In Proc. of 5th European Conference on Thermoelectrics, 2007.

[3] K. Bos, R. Huebener, and C. Tsuei. Prospects for Peltier cooling of superconducting electronics. Cryogenics, 38(3):325-328, March 1998.

[4] C. Richter. Proposal of New Object-Oriented Model Libraries for Thermodynamic Systems. Dissertation, TU Braunschweig, to be published in 2008

[5] D. Rowe, editor. Thermoelectrics Handbook, Macro to Nano. Taylor & Francis, 2006.

Dynamic Modeling and Self-Optimizing Control of Air-Side Economizers

Pengfei Li and Yaoyu Li
Department of Mechanical Engineering
University of Wisconsin – Milwaukee
pli@uwm.edu & yyli@uwm.edu

John E. Seem
Building Efficiency Research Group
Johnson Controls, Inc.
john.seem@gmail.com

Abstract

For the heating, ventilating, and air conditioning (HVAC) systems for commercial buildings, there has been a greater demand for reducing energy consumption. The economizers have been developed as a class of energy saving devices that may increase the energy efficiency by taking advantage of outdoor air during cool or cold weather. However, in practice, many economizers do not operate in the expected manner and waste even more energy than before installation. Better control strategy is needed for optimal and robust operation. This paper presents two related aspects of research on dynamic modeling and control for economizers. First, a Modelica based dynamic model is developed for a single-duct air-side economizer. The model development was based on Dymola and AirConditioning Library with some revision on water medium and heat exchanger modeling. Such transient model will lay a more quality foundation for control design. Second, for a three-state operation for air-side economizers, a self-optimizing control strategy is developed based on the extremum seeking control (ESC). The mechanical cooling can be minimized by optimizing the outdoor air damper opening via extremum seeking. Such has much less dependency on the knowledge of economizer model, and thus has more promise for practical operation. In addition, an anti-windup ESC scheme is proposed as an enhancement for the existing ESC techniques. The simulation results validated the effectiveness of the dynamic model of the economizer, demonstrated the potential of using ESC to achieve the minimal mechanical cooling load in a self-optimizing manner, and illustrated the possibility of ESC malfunctioning under actuator (damper) saturation and the capability of anti-windup ESC in preventing such undesirable behavior.

Keywords: *Modelica; transient modeling; economizer; extremum-seeking control*

1 Introduction

Buildings are responsible for a large portion of electricity and natural gas demand. Significant amount of energy consumption for buildings is due to the heating, ventilation and air conditioning (HVAC) systems. Improving the efficiency of building HVAC system is thus critical for energy and environmental sustainability. The economizers have been developed as a class of energy saving devices that may increase the energy efficiency by taking advantage of outdoor air during cool or cold weather [1]. Figure 1 is a schematic diagram of a typical single-duct air-handling unit (AHU) and controller. The AHU has a supply fan, three (outdoor air, relief air and mixed air) dampers for controlling air flow between the AHU and the outdoors, heating and cooling coils for conditioning the air, a filter for removing airborne particles, various sensors and actuators, and a controller that receives sensor measurements (inputs) and computes and transmits new control signals (outputs). The air economizer moves the dampers to let in 100% outdoor air when it is cool but not extremely cold outside. When it is hot outside, the dampers are controlled to provide the minimum amount of outdoor air required for ventilation.

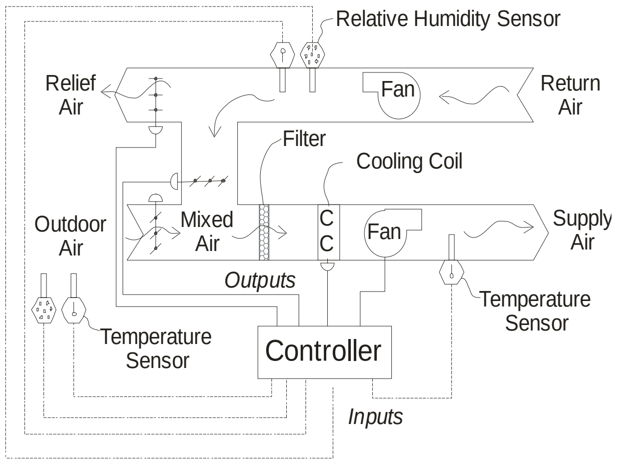


Figure 1: Single duct air handling unit

The American Society of Heating, Refrigerating and Air Conditioning Engineers (ASHRAE) recommends using economizers based on the cooling capacity size and weather characteristics for the building location [2], as described in the Appendix. ASHRAE [3] describes several control strategies for transitioning between 100% outdoor air and the minimum outdoor air required for ventilation. The control strategies are called "high limit shutoff control for air economizer." Following is a list of strategies that can be programmed in a computer control system.

- *Fixed dry bulb temperature*. This strategy compares the outdoor temperature to a transition temperature. If the outdoor air temperature is greater than the transition temperature, then the dampers are controlled for the minimum outdoor air required for ventilation.
- *Differential dry bulb temperature*. This control strategy compares the outdoor and return air temperatures. If the outdoor temperature is greater than the return air temperature, then the dampers are controlled for minimum outdoor air required for ventilation.
- *Fixed enthalpy*. This control strategy measures the outdoor air temperature and relative humidity (RH). Then the outdoor air enthalpy is calculated and compared with a transition enthalpy. If the outdoor air enthalpy is greater than the transition enthalpy, then the dampers are controlled for minimum outdoor air required for ventilation.
- *Differential enthalpy*. This control strategy determines the outdoor and return air enthalpy from measurements of the outdoor and return air temperature and relative humidity. If the outdoor air enthalpy is greater than the return air enthalpy, then the dampers are controlled for minimum outdoor air required for ventilation.

However, in practice, many economizers do not operate as expected and waste even more energy than before installation [4]. Temperature and RH sensor errors can have a large impact on the energy savings or possible penalty of economizer strategies. The National Building Controls Information Program (NBCIP) [5] said, "In the case of economizers, relative humidity and temperature measurements of outdoor and return air conditions are used to calculate the enthalpies of the two air streams. The air stream with the least energy content is then selected to provide building cooling. If one or both of the computed enthalpies is wrong, as can happen when humidity transmitters are not accurate, significant energy penalties can result from cooling of the incorrect air stream." The NBCIP [6] performed long term performance tests on 20 RH sensors from six manufacturers. Nine of the 20 RH sensors failed during the testing. All of the remaining sensors had many measurements outside of specifications. The largest mean error was 10% RH, and the largest standard deviation of the error was 10.2%. The best performing sensor had a mean error of –2.9% RH and a standard deviation of 1.2%. The specifications for the best performing sensor were ±3%. Control strategies not relying on RH measurement would greatly enhance the reliability of economizer operation.

Modeling and optimal control of air-handling units and economizers have been previously studied [7, 8]. However, due to the complex nature of HVAC system operation, the obtained model may not be accurate enough for the optimal operation of an economizer. Therefore, a model based optimal control approach is hardly effective in practice to seek the optimal outdoor air flow for minimizing the mechanical cooling. In contrast, an on-line self-optimizing control approach appears a more suitable option.

This research investigates the application of the extremum seeking control (ESC) [9-13] to optimize the use of outdoor air so as to minimize the energy consumption. The input and output of the proposed ESC framework are the damper opening and power consumption (or equivalently, the chilled water flow rate), respectively. This approach does not rely on the use of relative humidity sensor and accurate model of the economizer for optimal operation. Therefore, it provides a more reliable control strategy for economizer operation. The proposed ESC scheme works as part of a three-state economizer control strategy, as shown in the state diagram in Figure 2. State 1 uses heating to maintain the supply air temperature. In state 2, outside air is mixed with the return air to maintain the supply air at a given setpoint. In state 3, the extremum seeking control is used to control the dampers to minimize the mechanical cooling load. Also, the dampers must be controlled to guarantee enough outdoor air inflow to satisfy the ventilation requirement for the rooms. Figure 3 shows the control regions for different outside air conditions on a psychometric chart. The return air condition was 75 °F and 50% relative humidity, the cooling coil was ideal, and the minimum fraction of outdoor air to supply air was 0.3. The heating region is for state 1, the free cooling region is for state 2, and the three regions that need mechanical cooling are combined into state 3.

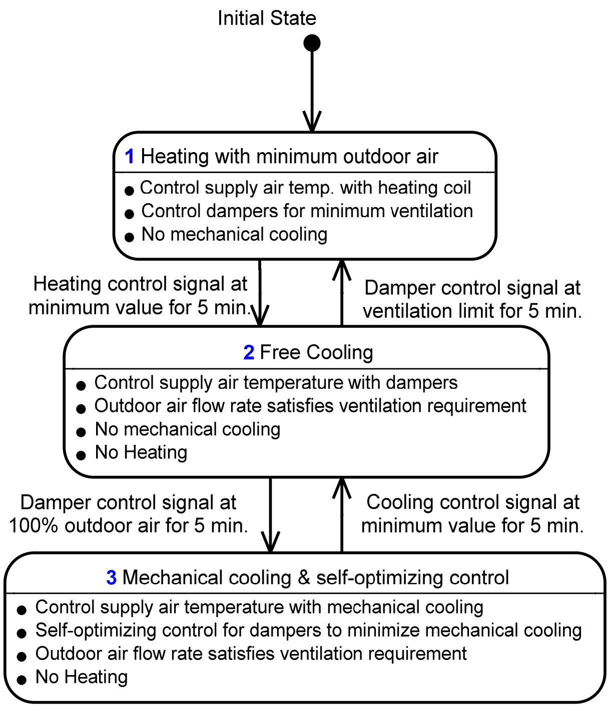

Initial State
1 Heating with minimum outdoor air
● Control supply air temp. with heating coil
● Control dampers for minimum ventilation
● No mechanical cooling
Heating control signal at minimum value for 5 min.
Damper control signal at ventilation limit for 5 min.
2 Free Cooling
● Control supply air temperature with dampers
● Outdoor air flow rate satisfies ventilation requirement
● No mechanical cooling
● No Heating
Damper control signal at 100% outdoor air for 5 min.
Cooling control signal at minimum value for 5 min.
3 Mechanical cooling & self-optimizing control
● Control supply air temperature with mechanical cooling
● Self-optimizing control for dampers to minimize mechanical cooling
● Outdoor air flow rate satisfies ventilation requirement
● No Heating

Figure 2: State transition diagram for the proposed control strategy.

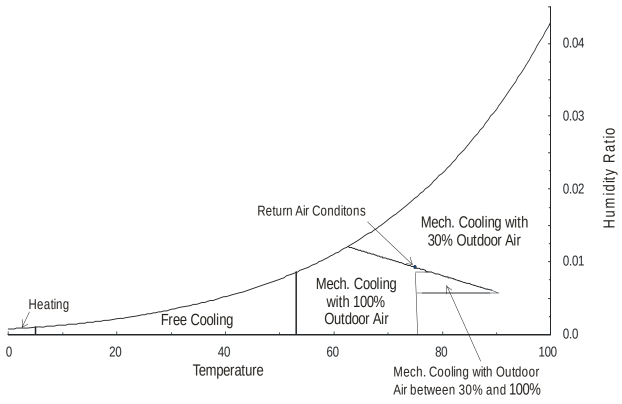


Figure 3: Control states for different outside air conditions for an ideal coil with return conditions 75 °F and 50% RH.

The proposed control scheme has the following advantages over existing economizer strategies:

- *Energy Savings*. Using ESC will lead to energy savings because the dampers will be controlled to minimize the mechanical cooling load. Also, the proposed strategy will save energy because it is not dependent on unreliable RH sensors.
- *Lower installed costs* because the proposed strategy does not require the outside air or return air temperature or RH sensors.
- *Lower maintenance costs* because the temperature and RH sensors do not need to be calibrated.

In addition to the ESC application for economizer control, an enhancement on the ESC is proposed: an anti-windup ESC scheme against damper (actuator) saturation. Due to the inherent integral action incorporated in the ESC loop, the integral windup due to the damper saturation would disable the ESC, as will be shown in Section 3. The back-calculation scheme is applied to the ESC loop to achieve the anti-windup capability.

In order to design and simulation the proposed control strategy, a quality dynamic model of economizer is needed. In this study, an economizer simulation model was developed in Modelica. Dynamic modeling of HVAC equipment has attracted increasing attention in recent years. A summary of previous work in dynamic modeling of vapor compression equipment was presented in [14, 15]. According to [15], the modeling regimes could be mainly classified as two categories: reference models and lumped models. The reference models are designed to best fit the underlying physics of the system, but will often involve partial differential equations (PDE) and high system order. In contrast, the lumped models will lead to lower order ordinary differential equations (ODE) based on some simplifications and/or space discretization. In particular, the first category of models requires extensive dynamic information from the heat exchanger. The finite-volume method was studied by MacArthur [16] but with simplifications in decoupling thermal responses from pressure responses, which may result in less accurate mass distribution predictions. This issue was latter resolved by MacArthur and Grald [17] from combining the mass and balance equations, where the pressure responses are involved. Nyers and Stoyan [18] modeled an evaporator using the approach of finite-difference. Williatzen et al [19] employed a profile assumption for the variation of refrigerant state within each phase region. Recently, Rasmussen [20] presents an novel modeling approach with more freedom of selecting the system states and is claimed to be equivalent to the common method of simplifying the governing PDEs to the desired ODEs. Zhou [21] developed a so-called forward model which was capable of solving the governing differential equations concerning energy storage and transfer in a cooling and dehumidifying coil. The lumped models have also been studied by several authors for simulation and control purposes [22-24]. Besides the modeling approaches involved, the fact that different time scales of the system dynamics are either interwoven or distinctive to a large extent yet poses another serious challenge to the dynamic modeling of HVAC. However, limited study has been done so far on developing effective and efficient dynamic models that are capable of handling system dynamics with different time scales and simultaneously satisfying research purposes ranging from dynamic analysis and control design of subsystems (e.g. AHU) to building energy savings and comfort. As for AHU modeling in particular, ASHRAE [15] said some of the quickest phenomena occur in the AHUs (coils, humidifiers, and economizers), when simulating such subsystems, realistic dynamics have to be considered for all components involved: heat and mass exchangers, fans, ducts and pipes, sensors and actuators. Compared to the control oriented transient analysis which features small time-scale, the energy saving and human comfort evaluation are coped with in a much larger time scale, but require accurate energy balances. For instance, the cooling coil usually has the slowest transient among the four major components in the vapor compression system, and thus has the largest impact on transient performance. It is necessary to consider mass distribution within the cooling coil as a function of time and space and this requires transient mass balances to allow for local storage [14]. On the other hand, for an AHU, the cooling coil is among the quickest responding components. Their transient response may significantly interact with closed loop controllers [15]. Thus, the multiple-time-scale compatibility is important for the dynamic/transient modeling of HVAC systems.

Control development for many HVAC systems, e.g. the economizer in this study, would not be possible without accurate and computationally efficient dynamic/transient models. Most simulation tools for HVAC systems have been based on steady-state modeling. Dynamic modeling and simulation is still in the research phase and not mature yet. Modelica, as an object-oriented language for physical modeling, has demonstrated its great capability for simulating multi-physical systems. Several Modelica based simulation packages have been developed, e.g. the Thermal-Fluid Library [25], the AirConditioning Library [26], the Modelica_Fluid Library [25] and the HITLib [27]. The AirConditioning Library is capable of handling both steady-state and transient simulation, however, it was mainly designed for automotive air conditioning systems. Some components need to be modified for modeling building HVAC systems such as economizers. In this study, a dynamic model of a single-duct air-side economizer is developed using Dymola (Version 6.1) developed by Dynasim [28], the Modelica Fluid Library (MFL) and the AirConditioning Library (ACL) (Versions 1.4 and 1.5) developed by Modelon [26].

The remainder of this paper is organized as follows. Section 2 describes the dynamic economizer model design. The details of ESC design are described in Section 3, along with the anti-windup ESC. Finally, simulation results that demonstrate the effectiveness of ESC and the two proposed enhancements are presented in Section 4.

2 Dynamic Economizer Model Design

The dynamic model of economizer was developed based on the Dymola 6.1, the MFL and the ACL 1.4 and 1.5. In addition to adopting the standard components in the commercial packages, we have made the following development: modification of water property calculation for the heat exchanger model, initialization with pressure-temperature pair, mixing box, and fan. Figure 4 shows the economizer model that we have developed in Dymola, which includes air ducts, air mixing box, fans, cooling coil, and a room space. The air duct model was adopted from the MFL. It allows detailed pressure drop calculation due to wall friction. The air mixing box model contains two sub-components: the air-mixing plenum and the damper module. The air-mixing plenum was developed using the splitter model from the MFL, while the damper module was developed by ourselves. We have also developed a fan model based on the similarity factors [29]. In addition, the cooling coil was developed based on the evaporator model from the ACL. A water medium model *CoolWater* was developed based on the IAWSP-IF97 formulation [30], and compared with the water medium model developed in the ACL. The pressure-temperature pair was used for both initialization and state derivation with the consideration of practical HVAC operation. Finally, a mixing volume model from the MFL was used to represent a room space.

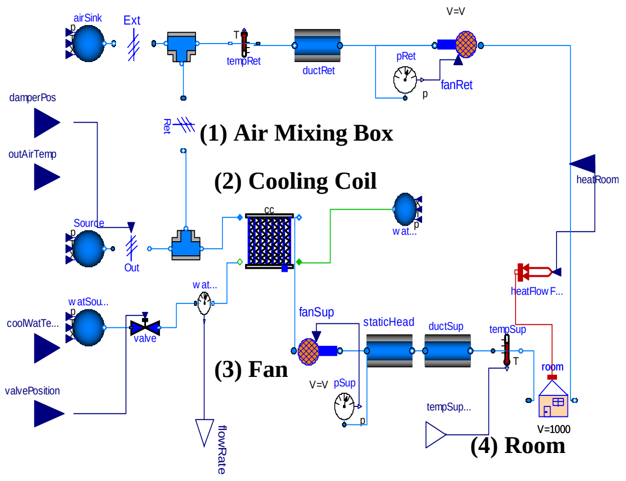


Figure 4: Dymola layout of the economizer model.

2.1 Air Mixing Box

The air mixing box is a component of the AHU that mixes the outdoor air and the return air from the conditioned indoor space. It consists of a damper module (outdoor, return and exhaust dampers) and an air-mixing plenum. The fraction of the outside air is regulated by the outdoor damper whose command signal is interlocked with the exhaust and return air damper. The supply air flow rate is kept as consistent as possible to ensure proper pressure balance at the building side. In addition, to provide adequate ventilation, the minimal OAD opening is limited by an actuator. The damper model was developed based on the work by Tan and Dexter [31]. The pressure drop across the dampers is given by $P_{loss} = R_{damp}m_{air}^2$, where m_{air} is the mass flow rate of the air through the dampers and R_{damp} is the resistance of the damper given by

$$R_{damp} = \begin{cases} R_{open}\exp[k_d(1-\alpha)] & if\ \alpha \geq 0.3333 \\ \dfrac{R_{open}}{3.0[(1/3-\alpha)L_d + 0.0429\alpha]^2} & if\ \alpha < 0.3333 \end{cases} \tag{1}$$

where α is the fractional opening of the damper (0 for fully closed and 1 for fully open), k_d is a constant depending on the type of blades used, R_{open} is the resistance of fully open dampers, and L_d is the leakage when the damper is fully closed. In Eq. (1), there exists a slight discontinuity of the damper resistance around 0.3333. It was smoothed by a third order polynomial covering the interval of [0.2833, 0.3833]. The four coefficients of the polynomial were determined with the two functional values and two derivative values at 0.2833 and 0.3833. The air-mixing plenum was formulated on the basis of the splitter model from the MFL.

2.2 Fan

Two fan models are employed in this study. The first fan model was based on the pump model from the MFL. The only change was on the medium flowing through, from water to the moist air. The second fan model was developed based on the similarity factor model in [29]. The relationship between the flow factor and pressure factor is given by

$$\varphi = C_1 + C_2\psi + C_3\psi^2 \tag{2a}$$

$$\varphi = \frac{Q}{AU} \tag{2b}$$

$$\psi = \frac{\Delta P_{total}}{\Delta P_{dynam,\ periph}} \tag{2c}$$

where $A = (\pi D^2)/4$, $U = (\pi DN)/60$, $\Delta P_{vel} = (\rho v^2)/2$, $v = Q / A_{ex}$, $\Delta P_{total} = \Delta P_{stat} + \Delta P_{vel}$, φ is the flow factor, ψ is the pressure factor, Q is the flow rate, A is the reference area, A_{ex} is the exhaust area, D is diameter of the impeller, v is the velocity of the outflow air, N is the rotation speed in rpm, ΔP_{stat} is the static pressure, ΔP_{vel} is the velocity pressure, and $\Delta P_{dynam,\ periph}$ is the peripheral dynamic pressure. C_1, C_2 and C_3 are coefficients of the polynomials relating the flow and pressure factors, which are fitted to the manufacture's fan performance data by the least-square estimation. A limited proportional-integral (PI) controller is used to regulate the rotation speed of the supply fan to maintain the static pressure of the supply air duct at the setpoint. In addition, the rotation speed of the return fan is synchronized by another limited PI controller, with the reference setting satisfying the steady-state equilibrium of overall flow rate. This is a simplified treatment, and it is being improved by a more accurate treatment described in the work by Tan and Dexter in [31] which considered the building over-pressurization and leakage flow.

2.3 Cooling Coil

Cooling coil is the most important component between the primary plant (e.g. chiller) and the air distribution system. As mentioned earlier, the cooling coil is among the quickest responding components in AHU and it also responds to the quickest perturbations. Therefore, the transient behavior of cooling coil may have significant effect on closed loop control performance [15].

Since Version 1.4, the ACL has developed a group of heat exchanger models that are capable of simulating both transient and steady-state operations. The dynamic energy and mass balances are formulated based on the finite-volume method. The number of discretization at the refrigerant side is proportional to that for the solid wall and the air side. The heat conduction in the solid wall is modeled as a one-dimensional problem perpendicular to the fluid flow direction. In particular, the simulation results of a cross-counter flow evaporator model used in an automotive R134a-system had been validated in an experiment conducted by Chrysler [32]. The measured data were compared with the simulation results of the medium properties and the steady-state heat transfer rates, for three sets of boundary conditions given by the mass flow rate, the inlet temperature, the inlet enthalpy, and the relative humidity of the ambient air. The heat transfer rates had good consistency while the refrigerant-side pressure drop and the air-side water condensing needed improvement.

There were some challenges to directly use the heat exchanger model from ACL for the cooling coil component in the economizer model. In the ACL Version 1.5, the choices of state variable pairs include pressure-enthalpy, density-temperature, and mass-internal energy. Such choices are suitable for the air flow and two-phase refrigerants in the automotive refrigeration systems. However, for the building HVAC systems, especially for cooling coils in the AHU, the working medium is typically single-phase, i.e. water. Also, the temperature range is limited to the ambient temperature variation. Therefore, it is necessary to reformulate the existing heat exchanger model in the ACL to accommodate the specific needs in building HVAC systems.

2.3.1 Medium Model Design and Implementation

An accurate water medium model is critical for the transient simulation of cooling and heating coils in the AHU. For the water property calculation, there are mainly two international standard formulations, namely, IAPWS95 [33] and IAPWS-IF97 [30, 33]. The former was developed for scientific computation, while the latter was developed for industrial applications. Prior to the release of Version 1.4, the ACL had included a large set of medium models for many refrigerants, but not the water medium. Since Version 1.5, the ACL has adopted a lookup-table (LUT) based incompressible fluid (water) medium model for heat exchanger modeling. However, it may have the following drawbacks. First, in the control volumes, pressure responses are decoupled with thermal responses, which may lead to inaccurate mass distribution predictions. Second, incompressible water model will also result in inaccurate pressure drop calculations, which will in turn affect the heat transfer property calculations.

To validate the accuracy of different formulations of water property model, the IF-97 formulae based model (abbreviated as "IF-97 model" later) and the LUT based incompressible water model (abbreviated as LUT model later) were compared with the IAPWS-95 standard. The FLUIDCAL program developed by Wagner's group was used to obtain the IAPWS-95 based water properties [34]. For Dymola 6.1, the water medium in Modelica_Media follows the IF-97 model, while the water medium of ThermoFluidPro in the ACL Version1.5 follows the LUT model. The comparison was conducted in the temperature range from 274.15 K to 373.15 K with an increment of 5 K, and the pressure input was set 5 bars for all cases. Table 1 summarizes the maximum errors of several properties based on the IF-97 and LUT models relative to those derived from the IAPWS-95 standard. Figures 5 through 8 compare the relative errors of the IF-97 and LUT models in density, specific entropy, C_p and C_v, respectively. Note that C_p and C_v are assumed identical in the LUT model. More discrepancies were observed for entropy and C_v.

Table 1: Water Properties Based on IF-97 and LUT Models Relative to IAPWS-95 Standard

Water Property	Maximum Relative Error (%)	
	IF-97	ACL1.5
Density	0.0015	0.09
Specific Entropy	0.018	28.223
C_p	0.052	0.189
C_v	0.075	11.833

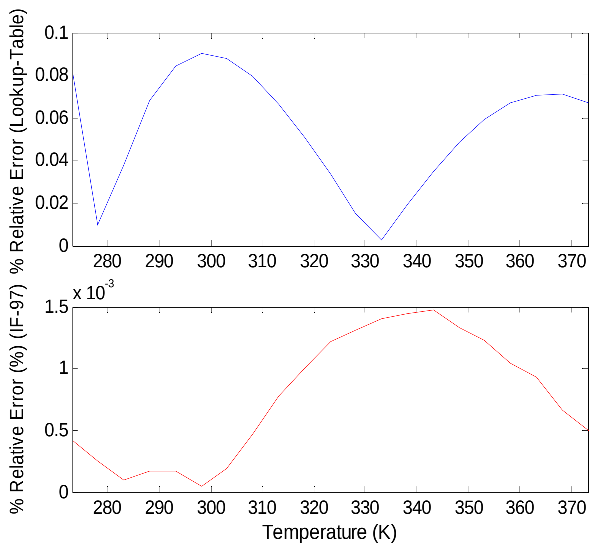


Figure 5: Density errors of the IF-97 and LUT models relative to the IAPWS95 standard

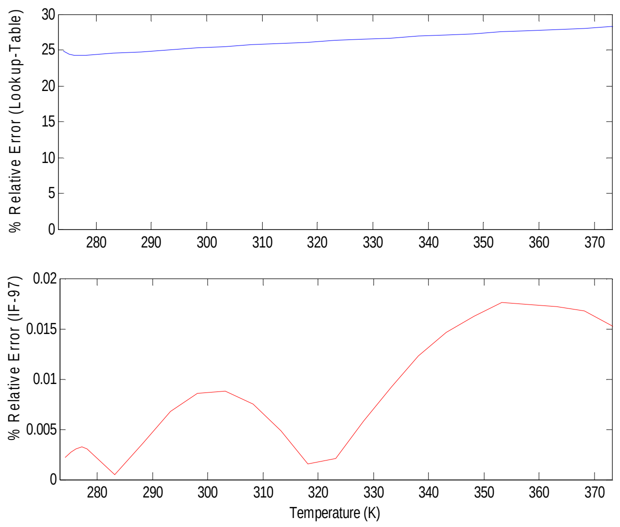


Figure 6: Specific entropy errors of the IF-97 and LUT models relative to the IAPWS95 standard

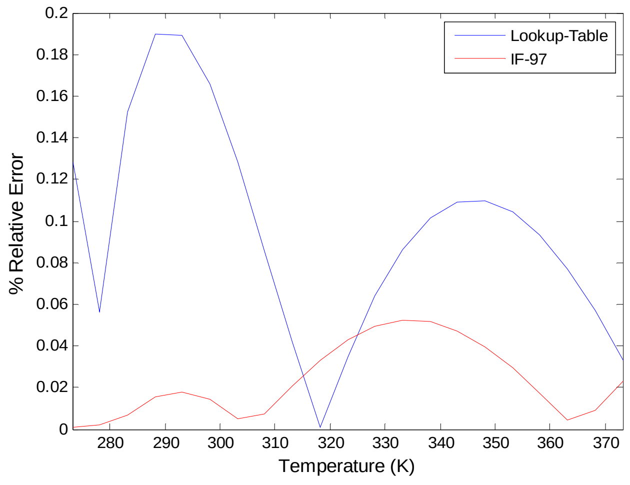


Figure 7: C_p errors of the IF-97 and LUT models relative to the IAPWS95 standard

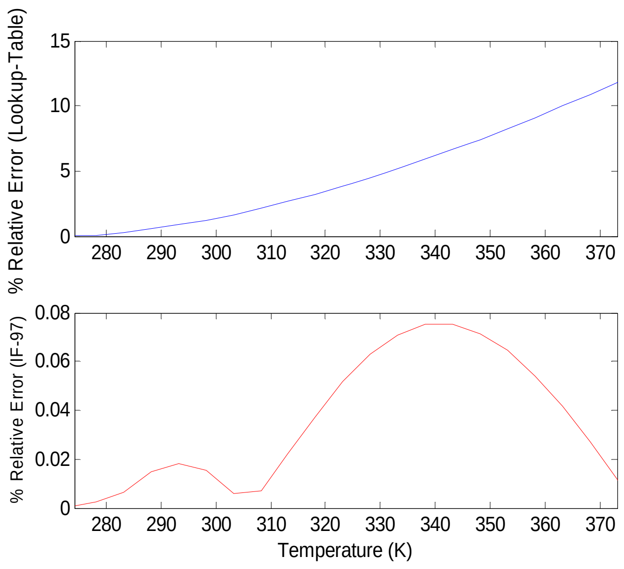


Figure 8: C_v errors of the IF-97 and LUT models relative to the IAPWS95 standard

Within the Modelica_Media Library, a group of *waterIF97* models have been well defined to compute the physical properties for water in the liquid, gas and two-phase regions based on the IF-97 formulae. However, there are several technical issues to use these *waterIF97* medium models directly in the functions of ACL. First, *waterIF97* medium model contains both single- and multiple-phase calculations, in which the multiple-phase portion is not needed for this application. In addition, earlier development in the ACL is well compatible with the automotive air conditioning systems whose working medium are various kinds of refrigerants. The composition is a critical argument contained in most functions developed in the ACL. For cooling and heating coils in the AHU, the single-phase water is the only working medium to deal with. The composition argument in the existing ACL functions results in significant inconvenience. For the single-phase water medium used in the heating/cooling coils, it would be more convenient to remove the composition argument.

Second, the medium property computation in the ACL covers both single- and multiple-phase processes, which are involved not only in the balance equations of the dynamic control volumes, but also in the calculations of various thermodynamic states, such as density, enthalpy and specific heats, which are irrelevant to the dynamic states of the control volumes. In addition, there are a lot of computations related to multiple-phase processes. A process/device involving only the single-phase water medium, such as the heating/cooling coil in the AHU, is a much simpler case. If we can remove all irrelevant computations, the resultant computational efficiency will be greatly improved.

Thirdly, the refrigerants used by typical automotive air conditioning systems are modeled on the basis of the Helmholtz functions with density-temperature as the pair of state variables. In many HVAC applications, it would be more convenient if the water properties are based on the pairs of pressure-temperature or pressure-enthalpy. In addition, for physical property calculations in the control volumes, the users can access the medium functions only at hierarchically higher levels, which limits the customization or reformulation of these functions for particular applications, especially when the user-preferred pair of state variables is not supported in the existing package.

To address the above issues, we decided to develop a simpler and more efficient water model, named as *CoolWater*, based on *Modelica_Media.Interfaces.PartialMedium*. The basic formulation of the *CoolWater* model was obtained from [35]. In particular, all redundant and conflicting variables and options in the original *waterIF97* model were either removed or modified, e.g. the BaseProperties code. To be consistent with the coding style and physical property calculations preserved in the ACL, several IF-97 based low-level medium functions and utilities were adopted from the Modelica_Media Library.

A heat exchanger model was developed based on the *CoolWater* medium described above. Heat exchanger modeling is generally considered the most computationally intensive entity in a refrigeration system [36]. To properly adapt the *CoolWater* model to the refrigerant side, equations in the dynamic control volumes should be rewritten, but the change should not degrade the overall inheritance structure and exactness of the heat exchanger model. Since the uppermost hierarchical structure of the heat exchanger is composed of only a few lines of code, the work of implementing single-phase water model should begin from the most rudimentary control volumes. In the development phase, different choices of state-variable pairs were first compared and evaluated in order to achieve both engineering convenience and numerical efficiency. It was stated in [37] that the mass-internal-energy pair could decrease the numerical efficiency. The density-temperature pair was considered by [38] a bad choice in the liquid region for compressible fluids due to the amplification of numerical error.

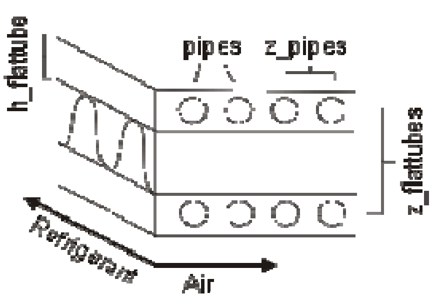
Currently, the state-variable pairs of pressure-temperature and pressure-specific-enthalpy have been formulated into the heat exchanger model for comparison purpose. The techniques of state variable transformations were performed in the dynamic balance equations for pressure-temperature and pressure-specific-enthalpy, respectively [38, 39]. The corresponding partial derivatives appeared in the balance equations could be computed using rudimentary IF-97 functions. To ensure consistent and convenient initialization, the pressure-temperature pair (compared to the pressure-enthalpy pair) has been added into the initialization options, since temperature is easier to set for HVAC operation rather than some other variables such as enthalpy.

2.3.2 Validation of Cooling Coil Model

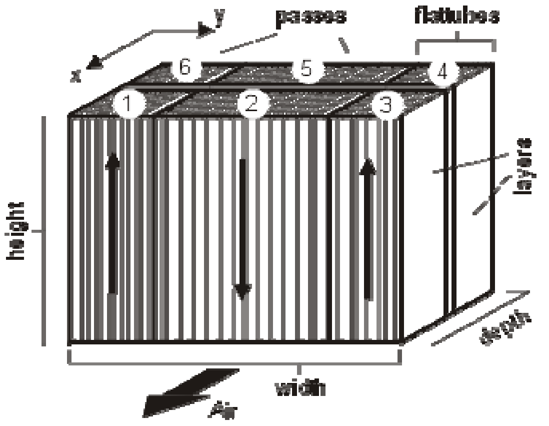
A cooling coil model was derived from the heat exchanger model described in the previous section. To validate this model, two comparisons were conducted: comparison of pressure-temperature and pressure-enthalpy and comparison of our cooling coil model and the cooling coil in ACL Version 1.5.

As described in [38], the advantage of using the pressure-temperature pair is that there are many medium property models which are explicit in this state pair. The sensitivity of using this state pair needs to be checked. It is known that using different dynamic state variable pairs may change the numerical sensitivity of the corresponding thermodynamic equations of state (EOS). For a bad choice of state pair, even a small error in one of variables of the state pairs may lead to a large error to other variables calculated from EOS. To address such concern, the pressure-temperature and pressure-enthalpy pairs were compared with an example cooling coil model.

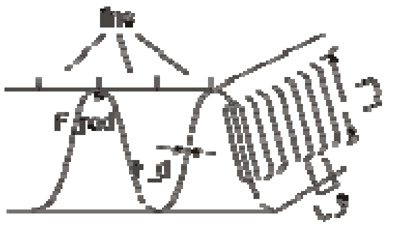
The cooling coil adopted a flat tube louvered fin heat exchanger model given in the ACL. It consists of louvered fins and extruded microchannel flat tubes, both made of aluminum. The schematic diagrams in Figure 9 show the geometry and flow pattern for the cooling coil model.



(a) Flow pattern of water and air



(b) Six-pass cooling coil with vertical flow of cooling water and cross flow of air



(c) Geometry of the triangular louvered fin

Figure 9: Schematic diagrams for the example cooling coil [40]

On both sides of the wall, several parallel flow channels are lumped into one uniform flow path. The cooling water path through the component is treated as one pipe flow with circular cross section and one air element associated with each flow segment. Each air element is further discretized along its flow direction. The total depth and height were set to be 0.06 m and 0.21 m, respectively. The width of the cooling coil could be then calculated from the known number of flat tubes and dimension of the flat tubes and fins. For the water side, as shown in Figure 9(b), there are 15 flat tubes in the 2nd and 5th flow passes, and 10 flat tubes in the each of the remaining flow passes. The dimension of the flat tubes could be determined through three parameters: height of one flat tube, center to center distance of two adjoining flat tubes, and the number of pipes in one flat tube. They were set to be 1 mm, 10 mm and 20, respectively. The wall thickness and radius of each pipe were set to be 0.1 mm and 0.4 mm. At the air side, the shape of the louvered fins was set to be triangular. The fin dimensions are summarized in Table 2.

Table 2: Dimensions of the louvered fins

Fin Dimension	Parameter Setting
Number of fins per 0.1 m	80
Louver length (mm)	7
Louver pitch (mm)	1.4
Louver angle (°)	28
Fin thickness (mm)	0.1
Fin radius (mm)	0.4

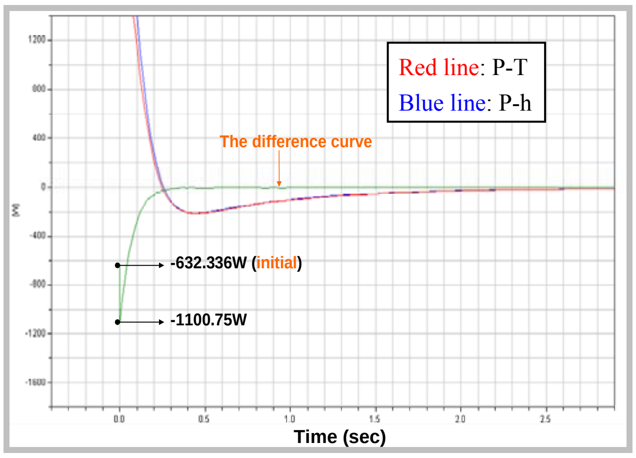


Figure 10: Internal energy in the 3rd control volume

For the two state pairs, the inlet air conditions were set identical. The flow rate, temperature and relative humidity of the inlet air were set to be 0.2 kg/s, 313K and 60%, respectively. For the water side, the chilled water flow rate was kept as 0.3 kg/s. For the pressure-temperature state pair, the initial temperature was set to 292.146 K. To be consistent with this setup, the inlet specific enthalpy was set to 8×10^4 kJ/kg for the pressure-enthalpy state pair. The total discretization number at the air side and water side was set to be 12 and 6, respectively. Figures 10 through 12 show the simulation results from our cooling performance test. The difference curves shown in the plots are the calculated numerical differences between these twos state variable pairs. The results indicate that the differences are noticeable only in the region of numerical transient responses, i.e. 0 to 0.5 seconds, which is not harmful to the overall transient and steady-state solutions.

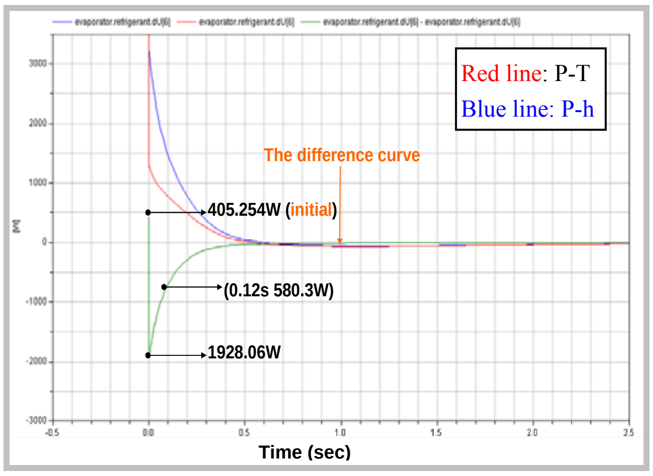


Figure 11: Internal energy in the 6th control volume

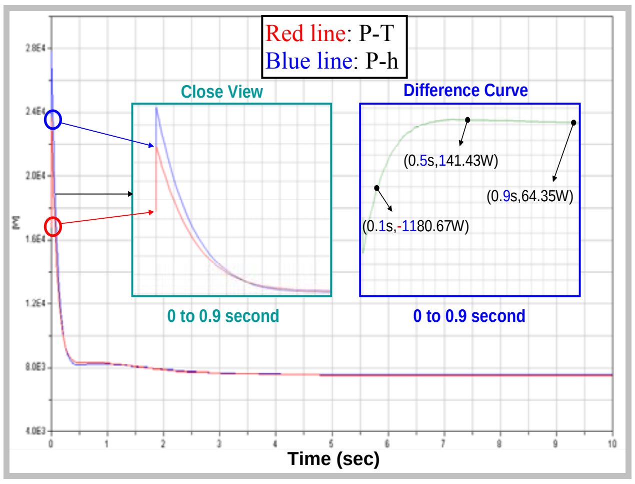


Figure 12: Total heat transferred from the heat exchanger

A further study was then performed to benchmark our development with the ACL Version 1.5. The heat exchanger model from ACL Version 1.5 was equipped with the LUT water model. In our case, the *CoolWater* model was used and pressure-temperature was selected as the state variable pair. The geometric configuration of the cooling coils was reinforced to be the same in the two cases. A similar cooling performance test was conducted, the initial air flow rate was 0.0675 kg/s and the air temperature and RH were given by 303.15 K and 60%, respectively. For the water side, the chilled water flow rate and initial temperature was kept as 0.1 kg/s and 293.15 K respectively. As shown in figure 13, the inlet temperatures at the water and the air sides respectively experienced ramp changes in sequence: at 30 second, the inlet water temperature first ramped to 298.15 K within 20 second, and then the inlet air temperature ramped to 308.15 K at 75 second within 20 second as well. Again, the total numbers of discretization at the air and water sides were set as 12 and 6, respectively. Figures 14 through 16 compare the simulation results of the two cases in terms of the specific enthalpy, the internal energy and the total heat transfer rate, respectively. The maximum relative error was found to be around 0.5%. For this single heat exchanger model test in our study, the computation time using the IF-97 model was about 50% more than that using the LUT model in the ACL Version 1.5.

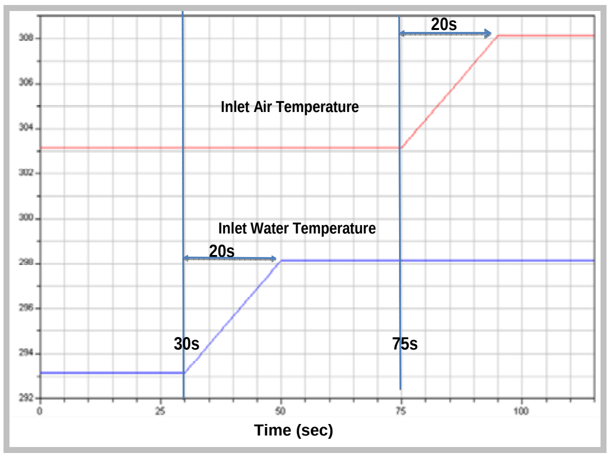


Figure 13: Sequential ramp changes of inlet water and air temperatures

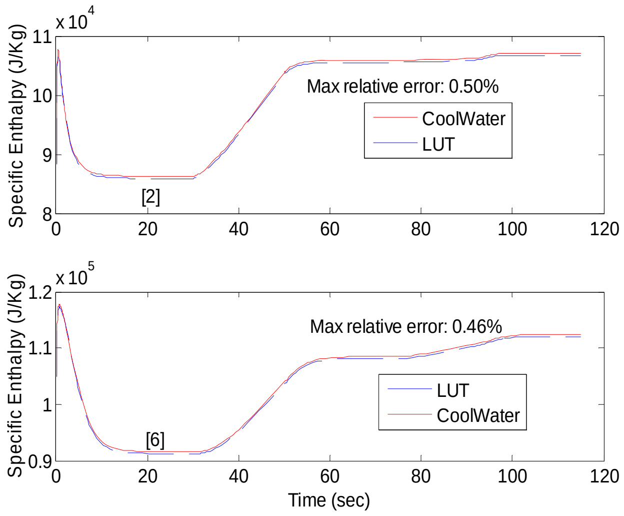


Figure 14: Specific enthalpy in the 2nd and 6th control volumes

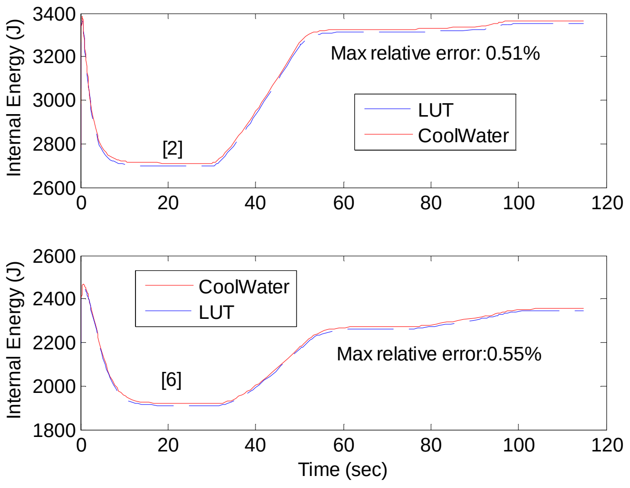


Figure 15: Internal Energy in the 2nd and 6th control volumes

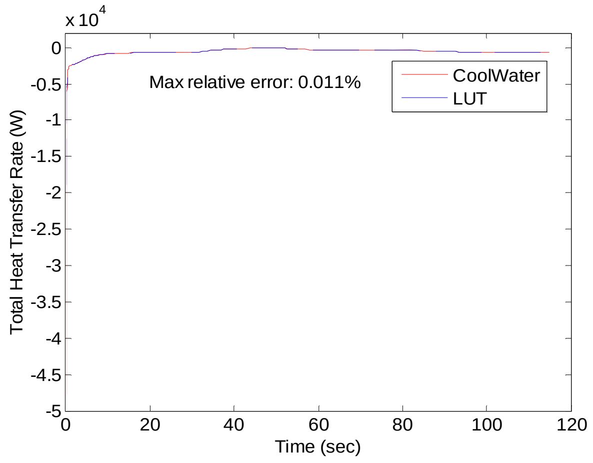


Figure 16: Total heat transfer rate at heat exchanger

3 Extremum Seeking Control (ESC) of Economizer Operation

3.1 Overview of ESC

The extremum seeking control deals with the on-line optimization problem of finding an optimizing input $u_{opt}(t)$ for the generally unknown and/or time-varying cost function $l(t, u)$, where $u(t) \in \mathbb{R}^m$ is the input parameter vector, i.e.

$$u_{opt}(t) = \arg\min_{u \in \mathbb{R}^m} l(t,u) \,. \tag{3}$$

Figure 17 shows the block diagram for a typical ESC system [41]. The measurement of the cost function $l(t, u)$, denoted by $y(t)$, is corrupted by noise $n(t)$. The transfer function $F_I(s)$ denotes the linear dynamics of the mechanism that command the control or optimization parameter vector $u(t)$. $F_O(s)$ denotes the transfer function of the sensor dynamics that measure the cost function, which is often a low-pass filter for removing noise from the measurement.

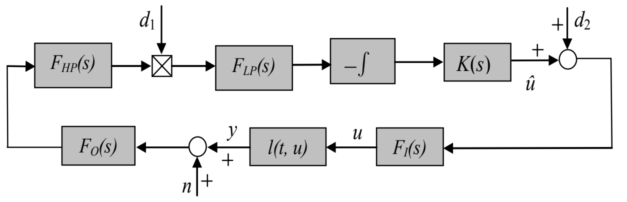


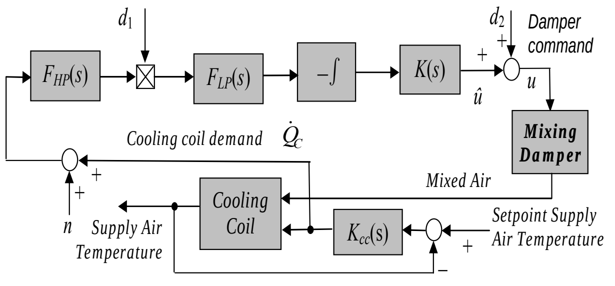
Fig. 17: Block diagram of extremum seeking control

The basic components of the ESC are defined as follows. The dithering and demodulating signals are denoted by $d_1^T(t) = [\sin(\omega_1 t) \quad \cdots \quad \sin(\omega_m t)]$ and $d_2^T(t) = [a_1 \sin(\omega_1 t) + \alpha_1 \quad \cdots \quad a_m \sin(\omega_m t) + \alpha_m]$, respectively, where ω_i are the dithering frequencies for each input parameter channel, and α_i are the phase angles introduced intentionally between the dithering and demodulating signals. The signal vector $d_2(t)$ contains the perturbation or dither signals used to extract the gradient of the cost function $l(t, u)$. These signals work in conjunction with the high-pass filter $F_{HP}(s)$, the demodulating signal $d_1^T(t) = [\sin(\omega_1 t) \quad \cdots \quad \sin(\omega_m t)]$ and the low-pass filter $F_{LP}(s)$, to produce a vector-valued signal proportional to the gradient $\frac{\partial l}{\partial u}(\hat{u})$ of the cost function at the input of the multivariable integrator, where $\hat{u}$ is the control input based on the gradient estimation. By integrating the gradient signal, asymptotic stability of the closed loop system will make the gradient vanish, i.e., achieving the optimality. Adding compensator $K(s)$ may enhance the transient performance by compensating the input/output dynamics. For a detailed explanation of ESC, consult references [12, 13, 41].

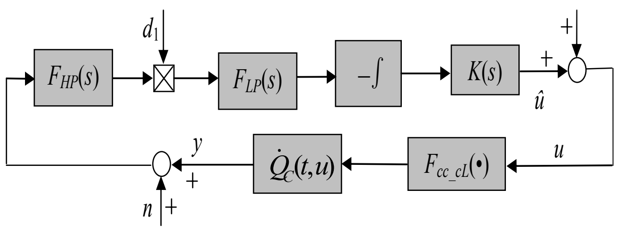
The earliest version of ESC can be dated back to Leblanc's work in 1922 [42]. There was great interest in this subject in 1950s and 1960's [10, 11, 43]. The research conducted by Krstić and his coworkers in the past decade ignited a resurgence of extremum-seeking control [12, 13]. Krstić and Wang first provided the stability proof for general SISO nonlinear plants based on averaging and singular perturbation methods [12]. More design issues were addressed in another paper by Krstić [13]. Later, the stability proof was extended to discrete-time situation [44]. The proposed ESC framework has been applied to various applications, such as maximizing biomass production rate [45], maximizing pressure rise in axial flow compressor [46], minimizing acoustic pressure oscillation to enhance combustion stability [47], minimizing the power demand in formation flight [48], and minimizing limit cycling [49], among others. The extremum seeking control was also studied along different paths. Özgüner and his coworkers combined ESC with sliding mode control [50-52] to study the vehicle ABS control. Based on the assumption of quadratic functional form with a finite number of parameters, Banavar developed an ESC scheme with an adaptation procedure of on-line identifying the parameters in the assumed function [53-55].

3.2 ESC for Energy Efficient Operation of Economizers

The ESC based economizer control is illustrated in Figure 18. The economizer control can be considered as a dual-loop structure. The inner loop is the supply air temperature control for the cooling coil, which has faster dynamics. The outer loop is the damper opening tuning for minimizing the cooling coil demand, which is realized with an ESC framework. The nonlinear performance mapping is from the outdoor air damper opening to the cooling coil demand, and the input dynamics are effectively the closed loop dynamics for supply air temperature control. In the three-state economizer operation scheme, as described in Section 1, the ESC is used for state 3 where mechanical cooling is required.



(a) Detailed block diagram



(b) Simplified block diagram

Figure 18: ESC based economizer control

3.3 Extremum Seeking Controller Design

Typical ESC design needs to determine the following parameters: the dither amplitude α, the dither frequency ω and phase angle ϕ, the high pass filter $F_{HP}(s)$, the low pass filter $F_{LP}(s)$, and the dynamic compensator $K(s)$. Based on averaging analysis, the dither frequency should be relatively large with respective to the adaptation gain, but should not be too large to trigger unmodeled dynamics and make the system more sensitive to measurement noise. Also, if the dither frequency is well out of the bandwidth of the input dynamics, the roll-off in the magnitude response will slow down the convergence [13]. Therefore, dither frequency ω_d is typically chosen to be just a moderate value smaller than the cut-off frequency of the input dynamic as long as it is enough to separate the time scales of the dither signal and the inner loop dynamics. Generally, the dynamic compensator should be designed based on the dither signal, adaptation gain and the frequency responses of the input dynamics. Particularly, a proper proportional-derivative (PD) action can increase the phase margin of the input dynamics and thus make the inner loop more stable. However, extreme values of the adaptation gain, especially the derivative gain, will make the system unnecessarily affected by noise and thus destabilize the system. Further design guidelines are summarized as follows.

1) The dither frequency must be in the passband of the high pass filter and the stopband of the low pass filter, and it should be below the first cut-off frequency of the tuning schemes $F_I(s)$.
2) The dither amplitude should choose to be sufficiently small.
3) The dither phase angle should choose to satisfy $\theta = -\frac{\pi}{2} < \angle F_I(j\omega) + \alpha < \frac{\pi}{2}$ and it is desirable to design the phase angle $\theta = -\frac{\pi}{2} < \angle F_I(j\omega) + \alpha < \frac{\pi}{2}$ such that θ is close to zero.

3.4 Anti-windup ESC

Actuator saturation is often encountered in control systems. To our best knowledge, the issue of actuator saturation has not been discussed for extremum seeking control. For the economizer control, the actuator saturation will happen when it is cool or hot outside. For instance, when the outdoor air is around 53°F, the outdoor air damper will be positioned fully open to allow 100% outdoor air to enter the AHU. When it is warmer than 100 °F, the damper will be closed to a minimum opening which only maintain the lowest ventilation for indoor air quality [56]. In other words, the optimal reference input is not inside the saturation limit, but rather at either limit point. Transition between the ESC operation and the non-ESC operation is affected by the saturation issue. The averaging analysis of ESC [43] showed that, at a large time scale, the ESC can be deemed as a linear system regulating the gradient signal with a PI controller. When saturation presents in the ESC loop, integrator wind-up is unavoidable and, in consequence, leads to the undesirable windup phenomena. Later in Section 4.3, a simulation study will show that, due to the windup issue, the ESC action may be totally disabled even when the air condition changes to a point demanding its re-activation. It is thus necessary to modify the standard ESC structure in order to avoid integrator windup.

There has been much work reported in the field of anti-windup control (AWC) [57, 58]. In order to keep the simple nature of ESC, a back-calculation method is proposed as in Figure 19, following the spirit of the references [58-60]. The difference between the input and output of the actuator is fed back to the input end of the integrator through some gain factor. Our simulation results have demonstrated that this method works well to prevent the integrator windup in ESC system. Future research needs to investigate the design guidelines for the proposed anti-windup ESC. The analysis will be based on combining the existing method for back-calculation AWC and the averaging analysis [61, 62].

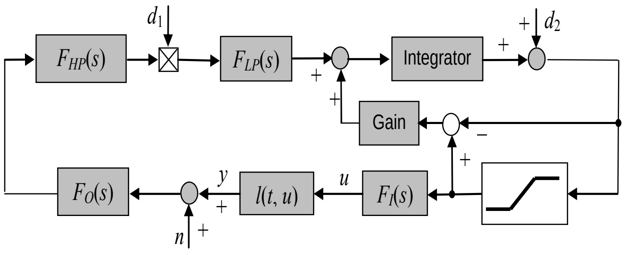


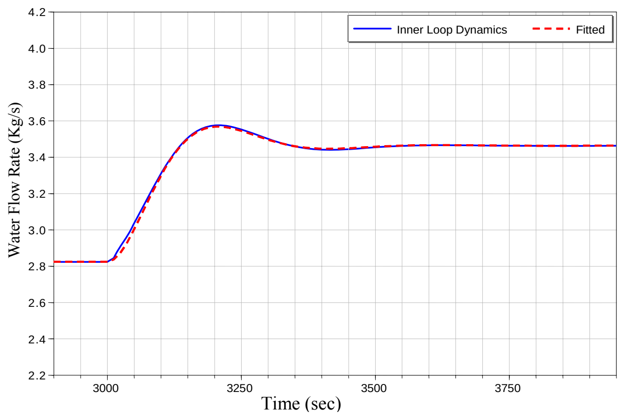
Figure 19: Block diagram for the anti-windup ESC

4 Simulation Study

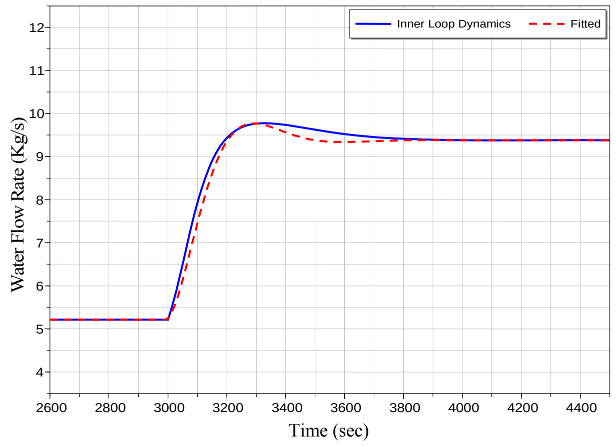
The proposed extremum seeking control schemes were simulated with the Modelica based economizer model described in Section 2. The economizer model was used to identify the system dynamics and then illustrate the ESC schemes presented in the Section 3. At the point of writing this paper, the condensation computation from ACL 1.5 has not been incorporated into the cooling coil model due to the software licensing delay. Only the dry air can be simulated. The simulation results in the following are presented for illustration purpose. More rigorous treatment will be done after the condensation computation is made up to deal with moist air.

4.1 ESC with Standard Design

As previously stated, the control objective in this study is to minimize the chilled water flow rate of the cooling coil by tuning the OAD opening. The input dynamics from the OAD opening to the chilled water flow rate was approximated based on several open-loop simulations. Fast (20 second) ramp input was used to approximate step input in order to remove the output jitter due to the inner loop PI control. Two fast-ramp responses are shown in Figure 20, which shows the second-order system behavior across the whole range of operating conditions.



(a) Damper opening from 100% to 70%



(b) Damper opening from 50% to 20%

Figure 20: Chilled water flow rate output under fast ramp change of outdoor air damper position

The following second order model was assumed to fit the fast-ramp test data:

$$F_I(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{4}$$

where ω_n is the undamped natural frequency and ζ is the damping ratio. The damping ratio ζ was first approximated by the percent overshoot suggested in [63], then the 10% to 90% rise time T_r was estimated. The ω_n can then be approximated via the following relationship with T_r and ζ [63]:

$$T_r = \frac{2.16\zeta + 0.60}{\omega_n} \tag{5}$$

which is accurate for $0.3 \le \zeta \le 0.8$.

A group of tests indicate that ω_n ranged from 0.0108 to 0.021 rad/sec. As a conservative approximation, ω_n was chosen to be 0.011 rad/sec. The damping ratio was estimated from the percent overshoot and was determined as 0.6. To properly separate the dither signal and plant dynamics, the dither frequency ω_d is selected as one tenth of the natural frequency. Next, the following high pass filter $F_{HP}(s)$ was selected:

$$F_{HP}(s) = \frac{s}{s + 0.0001} \tag{6}$$

which has a unit gain at the ω_d. The low pass filter was designed as

$$F_{LP}(s) = \frac{0.0006^2}{s^2 + 2\cdot 0.6\cdot 0.0006 s + 0.0006^2} \tag{7}$$

which has approximately 10dB and 20dB attenuation at ω_d and $2\omega_d$, respectively. To be consistent with the phase lag introduced by the input dynamics $F_I(s)$, the dither phase α was selected as 0.5π (radian), which makes $\theta = \angle F_I(j\omega) + \alpha \approx 0.1°$. The dither amplitude was chosen to be 10%.

The designed ESC was tested with a fixed operating condition. To be consistent with standard economizer design conditions, the supply air temperature is controlled at 55°F and the return air temperature is maintained around 75°F by providing a constant heat input to the indoor space. The system was started at minimal OAD opening (20%) to ensure adequate indoor air quality, and the ESC controller was turned on at about 3000 seconds to bring the system the optimum. The optimal OAD opening in this study is 100% since the outdoor air was set to 286K (55°F), which is always lower than the return air temperature 297K (75°F). Therefore, the more outdoor air intake, the less cooling water needed to be consumed. Figure 21 shows the time histories of the optimized chilled water flow rate and OAD opening. The obtained steady-state results are very close to the optimum since the assumed condition is mechanical cooling with optimal OAD opening at 100%.

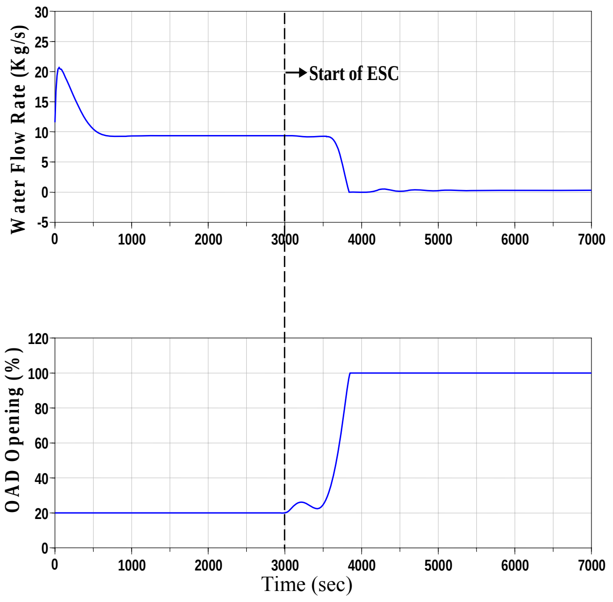


Figure 21: Tuning results of ESC with standard ESC.

4.2 Anti-Windup ESC

Another simulation study was conducted to verify the effectiveness of the proposed anti-windup ESC. Assume that a 20% damper opening is the minimum requirement for indoor air quality, and thus this was set as the lower saturation limit. The upper saturation limit was 100%. In the simulation study as shown in Figure 21, the initial outdoor air damper opening was set at 20%, the same as the lower saturation limit. The initial air temperature was again set to be 286 K. Figure 22 shows the integrator windup phenomenon when only the general ESC scheme was applied. Driven by the ESC, the damper opening was increased from 20% to 100% which was the corresponding achievable optimal setting. Then the outdoor air temperature was suddenly increased to 310 K (36.85 °C) at 6000 seconds, the new optimal opening was supposed to be below the lower saturation limit. However, the results show that the ESC was unable to respond to such change with reducing the damper opening. Rather the damper appeared “stuck” at the previous position. In comparison, as shown in Figure 23, applying the back-calculation based anti-windup ESC starting from 3000s effectively solved this problem. Therefore, the proposed anti-windup ESC scheme is shown to be able to handle the saturation windup problem.

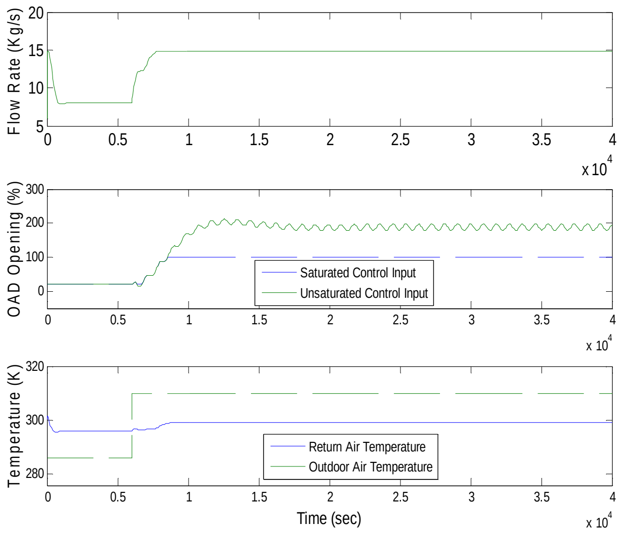


Figure 22: Standard ESC under actuator saturation

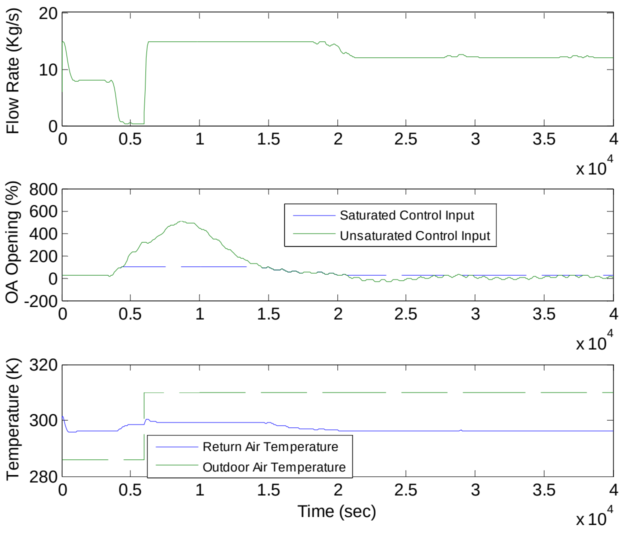


Fig. 23: Anti-windup ESC under damper saturation

5 Conclusions

In this paper, a Modelica based dynamic simulation model was developed for a single-duct air-side economizer based on Dymola and AirConditioning Library. In order to make the cooling coil modeling more effective and computationally efficient, revision was made on the water medium model and the associated heat exchanger modeling. An ESC algorithm was proposed as part of a three-state economizer operation, which aims to minimize mechanical cooling load for the economizer operation in commercial buildings. The standard ESC algorithm was enhanced by an anti-windup ESC scheme against damper (actuator) saturation. Simulations were conducted to search for the optimal outdoor air damper opening for standard ESC and the anti-windup ESC. The simulations results demonstrated the effectiveness of using ESC for tuning the outdoor air damper position to minimize mechanical cooling load. The proposed enhancement was also validated through the simulation results.

References

1. EPA, *Energy Cost and IAQ Performance of Ventilation Systems and Controls.* EPA Report, 2000. **EPA-4-2-S-01-001**.
2. ASHRAE, *Energy Standard for Buildings Except Low-Rise Residential Buildings*. 2004, American Society of Heating, Refrigerating and Air-Conditiioning Engineers, Inc.: 1791 Tullie Circle NE, Atlanta, GA 30329.
3. ASHRAE, *90.1 User's Manual ANSI/ASHRAE/IESNA Standard 90.1-2004*. 2004, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.
4. Financial Times Energy, I. *Design Brief Economizers*.
5. NBCIP, *Product Testing Report: Duct-Mounted Relative Humidity Transmitters*. 2004, National Building Controls Information Program.
6. NBCIP, *Product Testing Report Supplement: Duct-Mounted Relative Humidity Transmitters*. 2005, National Building Controls Information Program.
7. Song, L. and M. Liu, *Optimal outside airflow control of an integrated air-handling unit system for large office buildings.* Journal of Solar Energy Engineering, Transactions of the ASME, 2004. **126**(1): p. 614-619.
8. Guo, C., Q. Song, and W. Cai, *A neural network assisted cascade control system for air handling unit.* IEEE Transactions on Industrial Electronics, 2007. **54**(1): p. 620-628.
9. Blackman, P.F., *Extremum-Seeking Regulators*, in *An Exposition of Adaptive Control*. 1962, Pergamon Press.
10. Sternby, J., *Extremum Control Systems: An Area for Adaptive Control?*, in *Preprints of the Joint American Control Conference*. 1980: San Francisco, CA.
11. Åström, K.J. and B. Wittenmark, *Adaptive control*. 2nd ed. 1995, Reading, Mass.: Addison-Wesley. xvi, 574 p.
12. Krstić, M. and H.-H. Wang, *Stability of extremum seeking feedback for general nonlinear dynamic systems.* Automatica, 2000. **36**(4): p. 595-601.
13. Krstić, M., *Performance improvement and limitations in extremum seeking control.* Systems and Control Letters, 2000. **39**(5): p. 313-326.
14. Bendapudi, S. and J.E. Braun, *A Review of Literature on Dynamic Models of Vapor Compression Equipment*. 2002.
15. Bourdouxhe, J.-P., M. Grodent, and J. Lebrun, *Reference Guide for Dynamic Models of HVAC Equipment*, ed. M. Geshwiler. 1998: American Society of Heating, Refrigerating & Air-Conditioning Engineers.
16. MacArthur, J.W. *Analytical Representation of the Transient Energy Interactions in Vapor Compression Heat Pumps*. 1984. ASHRAE, Atlanta, GA, USA.
17. MacArthur, J.W. and E.W. Grald, *Prediction of cyclic heat pump performance with a fully distributed model and a comparison with experimental data.* ASHRAE Transactions, 1987. **Vol. 93, Part 2**.
18. Nyers, J. and G. Stoyan, *Dynamical model adequate for controlling the evaporator of a heat pump.* International Journal of Refrigeration, 1994. **17**(2): p. 101-108.
19. Willatzen, M., N.B.O.L. Pettit, and L. Ploug-Sorensen, *General dynamic simulation model for evaporators and condensers in refrigeration. Part I: Moving-boundary formulation of two-phase flows with heat exchange.* International Journal of Refrigeration, 1998. **21**(5): 398-403.
20. Rasmussen, B.P. and A.G. Alleyne, *Control-oriented modeling of transcritical vapor compression systems.* Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, 2004. **126**(1): 54-64.
21. Zhou, X., *Dynamic modeling of chilled water cooling coils*, in *School of Mechanical Engineering*. 2005, Purdue University.
22. Svensson, M.C., *Studies on on-line optimizing control with application to a heat pump*. 1994, The University of Trondheim.
23. He, X., S. Liu, and H. Asada. *Modeling of vapor compression cycles for advanced controls in HVAC systems*. 1995. Seattle, WA, USA.
24. Wang, S., J. Wang, and J. Burnett, *Mechanistic model of centrifugal chillers for HVAC system dynamics simulation.* Building Services Engineering Research and Technology, 2000. **21**(2): p. 73-83.
25. Modelica. 2007 [cited; Available from: http://www.modelica.org/.
26. Modelon. 2007 [cited; Available from: http://www.modelon.se/.
27. Videla, J.I. and B. Lie. *A New Energy Building Simulation Library*. in *Proceedings of Modelica 2006*. 2006.
28. Dynasim. http://www.dynasim.se/dynasim.htm.

29. Lebrun, J., *Variable Speed Fan*, *http://cbs.lbl.gov/diagnostics/model_library*. 2004.
30. Wagner, W., et al., *The IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam.* Journal of Engineering for Gas Turbines and Power, 2000. **122**(1): p. 150-180.
31. Tan, H. and A. Dexter, *Estimating airflow rates in air-handling units from actuator control signals.* Building and Environment, 2006. **41**(10): p. 1291-1298.
32. Limperich, D., et al. *System Simulation of Automotive Refrigeration Cycles*. in *Proceedings of the 4th International Modelica Conference*. 2005. Hamburg.
33. Wagner, W. and A. Pruß, *The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use.* J. Phys. Chem. Ref. Data 2002. **Volume 31**(Issue 2): p. 387-535.
34. Wagner, D.-I.W. and D.-I.U. Overhoff, *FLUIDCAL*. 2004. Basic Package Water (IAPWS-95) for calculating the thermodynamic properties of H_2O.
35. Hilding Elmqvist, H.T.a.M.O. *Object-Oriented Modeling of Thermo-Fluid Systems*. in *Proceedings of the 3rd International Modelica Conference*. 2003. Linköping.
36. Bendapudi, S., *Development and evaluation of modeling approaches for transients in centrifugal chillers*. 2004, Purdue University
37. Torge Pfafferott, G.S. *Implementation of a Modelica Library for Simulation of Refrigeration Systems*. *Proceedings of the 3rd International Modelica Conference*. 2003. Linköping.
38. Tummescheit, H., *Design and Implementation of Object-Oriented Model Libraries using Modelica*, in *Department of Automatic control*. 2002, Lund Institute of Technology.
39. Eborn, J., *On Model Libraries for Thermo-hydraulic Applications*, *Department of Automatic Control*. 2001. Lund Inst. of Technology.
40. Modelon AB, *AirConditioning Library Users Mannaul Version 1.5*. 2007.
41. Rotea, M.A., *Analysis of multivariable Extremum Seeking Algorithms.* Proceedings of the American Control Conference, 2000. p. 433-437
42. Leblanc, M., *Sur l'electrification des Chemins de fer au Moyen de Courants Alternatifs de Frequence Elevee.* Revue Generale de l'Electricite, 1922.
43. Tsien, H.S., *Engineering cybernetics.* 1954, New York,: McGraw-Hill. xii, 289p.
44. Choi, J.Y., et al., *Extremum seeking control for discrete-time systems.* IEEE Transactions on Automatic Control, 2002. **47**(2): p. 318-323.
45. Wang, H.H., M. Krstic, and G. Bastin, *Optimizing bioreactors by extremum seeking.* International Journal of Adaptive Control and Signal Processing, 1999. **13**(8): p. 651-669.
46. Wang, H.-H., S. Yeung, and M. Krstic, *Experimental application of extremum seeking on an axial-flow compressor.* IEEE Transactions on Control Systems Technology, 2000. **8**(2): p. 300-309.
47. Banaszuk, A., Y. Zhang, and C.A. Jacobson. *Adaptive control of combustion instability using extremum-seeking*. in *Proceedings of the American Control Conference*. 2000.
48. Binetti, P., et al., *Control of formation flight via extremum seeking.* Proceedings of the American Control Conference, 2002. **4**: p. 2848-2853.
49. Wang, H.-H. and M. Krstić, *Extremum seeking for limit cycle minimization.* IEEE Transactions on Automatic Control, 2000. **45**(12): p. 2432-2437.
50. Drakunov, S., et al., *ABS control using optimum search via sliding modes.* IEEE Transactions on Control Systems Technology, 1995. **3**(1): 79-85.
51. Yu, H. and Ü. Özgüner. *Extremum-seeking control strategy for ABS system with time delay*. in *Proceedings of the American Control Conference*. 2002.
52. Yu, H. and Ü. Özgüner. *Extremum-seeking control via sliding mode with periodic search signals*. in *Proceedings of the IEEE Conference on Decision and Control*. 2002.
53. Speyer, J.L., et al. *Extremum seeking loops with assumed functions*. in *Proceedings of the IEEE Conference on Decision and Control*. 2000.
54. Banavar, R.N., D.F. Chichka, and J.L. Speyer. *Functional feedback in an extremum seeking loop*. in *Proceedings of the IEEE Conference on Decision and Control*. 2001.
55. Banavar, R.N. *Extremum seeking loops with assumed functions: Estimation and control*. in *Proceedings of the American Control Conference*. 2002.
56. ASHRAE, *ASHRAE standard : ventilation for acceptable indoor air quality*. 2001, Atlanta, GA: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. 34 p.
57. Åström, K.J. and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*. 3rd ed. Prentice Hall information and system sciences series. 1997, Upper Saddle River, N.J.: Prentice Hall. xiv, 557 p.

58. Åström, K.J. and L. Rundqwist. *Integrator windup and how to avoid it*. in *Proceedings of 1989 American Control Conference*. 1989.
59. Fertik, H.A. and C.W. Ross, *Direct digital control algorithms with anti-windup feature.* ISA Transactions, 1967: p. 317-328.
60. Åström, K.J., *Advanced control methods – Survey and assessment of possibilities*, in *Advanced control in computer integrated manufacturing. Proceedings of the thirteenth annual Advanced Control Conference* H.M. Morris, E.J. Kompass, and T.J. Williams, Editors. 1987.
61. Sanders, J.A., F. Verhulst, and J.A. Murdock, *Averaging methods in nonlinear dynamical systems*. 2nd ed. 2007, New York: Springer.
62. Khalil, H.K., *Nonlinear systems*. 3rd ed. 2002, Upper Saddle River, N.J.: Prentice Hall.
63. Dorf, R.C. and R.H. Bishop, *Modern Control Systems (10th Edition)*. 2004: Prentice Hall.
64. DOD, *Heating, Ventilating, Air Conditioning, and Dehumidfying Systems*. 2005, United States Department of Defense.
65. Hydeman, M., et al., *Advanced Variable Air Volume System Design Guide*. 2003, California Energy Commission.

Appendix: Economizer Operation

The American Society of Heating, Refrigerating and Air Conditioning Engineers (ASHRAE) recommends using economizers based on the cooling capacity size and weather characteristics for the building location. ASHRAE [2] classifies climate data based on temperature with a number from 1 to 7, and the letters A, B, and C, which correspond to moist, dry, and marine climates, respectively. Table 1 contains climate zones for 16 cities in the United States. The fourth column (Economizer Requirement) shows the cooling capacity for which an economizer is required by ASHRAE [2]. No economizer is required in weather locations 1A, 1B, 2A, 3A, and 4A. In weather locations 3B, 3C, 4B, 4C, 5B, 5C, and 6B, an economizer is required when the cooling requirement is greater than or equal to 19 kW. In all other weather locations, an economizer is required when the cooling requirement is greater than or equal to 40 kW. ASHRAE [3] describes several control strategies for transitioning between 100% outdoor air and the minimum outdoor air required for ventilation. The control strategies are called "high limit shutoff control for air economizer." Following is a list of strategies that can be programmed in a computer control system.

- *Fixed dry bulb temperature*. This strategy compares the outdoor temperature to a transition temperature. If the outdoor air temperature is greater than the transition temperature, then the dampers are controlled for the minimum outdoor air required for ventilation. ASHRAE [3] said this is the most reliable and simple control strategy since a simple thermostat placed in an outdoor air intake can be used. Table 2 shows the transition temperature for different climatic zones. The U.S. Department of Defense [64] recommends this strategy.
- *Differential dry bulb temperature*. This control strategy compares the outdoor and return air temperatures. If the outdoor temperature is greater than the return air temperature, then the dampers are controlled for minimum outdoor air required for ventilation. This strategy should not be used in the following climatic zones: 1A, 2A, 3A, and 4A. Hydeman et al. [65] said, "Of all of the options, dry bulb temperature controls prove the most robust as dry-bulb temperature sensors are easy to calibrate and do not drift excessively over time. Differential control is recommended throughout California and the sensors should be selected for a through system resolution of 0.5 °F. Dry-bulb sensors work well in all but humid climates, which are not typical in California."
- *Fixed enthalpy*. This control strategy measures the outdoor air temperature and relative humidity. Then the outdoor air enthalpy is calculated and compared with a transition enthalpy. If the outdoor air enthalpy is greater than the transition enthalpy, then the dampers are controlled for minimum outdoor air required for ventilation. ASHRAE [2] recommends a transition enthalpy of 47kJ/kg and at locations with altitudes significantly different than sea level, the transition enthalpy should be determined for 24 °C and 50% relative humidity. This strategy should not be used in the following climatic zones: 1B, 2B, 3B, 3C, 4B, 4C, 5B, 5C, 6B, 7, and 8, due to the problem with humidity sensors.
- *Differential enthalpy*. This control strategy determines the outdoor and return air enthalpy from measurements of the outdoor and return air temperature and relative humidity. If the outdoor air enthalpy is greater than the return air enthalpy, then the dampers are controlled for minimum outdoor air required for ventilation. In 2003, the U.S. General Services Administration required a differential enthalpy economizer for air-handling units with a capacity greater than 3,000 CFM (1,416 LPS) unless the air handling system design precluded the use of an air-side economizer. Regarding the use of differential enthalpy controls, Hy-

deman et al. [65] said, "Differential enthalpy controls are theoretically the most energy efficient. The problem with them is that the sensors are very hard to keep calibrated and should be re-calibrated on an annual or semi-annual basis. Contrary to common perception, enthalpy controls do not work in all climates. In hot dry climates they can hunt and excessively cycle the economizer dampers when the hot dry outdoor air has lower enthalpy than the space(s) at cooling balance point. What happens is that the economizer opens up and the coil is dry, which in turn dries out the space(s) until the return enthalpy goes below the outdoor enthalpy. As a result, the economizer damper closes, the space humidity increases, and the cycle repeats."

Table A.1. Climate zones and economizer requirement for 16 US cities. (q_{cool}: cooling capacity)

Climate	Description	Cities	Economizer Requirement
1A	Very Hot - Humid	Miami, FL	None
1B	Very Hot - Dry	---	None
2A	Hot - Humid	Houston, TX	None
2B	Hot - Dry	Phoenix, AZ	$q_{cool} \geq 40$ kW
3A	Warm - Humid	Charlotte, NC	None
3B	Warm - Dry	Los Angeles, CA	$q_{cool} \geq 19$ kW
3C	Warm - Marine	San Francisco, CA	$q_{cool} \geq 19$ kW
4A	Mixed - Humid	New York, NY	None
4B	Mixed - Dry	Albuquerque, NM	$q_{cool} \geq 19$ kW
4C	Mixed - Marine	Seattle, WA	$q_{cool} \geq 19$ kW
5A	Cool - Humid	Chicago, IL	$q_{cool} \geq 40$ kW
5B	Cool - Dry	Denver, CO	$q_{cool} \geq 19$ kW
5C	Cool - Marine	---	$q_{cool} \geq 19$ kW
6A	Cold - Humid	Minneapolis, MN	$q_{cool} \geq 40$ kW
6B	Cold - Dry	Cheyenne, WY	$q_{cool} \geq 19$ kW
7A	Very Cold - Humid	Ashland, WI	$q_{cool} \geq 40$ kW
7B	Very Cold - Dry	Jackson, WY	$q_{cool} \geq 40$ kW
8	Arctic	Fairbanks, AL	$q_{cool} \geq 40$ kW

Table A.2. Transition temperatures for fixed dry bulb economizer.

Climatic Zones	Transition Equation
1B, 2B, 3B, 4B, 4C, 5B, 5C, 6B, 7B, 8	$T_{OA} > 24^\circ\mathrm{C}$
5A, 6A, 7A	$T_{OA} > 21^\circ\mathrm{C}$
1A, 2A, 3A, 4A	$T_{OA} > 18^\circ\mathrm{C}$

Session 4c

Automotive Applications

Using Modelica for modeling and simulation of spark ignited engine and drilling station in IFP

Masoud Najafi* and Zakia Benjelloun-Dabaghi†

Abstract

Modeling and simulation are becoming more crucial since engineers need to analyze very complex systems composed of several components from different domains. Current tools used in IFP (French Institute of Petroleum) are generally weak in treating multi-domain models because the general tools are block-oriented and thus demand a huge amount of manual rewriting to get the equations in explicit form. The most popular tool used at IFP in simulation of 0D/1D systems and control design area is Simulink. In this paper, we present the use of the Modelica language in modeling and simulation of two industrial applications.

Keywords: Modeling; Modelica; Scicos; SI Engine; Drilling station

1 Introduction

Scilab[1] is a free and open-source software for scientific calculation and Scicos[2] is a toolbox of Scilab that provides an environment for modeling and simulation of hybrid dynamical systems [1, 2]. They can be compared with Matlab and Simulink, respectively. The underlying hybrid formalism in Scicos allows modeling and simulation of very general hybrid dynamical systems, *i.e.,* systems including continuous, discrete-time and event based behaviors.
Scicos supports acausal modeling or modeling physical systems with components. This has been done, in particular, by lifting the causality constraint on Scicos blocks and by introducing the possibility of describing block behaviors in the Modelica language. This extension allows the user to model physical systems described by mathematical formula. Most physical components are more naturally modeled with components simply because physical laws are expressed in terms of mathematical equations [3].
Modelica is a modern object-oriented programing language based on equations instead of assignment statements. Modelica has a multi-domain modeling capability, *e.g.*, electrical, mechanical, thermodynamic, hydraulic, and control systems can be described by Modelica. Modelica programs are built from classes that contain elements, the variable declarations, and equations. In order to write a complicated model easily and efficiently, the model is decomposed into several components. Then, by interconnecting components the model is constructed [4].
In the following sections, we will present two industrial applications: drilling station and spark ignited engine. These applications have been already modeled in Simulink which is a popular tool at IFP mostly used for simulation of 0D/1D systems and control system design. In this paper, we will present the way these applications have been modeled with Modelica and simulated in Scicos.

2 Modeling a drilling station

Modeling in the oil and gas industry is used in several stages of operations, from exploration activity to refining of the crude oil. The purpose of modeling is to improve an understanding of the problems that are usually difficult or expensive to deal with in the real physical system. Drilling a well into a reservoir is an expensive, risky, and time-consuming process. So the problems and malfunctions should be detected as soon as they appear. Most of problems in drilling industry are due to lack of a complete knowledge about the environment and the process. Modeling and simulation are inevitable to detect and control of such problems. In previous works done at IFP a model of drilling station has been developed [5, 6, 7]. The particularity of this work, inspired directly from cited works, lies in using Modelica language and formal computing to

*Masoud Najafi, INRIA-Rocquencourt, Domaine de Voluceau, BP 105, 78153, Le Chesnay, France masoud.najafi@inria.fr

†Zakia Benjelloun-Dabaghi, French institute of Petroleum (IFP), 1 & 4, avenue de Bois-Préau, 92852, Rueil-Malmaison, France zakia.benjelloun-dabaghi@ifp.fr

[1]www.scilab.org

[2]www.scicos.org

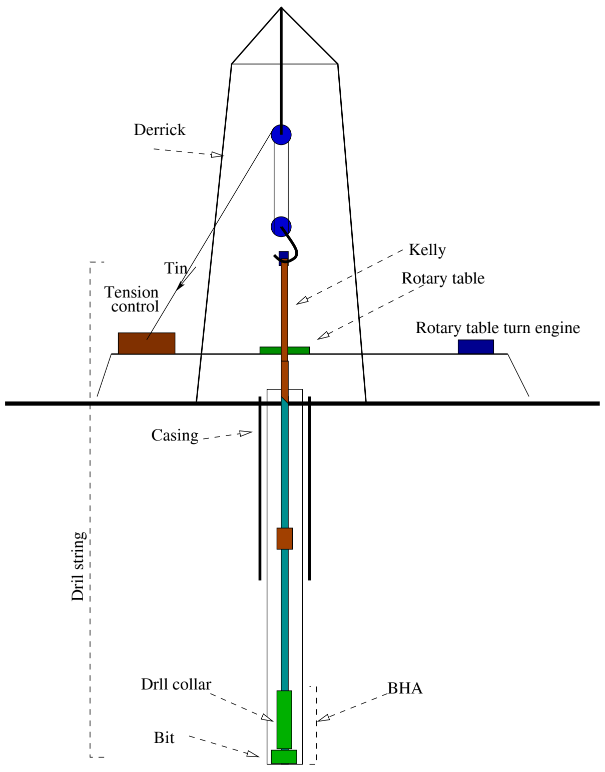


Figure 1: A schematic diagram of a drilling well station.

model and simulate the drilling well station model.

A drilling well station is composed of several parts. The first visible part of a drilling well is the rig which is a structure housing equipments used to drill into underground reservoirs for water, oil, or etc. The basic components of a rotary drilling rig are the derrick and hoist, rotary table, kelly, drill pipe, bit, and pump as shown in Fig. 1.

The *derrick* is the support structure that holds the drilling apparatus and the drill string. The *drill string* consists of rotary table, kelly, drill pipe, drill collars, and bit. The *rotary table* is a circular table in the derrick floor which is rotated by the electrical or diesel engines. The kelly is a four or six-sided pipe that passes up through the rotary table and transfers rotary motion of the rotary table to the drill string. When rotated by the rotary table, the kelly is free to be raised or lowered by a cable connected to the top of the derrick down the kelly.

Drill Pipe is always the longest component in a drill string. Typically thousands of meters of drill pipe are used to drill an oil well. Drill pipe is manufactured in segments of 10 meters lengths. The top joint of the drill pipe is connected to the kelly. Bottom joint of the drill pipe is directly connected to larger diameter pipes called *drill collars*. One of the earliest problems drillers encountered in rotary drilling was that of keeping their boreholes straight. The deeper drillers went, the more the boreholes deviated from vertical. Drill collars weigh more than drill pipe and are designed to lower the center of gravity of the drill pipe. This helps control drilling (*e.g.,* making a straight hole) and prevents the pipe from kinking and breaking. Two to twenty drill collars are often used.

The *drilling bit* is the end of the drill string that actually cuts up the rock. The bit screws into the bottom of the drill collars. The most common bit is the tricone bit which has three rotating cones. The cones have teeth that are designed to chip and flake away the rock as the bit is rotated.

2.1 Model of the drilling well

The drilling model is a set of differential equations describing behavior of components of the drilling station, including the bit and the rock interactions. The model should be as simple as possible to explain the desired malfunctions. The diagram in Fig. 2 shows the model composed of four main components: a rig, a drill pipe, a drill collar, and a drilling bit. These component interact with each other via four main variables:

T: the torque that a component applies on another,
F: the force that a component applies on another,
Ω: the angular velocity of a component,
V: the longitudinal velocity of a component.

A more detailed description of components' model will be given in the following sections.

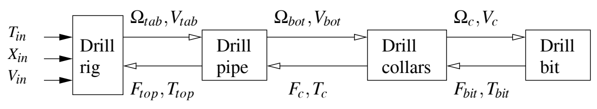


Figure 2: Connecting variables

The model is nonlinear and one-dimensional and provides several bottom/surface transfer functions which can be used for real-time estimation of borehole variables. Furthermore, the model can be used for stability analysis that is extremely important in controlling the drilling process. Another important use of the model is the simulation of transient and steady state behaviors.

2.1.1 Drilling rig

Drilling rigs may have very complex structure varying in form and size. From the modeling point of view the rig imposes the boundary conditions on the drill string structure. A first approach to model the rig is to

consider its geometric structure and the elements that constitute the rig. This may give an exact model, but it would not be practical. Because in fact this model would be very complex and numerically would be so slow that it could not be used in real-time applications. Another problem with this approach is the fact that it cannot be used for another rig.
In [5], the model of two mass-spring-damper has been proposed for the model of longitudinal motions of the drilling rig, as shown in Fig. 3. Although the model is simple, it can provide a very good low frequency response (up to 20 Hz) which is quit enough for our purpose. Further more, when the rig changes, unlike the first method which needs a complete new model, here we need just a new identification for parameters of the model.

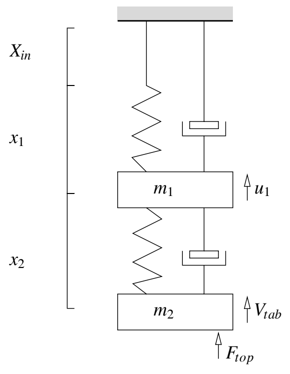


Figure 3: A mechanical model for the drilling rig

The mass m_1 and m_2 can be interpreted as the mass of the hook and the kelly, respectively [5]. F_{top} is the necessary force on the well surface to bore the drill string down into borehole. V_{tab} is the kelly's longitudinal velocity.
Rotary table is modeled as a rotating mass with inertial momentum. In Fig. 2, Ω_{tab}, T_{in}, T_{top} are the angular velocity of the rotary table, the torque applied on the rotary table, and the torque needed to turn the drill string, respectively.

2.1.2 Drill pipe

The drilling pipe is composed of multiple segments which are screwed together to construct a pipe with thousands of meters. Due its length, the drilling pipe exhibits torsional, longitudinal, and lateral motions. In this paper, only longitudinal and torsional motions are considered. Precise modeling of the drilling pipe needs complicated methods such as finite elements. In order to simplify the model, the drilling pipe is discretized to $N = 15$ sections, see Fig. 4. This modeling approach fulfills the precision requirements with a minimum number of variables [6].

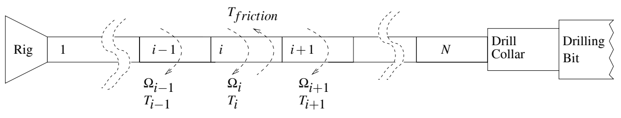


Figure 4: Discretizing the drilling pipe

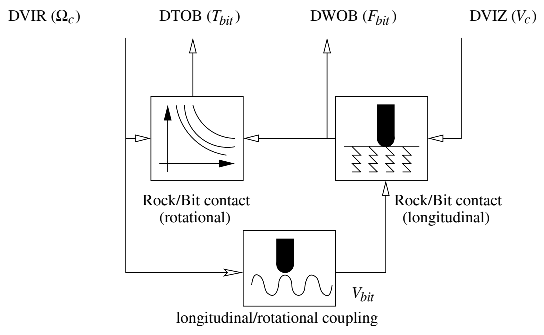
Applying Newton's laws for rotation, we can obtain the model of each segment.

2.1.3 Drill collars

The drill collars are modeled in the same way as the drill pipe. Since, the length of the drill collars are smaller than that of the drill pipe, we do not discretize the drill collars and we consider a single rigid rod. In order to obtain the model of the drill collars, Newton's laws for rotation are used.

2.1.4 Drill bit

The model of the Rig, drill pipe, and drill collars are composed of two uncoupled dynamics: a longitudinal and a rotational dynamics. These two dynamics should be coupled in the drill bit model. Thus, beside the longitudinal and rotational dynamics in the drill bit, a coupling dynamics is necessary. The diagram in Fig. 5 shows these dynamics.



DVIR (Ω_c)
DTOB (T_{bit})
DWOB (F_{bit})
DVIZ (V_c)
Rock/Bit contact (rotational)
Rock/Bit contact (longitudinal)
V_{bit}
longitudinal/rotational coupling

Figure 5: Drilling bit model

In Fig. 5, DVIZ, DVIR, DWOB, DTOB represent the downhole longitudinal velocity, the downhole angular velocity, the downhole weight-on-bit, and the downhole torque-on-bit, respectively. The DTOB (T_{bit}) which is the torque resistance against the rotation due to the rock/bit contact is computed by iso-weight tables. These tables are used to compute the necessary torque as a function of DWOB and DVIR (Ω_c). The

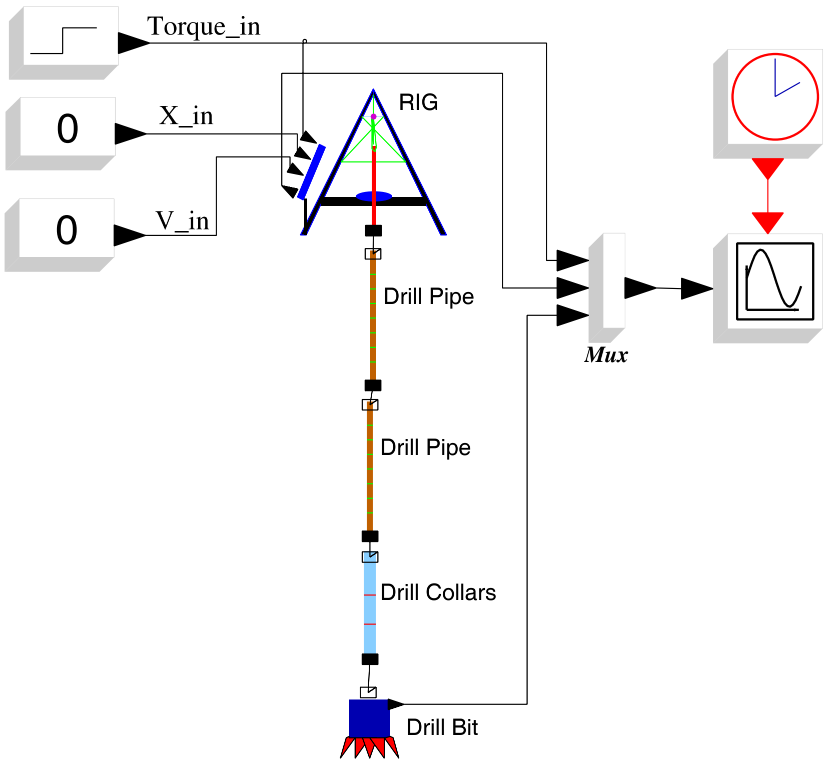


Figure 6: Model of the drilling rig in Scicos

bottom end of the drilling bit is a tricone transforming the rotational motions into longitudinal motions. Ω_c is computed as a function of V_{bit} and the geometric structure of the bit. The WOB is computed as a function of the axial speed of the tricone bit and the longitudinal speed of the bit.

2.2 Simulation example

For each component of the drilling well, *i.e.,* rig, drilling pipe, drill collars, and drilling bit, we have developed a Modelica model. The model of the drilling rig has three control inputs: the rotary table applied torque (T_{in}), the longitudinal speed of the kelly (V_{in}), and initial position of the kelly (X_{in}), see Fig. 2. Thus, the `rig` block has three explicit inputs. Each Modelica model is considered as an implicit block in Scicos. These blocks should be connected to build the model of the drill well.

The Scicos diagram constructed by connecting developed Modelica blocks is shown in Fig. 6. This Scicos model is composed of four implicit blocks and five explicit blocks. `Rig`, `drill pipe`, `drill collar`, and `drill bit` blocks are implicit blocks (written with the Modelica language). There are three explicit blocks providing input variables of the `rig` block. There is a `scope` block to visualize output variables in the model, and a `clock` block to activate the scope block to sample its inputs. Note that the connection type between the implicit blocks is different from that between explicit blocks. These connections represent physical connection, *i.e.,* there is no flow direction.

With the the developed model, the user is able to simulate the model in different situations. Unwanted vibration/oscillation is a well known recurrent phenomenon in rotary drilling that may cause catastrophic bit failures [5, 6, 7, 8, 9]. This phenomenon is the result of torque fluctuations due to Coulomb frictions. These frictions are are included in our model, so it should be possible to simulate this phenomenon which is known as stick-slip. In order to demonstrate this phenomenon, the simulation is started at steady state an-

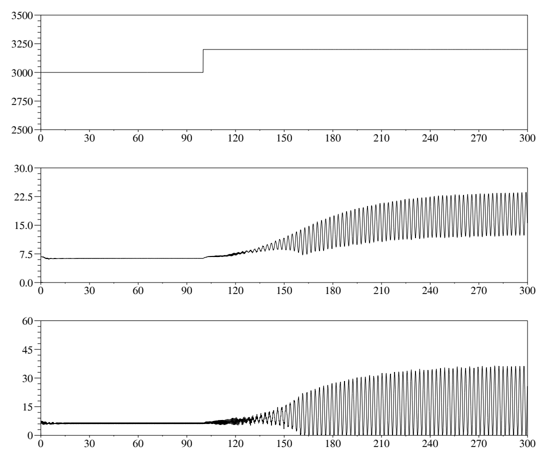


Figure 7: simulation of a drilling station exhibiting oscillations in the rotation speed

gular speed of 66 rpm. The input torque is T_{in}=3000 N.m., X_{in}=-0.03, and V_{in}=0. With these inputs, the system is stable. At t=100 sec, the torque is increased to 3200 N.m. which sets off the oscillation. The simulation result is given in Fig. 7. In the top subplot, the input torque applied on the rotary table is shown. The middle subplot shows the angular velocity of the rotary table, and the bottom subplot shows the angular velocity of the drilling bit.

3 Mean value SI engine

The model of the SI engine described in this section is a nonlinear, low frequency model of a fuel-injected four cylinders SI engine which is generally referred to as a *mean value* model. Mean value engine models attempts to capture dynamics in a time-scale spanning over several combustion cycles. Fast events are not of interest other than their effects on a larger scale. Most cyclic dynamics are modeled by their average value over a cycle. The speed and torque output of the engine and the pressure in the inlet manifold are the aspects of most interest in mean value engine model that we have developed. Mean value engine model generally represents a basis for the development of different engine control strategies.

The model of the overall engine is composed of several components. In order to develop the model of the SI engine easier, the engine subsystems including the air throttle, the intake manifold, exhaust gas recirculation (EGR), the canister purge mechanism, sensor dynamics, combustion chamber, and the load are modeled. Inherent system delays in the four-stroke engine cycle including the induction-to-power stroke delay, effects of the air/fuel ratio or fuel richness are not modeled in this work. The system including fundamental components, sensors, and actuators is illustrated in Fig. 8.

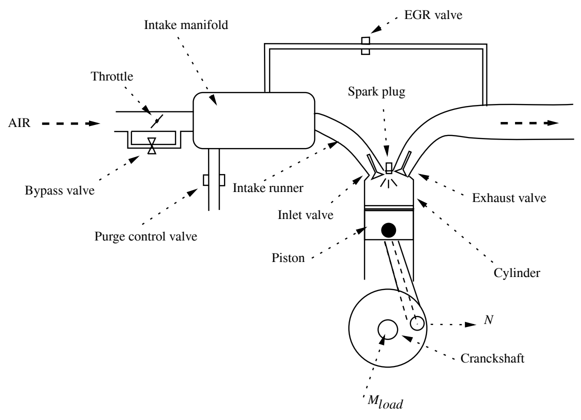


Figure 8: Principle sketch of SI-engine

3.1 Model of the SI engine components

In this subsection, a brief description of the engine components and their corresponding Scicos block is given, more details are given in [10]. These components are shown in Fig. 9.

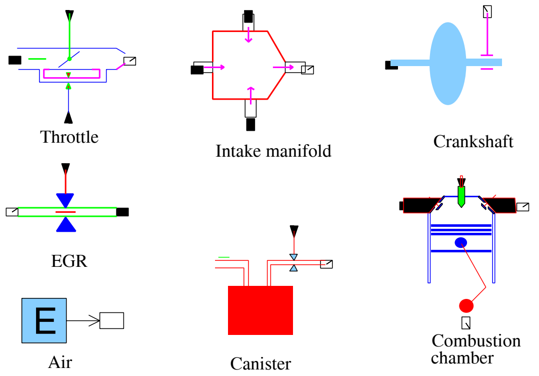


Figure 9: Scicos toolbox for engine components

3.1.1 Air intake throttle

The air throttle that controls the air flow rate into the air manifold and the combustion chamber can be modeled as a flow restriction. The model of a flow restriction highly depends on the pressure difference across the restriction, if small enough, the gas density is considered equal on both sides, *i.e.,* the gas is considered as an incompressible fluid. If, on the other hand,

large pressure differences can be expected the restriction should be modeled assuming compressible fluids. We have assumed that there is no back flow and the temperature is unchanged across the throttle [11, 12].
When the engine is in idle mode, the necessary air for the maintaining the minimum power of the engine is supplied through an air passage, called air bypass passage. The bypass area is controlled by the engine control unit (ECU).
The schematic of the throttle block in Scicos is given in Fig. 8. The air throttle component modeled with Modelica has two implicit ports and two explicit inputs. In Fig. 9, the square ports are implicit and triangle ones are explicit. Implicit ports represent inlet and outlet air flows and explicit input ports represent control signals. The implicit ports are modeled with the `connector` keyword in Modelica.

3.1.2 Exhaust gas recirculation (EGR)

In order to reduce harmful emissions resulting form the combustion, some of the exhaust gas is diverted back into the combustion process. In this method the inlet and exhaust manifolds are connected with a pipe and the recirculated gas flow rate is controlled by a valve [12]. The EGR control valve is modeled as a restriction [11, 12]. The schematic of the EGR block in Scicos is given in Fig. 9. The EGR block has two implicit ports and one explicit input port representing the control signal of the EGR valve.

3.1.3 Canister

Most of the hydrocarbon emissions in modern cars are from the exhaust, but a considerable part also comes from evaporative losses in the fuel tank. Most modern cars use an evaporative emissions management system to reduce these emissions. The basic function of this system is to trap and store the fuel vapors from the fuel tank in a canister until the engine is started. Then after the trapped fuel vapors is drawn into the engine by intake air manifold and combusted. In order to control the flow of vapors into the engine, a purge control valve with no back flow is used. The canister purge valve is modeled as a restriction [11, 12]. The schematic model of the Canister block in Scicos is given in Fig. 9.

3.1.4 Intake Manifold

The air flowing through the air throttle, the EGR, and the canister are mixed in the intake manifold and are send into the combustion chamber through the intake runner. We have assumed an isothermal manifold heat transfer, *i.e.,* constant manifold air temperature. The air in the intake manifold is composed of fresh air, fuel, and burnt gas. The concentrations can be described as functions of the partial pressures of fuel and air in the intake manifold. Using the ideal gas law, we can obtain the model of the intake manifold pressure. In SI engines, the inlet manifold pressure is reduced by the throttle in order to control the output torque. The flow rate in the intake runner is imposed by the pumping mechanism of the combustion chamber and the crankshaft rotation [11, 12].
The manifold air pressure sensor (MAP sensor) response is not as fast as the variation of pressure in the manifold, so its dynamics cannot be ignored and a first order filter is used to estimate the manifold pressure. The schematic of the manifold block in Scicos is given in Fig. 9. The block has four implicit ports and one explicit output port representing the MAP sensor output.

3.1.5 Combustion chamber

The combustion chamber is the heart of the engine. The air/fuel mixture flows into the cylinders and reacts and usable energy is extracted from the heated gas which is then expelled. In this work, the effects of the air/fuel ratio are not modeled. The cylinder is continuously swept by a piston which is connected to the crankshaft via a rod. The top of the cylinder houses intake and exhaust ports and a spark-plug in SI engines. The cylinder and the crankshaft have two important roles: torque generation and air pumping. When gas burns and expands, the piston is forced down. The downward movement is then transformed into rotational movement. The applied torque on the crankshaft depends on several parameters, such as the air/fuel mixture ratio, spark ignition time, manifold pressure, angular velocity of the crankshaft, etc. Since there is no accurate and simple physical model describing the generated torque, it is customary that a map is used. This map gives the optimum generated torque as a function of the manifold pressure (P_{man}), and the angular velocity of the crankshaft (ω). Thus, the optimal obtainable torque is defined as

$$\tau_{gen}^{opt} = F(\omega, P_{man}).$$

This map gives the value of the produced torque regardless of other important effects such as the effects of spark advance. Adjusting the spark advance timing, we can optimize engine efficiency to deliver peak combustion pressure when the piston reaches about 10° after top dead center angle. Incorrect spark timing can have a significant effect on emission output and vehicle drivability. The amount of the spark advance needed by the engine varies as function of the number of different operating conditions. The coolant temperature, fuel quality, and engine load are just a few of the many factors that can significantly impact ideal ignition time [13, 14, 15]. The effects of the spark timing on the produced torque is obtained by a using a experimentally obtained map. The map that we have used in our simulation gives the spark advance efficiency or the ratio of the produced torque with respect to the optimal torque, *i.e.,*

$$\eta = \frac{\tau_{gen}}{\tau_{gen}^{opt}} = H(|SA|)$$

where $|SA|$ is the absolute value of the spark advance timing. Note that $H(0) = 1$ and $|SA| < 40^\circ$.
The up/down movement of the cylinder creates a pumping effect; when the piston moves downward, the air is inhaled from the intake manifold and when the piston moves upward, the burnt air is exhaled to the exhaust manifold. In an internal combustion engine, the pressure on the intake side will normally be lower than on the exhaust side. Pumping gas from low to high pressure costs energy and this energy is taken from the crankshaft. The amount of the pumped air depends on several variables such as the cylinder volume, the angular velocity of the crankshaft, pressure in the intake manifold, pressure in the exhaust manifold, and the air temperature. Again, since there is no accurate and simple physical model describing the amount of the pumped air, a map is used to describe the total gas flow rate as a function of manifold pressure (P_{man}) and engine speed (ω). The maps used in our model have been obtained at IFP for a four cylinders SI engine. The schematic of the combustion chamber block in Scicos is given in Fig. 9. This block has three implicit ports for the air intake runner, the exhaust outlet, and the connection with the crankshaft. The block has one explicit input port representing the spark advance signal coming from the controller.

3.1.6 Crankshaft dynamics and perturbations

The crankshaft dynamics are modeled using the Newton's second law for rotating masses. All perturbations due to instabilities in combustion, differences in generated torque in cylinders, and variations in fuel injection in different cylinders are modeled with noise generator blocks (explicit Scicos blocks). This perturbations represent the load applied on the engine including controllable loads such as effects of A/C or antifrost systems on the engine and uncontrollable perturbations modeled with a zero mean random noise. The schematic of the crankshaft block in Scicos is given in Fig. 9. This block is connected to the combustion chamber block via an implicit port representing the mechanical connection of the crankshaft to the combustion chamber. The block has an explicit output port providing the angular velocity.

3.2 Simulation example: idle speed control

In this section, the engine components are assembled to construct the model of an SI engine, see Fig. 10. The engine model is then used to validate start-up and idle speed control strategies. The controller can be developed with standard (explicit) Scicos blocks. Its modeling with explicit blocks in Scicos has the advantage of using the rich control toolbox of Scilab.
In our model, the selected controller is relatively simple, *i.e.,* a PI controller. This controller will be active as soon as the engine speed exceeds 700 RPM. During the start-up phase, the spark advance is set to 20° and the throttle bypass area is 15%. When engine speed superseded the 700 RPM threshold, the control is handed over to the PI controller that adjusts the spark advance and the bypass area as a function of the reference speed, *i.e.,* 750 RPM, instantaneous MAP sensor and the engine speed. The simulation results of an engine start-up and the idle speed control is given in Fig .11. In this simulation, in order to test the the idle speed controller, different loads (τ_l) are applied at instants t=20 sec and t=40 sec, see the bottom plot of Fig. 11. In the top plot of Fig. 11, the engine speed is shown. The engine speed is relatively regulated around 750 RPM in spite of the loads and random perturbations. The middle plot of Fig. 11 gives the intake manifold pressure that decreases from atmosphere pressure as engine starts up and varies as load changes.

4 Future Works

The Modelica compiler used in Scicos has been developed in the SIMPA[3] project with the participation of INRIA, IMAGINE, EDF, IFP, and Cril Technology. Recently, the ANR[4]/RNTL SIMPA2 project has been

[3]Simulation pour le Procédé et l'Automatique
[4]French National Research Agency

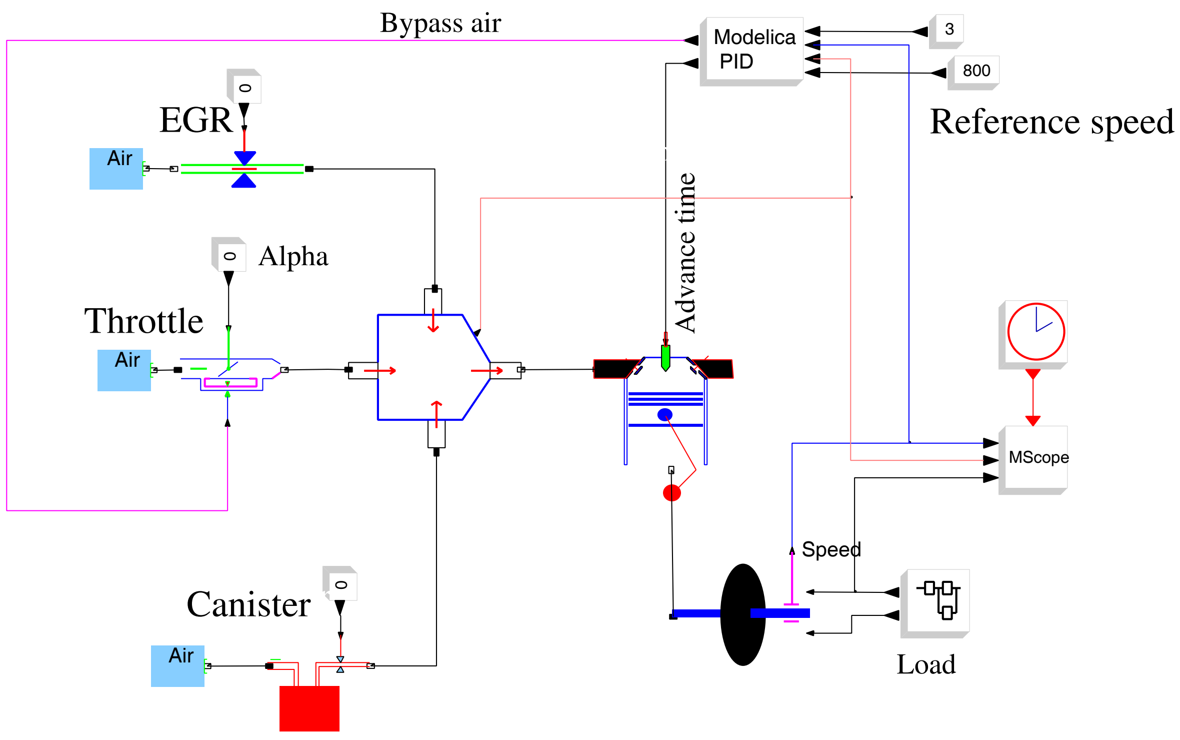


Figure 10: The Scicos model for a mean SI engine

launched to develop a more complete Modelica compiler. The main objectives of this project are to extend the SIMPA compiler to fully support inheritance and hybrid systems, give the possibility to solve inverse problems by model inversion for static and dynamic systems, and enhance initialization of Modelica models.

5 Conclusion

In this paper, we modeled a drilling station and a mean value SI engine with Modelica in Scicos. It should be noted that these models have been already modeled and simulated in Simulink at IFP. The modeling in Modelica was performed in order to compare two modeling environments. Modeling in Modelica has several advantages: Modelica is a declarative language with which very general hybrid systems can be modeled. The Modelica models are independent of the simulation tool and can be simulated in any Modelica simulator. Another important advantage of using Modelica lies in the symbolic manipulation of models. Because it gives the possibility of several simplifications such as efficient discontinuity handling, index reduction, and generation of the analytical Jacobian. Another advantage of Modelica models comparing to Simulink models is the facility in model construction and navigation in the model. For example, the model of the drilling station in Simulink is composed of more that 500 blocks distributed in 116 subsystems whereas the Scicos model is just composed of 9 blocks. The model of the SI engine in Simulink is composed of 203 blocks distributed in 30 subsystems whereas the Scicos model is composed of 20 blocks. The reduced number of blocks helps the user to construct and debug the model easier and faster.

6 Acknowledgements

The authors gratefully acknowledge the helps and comments of Isabelle Rey-Fabret[5] to understand the model of drilling station and Gilles Corde1 for providing the engine model.

[5]Research professor at IFP and École des Pétroles et Moteurs

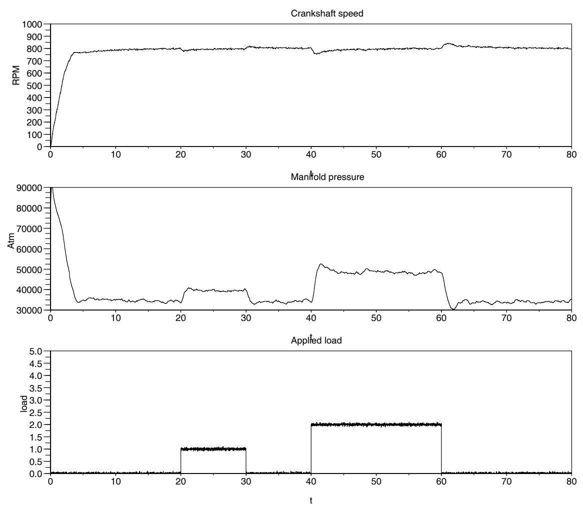


Figure 11: simulation result for start-up and idle speed control of the engine

References

[1] Campbell S. L., Chancelier J.P., Nikoukhah R., Modeling and simulation in Scilab/Scicos, Springer Verlag publishing, 2005.

[2] Chancelier J. P., Delebecque C. , Gomez C., Goursat M., Nikoukhah R., Steer S., An introduction to Scilab, Springer Verlag, Le Chesnay, France, 2002.

[3] Najafi M.,The Numerical Solver for the Simulation of the Hybrid Dynamical Systems, Doctor of Science Thesis, Paris XII University, 2005

[4] Fritzson P., Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Wiley-IEEE Press, 2004.

[5] Pavone D., and Desplans J.P., Analyse et Modélisation du comportement dynamique d'un rig de forage, *IFP report No 42208*, 1996.

[6] Pavone D., and Desplans J.P., Analyse, Modélisation et élimination du stick-slip observé sur les expériences de Norvège, IFP report No 41317, 1994.

[7] Rey-Fabret I., and Mabile C., and Oudin N., Detecting whirling behavior of the drill string from surface measurments, Proc. SPE 72nd, annual technical conference and exhibitions of the Society of Petroleum Engineer, San Antonio, USA, pp 223-232, 1997.

[8] Dufeyte M.P., and Henneuse H., Detection and monitoring of the stick-slip motion: field experiments, Proc. SPE/IADC Drilling conference, Amsterdam, Netherlands, 1991.

[9] Dawson R., and Lin Y. Q., and Spanos P.D., Drill string oscillation, Proc. Spring. Conf. of Socienty for experimental Mech, Houston, June, 1987.

[10] Najafi M., Modeling complex systems with Modelica in Scicos: Application to mean value spark engine, ESM'2007, Westin Dragonara Hotel, St. Julian's, Malta, 2007.

[11] Nakayama Y. and Boucher F., Introduction to Fluid Mechanics, Arnold, London, Great Britain, 1999.

[12] Silverlind D., Mean Value Engine Modeling with Modelica. M, Department of Electrical Engineering, Linkoping University, Master's thesis, 2001.

[13] Heywood J.B., Internal Combustion Engine Fundamentals. McGraw-Hill series in mechanical engineering, international edition, 1988.

[14] Aquino C., Transient A/F control characteristics of the 5 liter central fuel injection engine. In Society of Automotive Engineers technical paper, 1981.

[15] Degobert P., Automobiles and Pollution. In Society of Automotive Engineers, Inc. Translation of Automobile et pollution, 1995.

Controller Development for an Automotive Ac-system using R744 as Refrigerant

Sanaz Karim Hubertus Tummescheit
Modelon AB
Ideon Science Park, SE-22370 Lund, Sweden
Hubertus.Tummescheit@modelon.se, Sanaz.Karim@gmail.com

Abstract

Due to recent regulatory changes in Europe, CO_2 or R744 is considered a serious alternative to be the successor of R134a for the AC-system of cars for the European market. Research into R744 as a working fluid for automotive AC started in the early nineties and continues even today. There are still open issues in both design and control of R744 systems, e.g. the choice of an expansion device that satisfies both cost and performance constraints, control in the subcritical region and controling transcritical transients. In a Masters thesis project organized in cooperation with Daimler AG in Sindelfingen, these issues were investigated using a well validated model of an R744 prototype system modeled using the AirConditioning Library by Modelon AB and Dymola from Dynasim AB. The preferred choice for the expansion device from a cost point of view is a two-stage orifice with a pressure-activated bypass for high load conditions. The solution with the two-stage valve is compared to a reference system that uses an electronically controlled valve that is controlled to the COP-optimal high side pressure. Unfortunately, the two-stage valve can exhibit both limit cycling behaviour and multiple steady states depending on the plant operation history, both undesirable properties. For the investigated system the drawbacks could be eliminated by proper control design. Another problem that was investigated was the load distribution between a front- and a back seat evaporator for a two-evaporator version of the same system. Again for cost reasons, the refrigerant side of the second evaporator is not controlled, instead flow is split between the two evaporators using a fixed expansion device for the rear evaporator.

Keywords: air conditioning; compression cycle; simulation; CO2; R744; control design, COP optimization

1 Introduction

Under the Kyoto protocol agreement, by the year 2012, industrialized countries have to reduce their collective emissions of greenhouse gas 5% below their 1990 levels. Since the current refrigerant used in vehicles, R134a, has a GWP (Global Warming Potential) of 1410, R744 (CO_2) technology has been proposed as a natural alternative to current R134a-based systems. The main benefits of R744 as a refrigerant are:

- Energy-efficient
- Non-toxic
- Non-flammable
- No ozone depletion potential (ODP=0)
- Low global warming potential (GWP=1)

Apart from the environmental benefits listed above, using R744 as a refrigerant for air-conditioning (A/C) systems can decrease the fuel consumption under some climate conditions.
Daimler AG and some of its suppliers have developed and validated specific component and system models for R744-cycles based on the AirConditioning Library by Modelon. These models were used to investigate control strategies for both the single evaporator and the dual evaporator system prototype for an S-class Mercedes.

2 A/C Systems Optimization and Control

The role of the HVAC-unit in the A/C system is to provide maximum cooling power in order to cool down the air and dehumidify it before re-heating and ventilation. To increase cooling power at very high ambient temperature, traditionally a lower COP (more fuel consumption) is accepted. The current practice is to control the air temperature after the evaporator to a constant, low temperature (slightly above 0 Celsius to avoid frost) and control the actual cabin temperature by mixing in warm outside air in the HVAC box to obtain the desired temperature in the cabin. However, most of the operating times the optimization of the COP is the more reasonable control target from the point of view that fuel consumption should be minimized. These two control targets can be fulfilled by inserting two decoupled SISO control loops, one of them controling the high pressure and the other one controling the evaporator outlet temperature by considering the strong crosscoupling between these two variables.

2.1 Optimum High-Pressure Control

To achieve maximum COP in R744 systems, a simple SISO control strategy with two control loops has been proposed by [1]. They consider the high- pressure as the main variable that affects the COP and cooling power. Since the heat rejection process of the R744 refrigeration cycle takes place in the supercritical region, where the pressure is independent of the temperature, the system efficiency is a nonlinear function of the working pressure and the ambient temperature.
For each ambient (gas cooler air inlet) temperature, there is an optimum high-pressure, which results in the maximum COP. With the increase of the ambient temperature, the optimum pressure increases.
The other boundary conditions (evaporator temperature, air humidity and flow rate) have negligible effect on the optimum high-pressure.

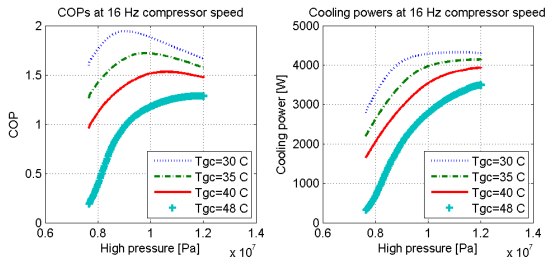


Figure 1: Comparison of COP and cooling power with the change of high-pressure

A variable swash plate controller is used as low-pressure (evaporator air outlet temperature) controller and since any change in a angle of the swash plate will affect the pressure ratio as well as the compressor power; it is expected that it changes the optimum high-pressure as well. Therefore a high-pressure regulator which controls the refrigerant flow, based on the ambient temperature and compressor speed is suggested by [1]. For the purpose of simplification the effect of speed is neglected and the controller is reduced to a controller which works just based on the ambient or gas cooler temperature and is designed at a low speed, since at higher speeds of the compressor the role of the optimum high-pressure is less significant.
An electronic expansion valve like a PWM-valve can be used as an actuator to change the flow rate to achieve desired the high-pressure, but to get rid of the costs of the high-pressure controller and gas cooler temperature measurement device, a two-stage orifice expansion valve has been developed whose internal control mechanism is described in section 3.1.

2.2 Evaporator Temperature Control

Under low load conditions, it is necessary to control the compressor power to reduce the cooling power to the desired range for the A/C system and not let it reach its maximum possible capacity. These conditions are low cooling load and/or high engine speed. Since the compressor of the automotive A/C unit draws its driving force from the engine, its power is a function of the engine speed, which is a highly fluctuating variable. Control of the compressor capacity is necessary to compensate engine speed disturbances, to satisfy the comfort requirements and to avoid temperature variations. Control is particularly important at higher speeds, which cause an undesirable power of the compressor and too low temperature at the evaporator.
Among the various methods proposed to control the compressor capacity, using a variable displacement compressor is the most attractive one. The most popular variable displacement compressor for automotive use today is the swash-plate compessor[1]. Changing the inclination of the swash plate changes the displacement of each of the many pistons of the compressor. This causes a change of the pressure ratio, both high-pressure as well as low-pressure are affected, but the effect of the expansion valve on the high pressure is dominant. The control of the swash plate angle and thus the relative volume is used as a low pressure controller in spite of its influence on the high pressure.
In the sub-critical region, where the heat rejection takes place isothermally, evaporator refrigerant and air outlet temperature are functions of the low-pressure,

[1]For hybrid cars with sufficient electrical power, other options would be advantageous, because they open up the new possibility of using a speed control of the compressor.

thus the swash plate control makes it possible to control the evaporator temperature and via the temperature also the power.
Concerning the previous section, at a constant speed, it is acceptable to neglect the cross coupling between the first SISO loop which tries to maximize COP by high-pressure control and the second one which aims to control the low-pressure (evaporator air outlet temperature), but it is not satisfactory to decouple these loops in the case of speed changes.
Assuming constant speed, control of the evaporator air outlet temperature in the case of low cooling load can improve the COP significantly due to a smaller pressure ratio and consequently smaller power uptake of the compressor.

3 AirConditioning Library

The AirConditioning Library and the simulation tool Dymola, both based on the standardized, freely available modelling language Modelica, have been selected by the German automotive OEM as the preferred tool for model development and exchange for the A/C system in passenger cars. The library contains a complete range of component models and templates of typically used and proposed A/C system architectures and all currently used as well as new and proposed refrigerants for automotive applications. The modeling detail is appropriate for component selection, system architecture design, system integration for overall vehicle thermal management and climate control design. Prototype systems for future technologies often contain components that differ from those needed for conventional designs, but due to the open code and the given modeling infrastructure, it is straightforward to add unusual components to the Library. In this case a two-stage orifice model with a pressure operated bypass had to be added.

3.1 Two-Stage Orifice Model

This valve has an internal mechanism to drastically change its Kv value based on the pressure difference between the low- and high-pressure side [6]. It consists of a standard orifice and a bypass which is closed for small pressure differences. As shown in Figure 2, the refrigerant flows only through the orifice at pressure differences below a pressure difference Δp, in this case set to 73 bar. The bypass starts to open at a rising pressure difference of Δp with a very steep gradient, and for higher pressures, the Kv-value rises almost linearly with the pressure difference. This results in a very non-linear pressure – mass flow characteristic which is prone to limit-cycling behaviour. The cycle is caused by interaction between the dynamics if the mass storage at the high- and low pressure levels in combination with the differences between the mass flow characteristics of the compressor (almost no change for pressure difference above and below Δp) and the valve (almost a step function at Δp). When the rising pressure opens the valve for a pressure difference higher than Δp, the opening bypass will increase the mass flow from the high pressure side so rapidly that the pressure difference falls below the bypass opening limit, because the compressor mass flow does not increase in the same degree and the cycle starts again.

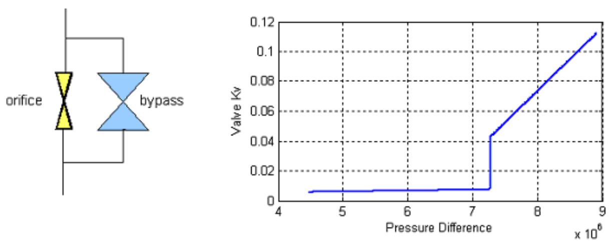


Figure 2: Two stage orifice valve

The highly nonlinear behaviour of the valve's Kv-value can under some situations give rise to limit cycling around the steep part of the characteristic where the valve opens, and it may even lead to two steady states with different COP's, one at a pressure difference above the opening pressure, the other one at a pressure difference below the opening pressure, at identical boundary conditions.

4 Single Evaporator, Two-stage Orifice Valve System Control Design

While no direct control of the high-pressure is possible anymore when using the two-stage orifice valve, it is still desired to keep the COP as close as possible to its optimal value in order to reduce fuel consumption. As previously mentioned, the first control target remains to regulate the evaporator temperature by means of the compressor relative volume control, the COP control is of secondary importance. To achieve these goals, a simplified control structure proposed by [4] was used as a starting point for the control design. That structure was developed for the same type of two-stage orifice valve and used a complex feed-forward map with three

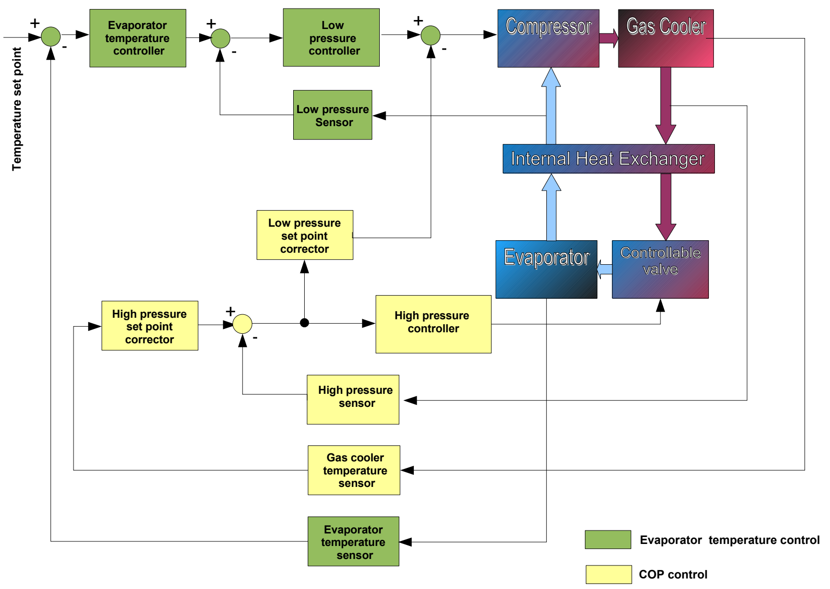


Figure 3: Control structure for R744 AC-cycle with electronically controllable expansion device, assuming COP-optimal control via the valve to control the high pressure side and temperature/power control via the compressor to control the low pressure side.

inputs (engine speed, air mass flow and inlet air temperature) to mimic the optimal high pressure control with a fully controllable valve. There are a number of reasons why the control structure suggested in [4] uses a high-pressure controller in place of the low-pressure one for controlling the evaporator temperature. The refrigerant high-pressure sensor required for controlling the high-pressure is already present for monitoring and protection functions in todays R134a circuits, so no additional sensors are needed and this suggestion gets rid of the cost for a low-pressure sensor. There are a number of reasons why the control structure proposed by [4] was dropped in favour of a simpler one. On the low pressure side the existing evaporator outlet temperature sensor can be used due to the simple temperature-pressure relationship of the saturation curve:

- The feed forward is not robust to changes in the environment conditions, in particular not to changes in humidity, which today is not measured due to too costly sensors. The feed forward only works well in a limited range of operating conditions and actually decreases control performance in other situations. A feed forward based design that includes humidity measurements would most likely avoid the robsutness drawback.

- Using the components chosen in the given prototype R744 system with the two-stage orifice valve, undesirable limit cycling behaviour occurs at some operating points. It is not possible to remove the limit-cycling behaviour with the given control structure.

- For engine speed disturbances, the feed forward scheme for controling evaporator outlet temperature from with a feedback on the high side pressure did not work reliably.

- The occurence of multiple steady states, see section 4.3.

The current investigation was not done with a fully realistic sensor model for the evaporator temperature. If a cost-effective temperature sensor would be too slow to control engine speed variations, a low pressure controller would still be preferrable to the high pressure one with feedforward due to the list of drawbacks above.

4.1 Performance of the Valves

To compare the operation of the controllable valve in an optimized cycle and a two-stage valve without control of the high pressure, all boundary conditions and the compressor speed are kept constant and simulation were performed for three different load cases and both valves.

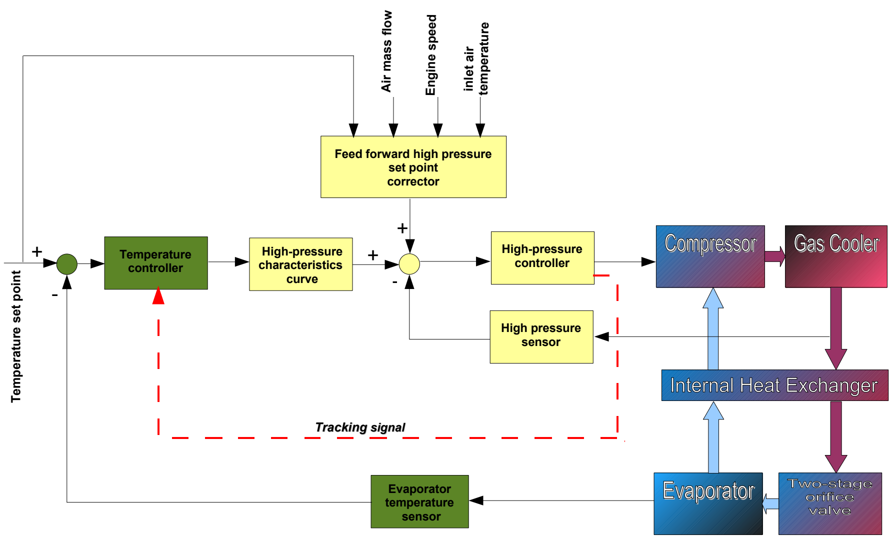


Figure 4: Proposed Control Structure by [4], simplified compared to the control structure in 3. In this case it is also assumed that the temperature set-point for the evaporator is adapted at low load to improve the COP.

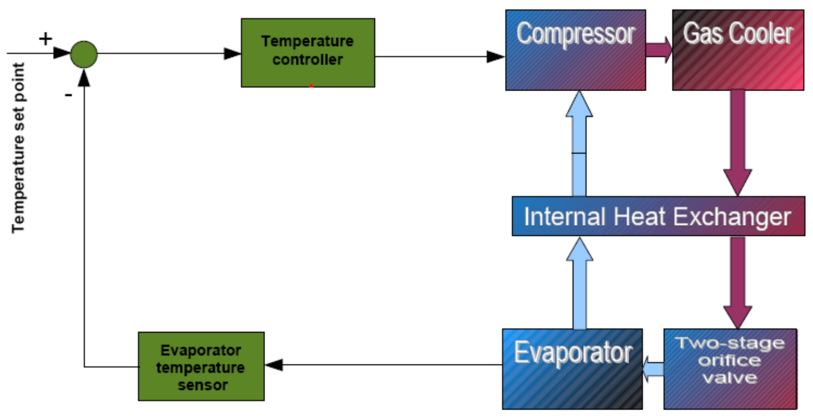


Figure 5: Alternative control structure for control of the low pressure side. The evaporator temperature set point is used to improve COP, which means that a higher complexity is needed in the supervisory part of the HVAC control that needs to determine the proper temperature set point.

1. Low cooling load and no control on evaporator outlet air temperature (Fixed relative volume of the compressor)
2. High cooling load and no control on evaporator outlet air temperature (Fixed relative volume of the compressor)
3. Evaporator temperature controlled (low cooling load)

4.1.1 Case 1

The system with two-stage valve has lower COP and higher cooling power than the optimized cycle, even for the lower ambient temperature. At higher temperature losses decrease, see Figure 6.

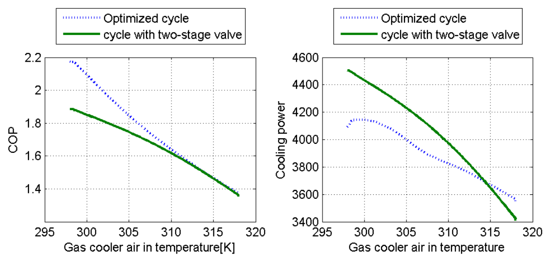


Figure 6: Comparison of the two valves, case 1

The high-pressure with two-stage orifice valve is kept fixed around 110 bars, while the variable Kv valve allows the pressure to change in a wider range. The reasons is behind the internal mechanism of the two-stage orifice valve which does not result in a Kv-value close to the controlled Kv for most of this range (Figure 7).

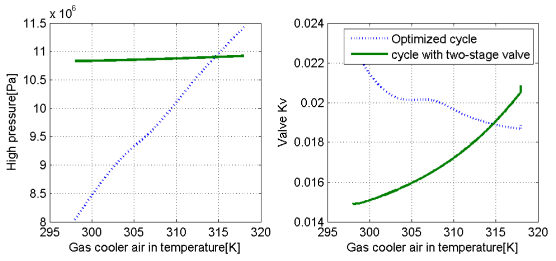


Figure 7: Comparison of the two valves, case 1

4.1.2 Case 2

In comparison with the previous case, at higher loads, the cycle with two-stage orifice valve has a COP near to the optimum value but at higher ambient temperatures it does not achieve equally high cooling power as the optimized cycle.

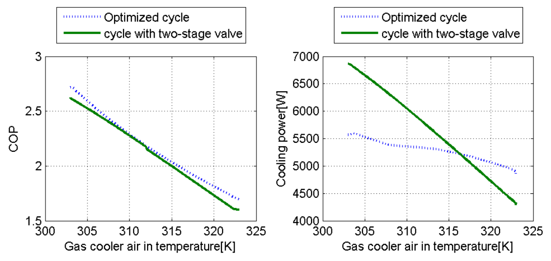


Figure 8: Comparison of the two valves, case 2

For this cooling load, the Kv shows a smaller deviation from the optimized one.

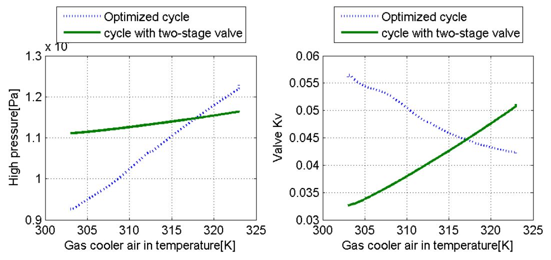


Figure 9: Comparison of the two valves, case 2

4.1.3 Case 3

When the low-pressure is controlled via the relative displacement of the compressor, the COP is improved for both cycles.

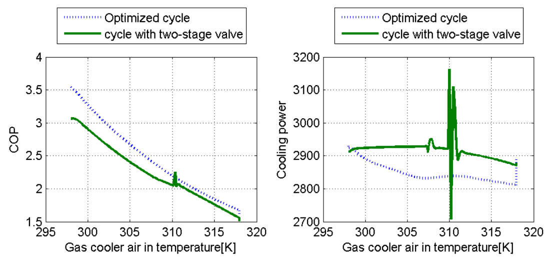


Figure 10: Comparison of the two valves, case 3

Since the pressure difference is low, at the lower ambient temperatures, the refrigerant passes through the fixed orifice of the two-stage orifice valve and provides the high-pressure that is needed for better COP. Both the valve-Kv values and correspondingly the resulting high pressures are closer to one another for this load case and control scheme than for the previous two ones.

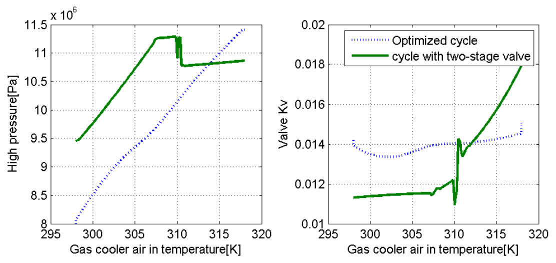


Figure 11: Comparison of the two valves, case 3

As has been demonstrated in this section, for lower ambient temperatures, the COP of the cycle with two-stage orifice is up to 40% less than ideal cycle, therefore it is suggested that in this range of ambient temperatures, the evaporator temperature is controlled to the highest possible value to improve the COP. Assuming the evaporator temperature is controlled with the two-stage valve cycle, the differences between the solution are not as dramatic as a first look suggests. The worst case scenario is, however, handled better with the optimized cycle that provides the highest cooling power at the highest load case.

4.2 Limit-Cycling Behaviour

In some operating points, which result in a higher pressure-difference than 73 bars, as a consequence of the rising pressure, the bypass starts to open and decreases the high-pressure, the decrease in high-pressure causes the closing of the bypass and this limit cycle continues until one of the inputs alters the pressure difference and mass flow rate. To observe the role of flow rate and pressure change in the limit cycle phenomenon directly, all the boundary conditions are kept constant and the relative volume of the compressor is changed manually to provide the appropriate pressure difference and flow rate. Figure 12 illustrates above explanations.

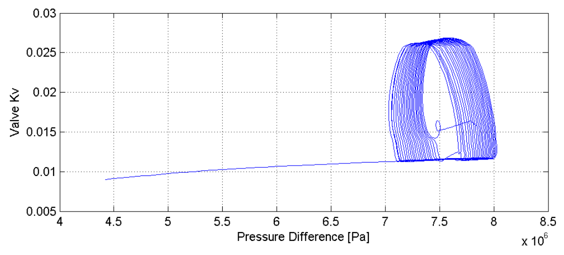


Figure 12: Limit cycle

Other output parameters, which are correlated with the high-pressure, will also show this limit cycle. The effect on the evaporator outlet air temperature is negligible (less than 1^{o}C in this case) and it is seen in Figure 13 that the low-pressure controller can remove the fluctuations. Therefore passengers do not sense the oscillations of the temperature.

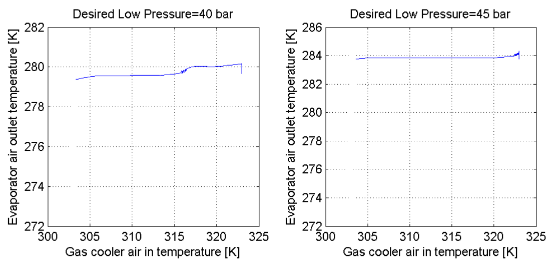


Figure 13: Temperature and limit cycle

But the effect on the cooling capacity and COP is quite considerable. In the temperature interval where this phenomenon happens, the highest deviation of the COP is about 50% less than the expected average value (Figure 14).

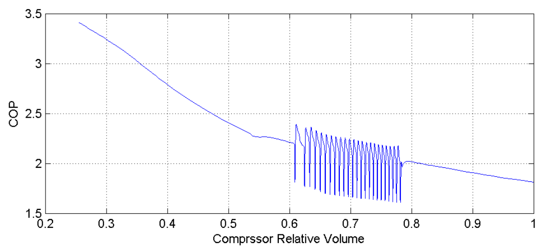


Figure 14: COP and Limit cycle

However, this limit cycle does only occur at few operating points and its characteristic differs in different circumstances. The following observations demonstrate this statement: Assuming a low-pressure controlled cycle, the ambient temperature varies in the range from 30^{o}C to 45^{o}C, and other operating conditions are kept constant. Figure 15 shows the phase portrait plot of two different cases when the limit cycle takes place. One of them happens when the desired low-pressure is 40 bar and the other one at 45 bar.

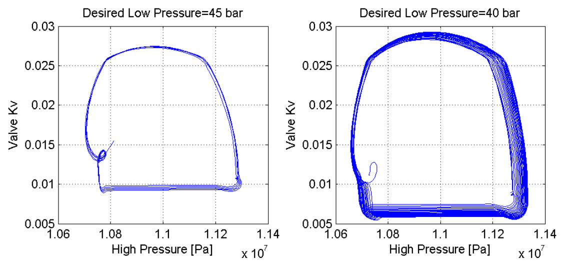


Figure 15: Portrait plot of the valve Kv against the pressure-difference

Under normal driving conditions, boundary conditions will rarely ever be constant for a sufficiently long time such that these limit cycling conditions will be noticeable, but they are nonetheless an undesired side effect of the valve construction.

4.3 Multiple Steady-States

In the case of high-pressure control and in the vicinity of 73 bar pressure-difference, when the two-stage orifice valve changes its flow configuration, a *bistability* phenomenon takes place. In this case, any disturbances which leads to small variance in the pressure-difference, causes the valve to jump to the alternate path while the high-pressure is kept constant by the controller. Therefore the system is able to exist in either of two steady states, while the high-pressure is fixed. Figure 16 shows that a small disturbances of the pressure, pushes the system to another steady state and causes a significant change in the cooling power. Although this will be compensated by the outer loop later on, it is another situation where the high-pressure loop in combination with the two-stage valve acts against the main purpose of control.

5 Dual Evaporators

Today luxury cars allow passengers to control a different climate in up to four climate zones. This requires the presence of two or even three evaporators to generate the cooling capacity for front and rear passengers. The Electronic Control Unit (ECU) controls the position of the different temperature blend doors

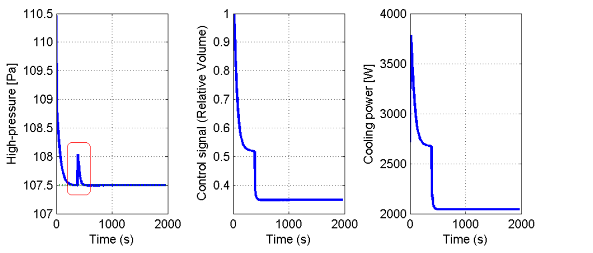


Figure 16: Bi-stable behaviour for high pressure control

to provide the passengers with their desired temperature in different zones. In the cooler unit, the high-pressure refrigerant splits and flows from two different expansion devices to the front and rear evaporator. The cooling capacity is divided accordingly between both evaporators. But the amount of the division depends on the operating conditions and structure of the valves. If a variable displacement compressor is used to control the front evaporator outlet air temperature, and a two-stage orifice valve to improve the COP, then a fixed orifice can be used to pass the refrigerant to the rear evaporator. In this case, there is no direct control on the outlet air temperature of the rear evaporator. To have full control authority on both evaporator temperatures, a controllable second expansion device would be needed. Alternatively, a model-based controller could be designed to control the compressor relative volume based on the measured value of the outlet air temperature of both evaporators. The easier way to control the cooling capacity of the rear evaporator is to use a variable speed fan and change the air flow around the evaporator, while the temperature of the front evaporator is controlled with the compressor relative volume variation using the same SISO approach as for the one-evaporator system. This will change the balance point of the rear evaporator low-pressure and this in turn changes the front evaporator low-pressure. The behaviour of the latter control system is investigated in [2], where in the modeling of the dual evaporator system, it is supposed that the front evaporator uses fresh air for ventilation and the rear compartment has just one zone. The outlet air of the front evaporator enters the car cabin, it is mixed with recirculation air of the rear compartment and then enters the rear evaporator for the second phase of cooling.

5.1 Cooling Power Distribution

To compare the cooling power of the one-evaporator system with the two-evaporator one, both systems are simulated under the same cooling load and at the same operating conditions. Note that all other components are the same, which means in particular that the heat rejection capacity via the gascooler is identical for both systems. Figure 17 illustrates that the summation of the capacity of the front and rear evaporator is equal to the capacity of one-evaporator system in this condition. It also shows that the outlet air temperature of the front evaporator is same for both cases. With a perfect model which includes the corresponding effects of the rear compartment on the front one, this distribution scheme may change a little and more compressor work will be needed to keep the front evaporator temperature constant.

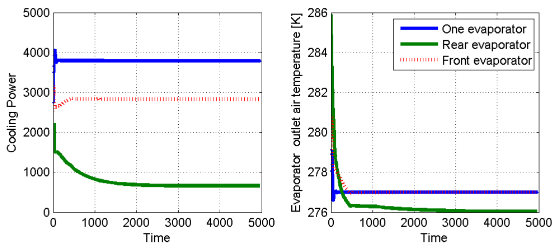


Figure 17: Cooling power distribution between two evaporators in comparison with one-evaporator system

Figure 18 shows the cooling power distribution against the ambient temperature. At higher temperatures, the pattern of distribution will change but acceptable cooling power is still provided for both evaporators.

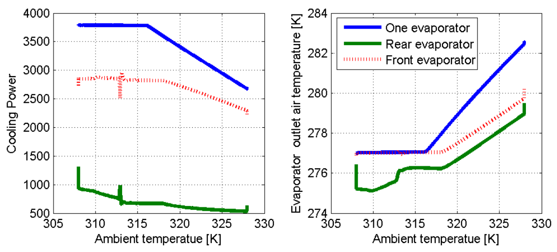


Figure 18: Cooling power distribution between two evaporators in comparison with one-evaporator system

5.2 Rear AirFlow Effect

In order to order to manipulate the cooling power of the rear evaporator, it is possible to change the air mass flow through it. The simulation was run in a limited range of airflow variations under two different cooling loads. Figure 19 shows the change of the rear evaporator cooling power when the air mass flow is changed at 5000 second. Figure 20 shows the cooling power variation against the air mass flow variation under a high and a low cooling load.

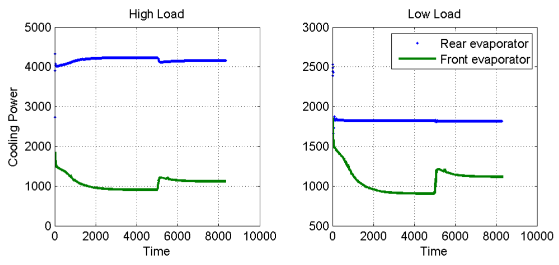


Figure 19: Rear evaporator air flow change.

It is seen that the rear cooling power is changed while the front one is almost kept constant. At lower cooling loads, the rear evaporator capacity is more sensitive to the air mass flow change.

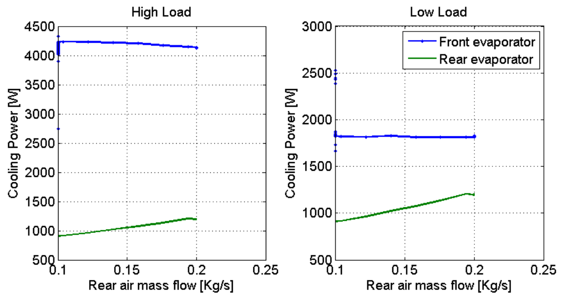


Figure 20: Cooling power over air mass flow.

Therefore, at these conditions, using a two-stage valve besides the front evaporator temperature control is possible, while the capacity of the rear evaporator is controlled by means of adjusting the air mass flow.

6 Conclusions

Various aspects of system and control design for a prototype of a R744 automotive A/C system for the Mercedes S-class were investigated by simulation using the AirConditioning Library and Dymola. Different system designs with a controllable expansion valve and a two-stage bypass orifice were compared and show that the controllable valve gives up to 15 % better COP than the two-stage valve. Several control designs were compared and the result was that the simplest control structure proved to be most robust and had better performance than the more complex versions. Furthermore it is demonstrated that the system with the highly non-linear two-stage valve exhibits limit-cycling behaviour and bistability around the part of the valve characteristic that looks almost like a step function in the valve coefficient Kv. For the two-evaporator system which uses a two-stage orifice valve to regulate the pressure of the front evaporator, simulation results suggest that the same approach of control for the one-evaporator system is also applicable for the dual evaporator system. With the given limited control authority, pressure and temperature of the rear evaporator will always be defined by the controlled conditions for the front evaporator and the boundary conditions. Instead of temperature control for the rear compartment, the capacity of the rear evaporator can be controlled using a variable speed fan, but only within certain limits.

References

[1] Yang W., Fartaj A., Ting S-K., Co2 Automative A/C System Optimum High Pressure Control,SAE International 2005-01-2022, 2005.

[2] Karim, S. Open Issues in Control of Automotive R744 Air-Conditioning Systems: Masters Thesis Nr E3492E, Department of Electrical Engineering, Dalarna University, 2007.

[3] Tummescheit H. Design and Implementation of Object-Oriented Model Libraries using Modelica. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.

[4] Lochmahr K., Baruschke W. and Britsch-Laudwein A., Control System for R744 Refrigerant Circuits, ATZ worldwide, 2005.

[5] Åström K. and Hägglund T., Advanced PID Control, ISA-The Instrumentation, Systems, and Automation Society, 2006

[6] Lemke N., Tegethoff W., Köhler J., and Horstmann, P., Expansion Devices for R744 MAC Units, Vehicle Thermal Managment Systems 7 Conference and Exhibition, SAE International, 2005.

Implementation of a *Modelica* Online Optimization for an Operating Strategy of a Hybrid Powertrain

Henrik Wigermo, BMW Group, Energymanagement 88077 München
Johannes von Grundherr, BMW Group,Energymanagement 88077 München
Thomas Christ BMW Hybrid Cooperation, Vehicle Architecture,
1960 Technology Dr., Troy, Michigan USA

January 21, 2008

Abstract

The paper presents a method of implementing an optimization based control algorithm within the Modelica framework. To find the optimal point within a given objective function the golden section search is employed. Its implementation in Modelica is presented. The optimizer based control strategy is applied to control a simplified electrical circuit and to a hybrid electric vehicle.

Keywords: Modelica; Optimization; Hybrid Vehicle; Simulation; Fuel Consumption

1 Introduction

Online optimization is increasingly being implemented for better results in controlling complex systems. It is especially helpful if the control objective depends on several input parameters which influence the outcome in a non intuitive way. One example is the operational strategy of a powersplit hybrid electric vehicle.
Compared to conventional transmissions, hybrid transmissions allow for several additional degrees of freedom: The combustion engine speed can be controlled independently from vehicle speed and battery power can be used for propulsion or the storage of braking energy. Although the main control objective is the fuel economy of the vehicle, other goals like dynamic response, driveability, acoustic impression and tailpipe emissions have to be achieved. In many cases the definition of the control objective is given by a calibration table or multidimensional mappings. Since a mapping normally cannot be expressed analytically, the solution to the optimization problem has to be computed online for each control step.
In the development process of hybrid vehicles, simulation is a key issue. It is used to study aspects like fuel consumption and performance and to understand complex system interactions. Since the hybrid vehicle powertrain is composed of mechanical, electrical, chemical and thermodynamical components, *Modelica* is a very useful tool for this. The control software of the hybrid vehicle is normally implemented using tools like Simulink or ASCET. The actual powertrain control is only a small part of the entire controls software. A great deal of code which is interconnected to the actual powertrain control concerns system diagnosis or remedial actions, and does not need to be simulated. To study the powertrain behavior only the relevant parts of the control code are transferred to *Modelica*.
In this paper, we shall present a simple optimization algorithm and give an example on how it can be implemented in *Modelica*. We will also take a look on a possible employment of such an algorithm; the powertrain control of a hybrid electric vehicle. In addition, the following points have been investigated: How will an algorithm that requires fixed time-steps work together with an complex vehicle model? How does the optimization influence the simulation time? How can standard *Modelica* elements like tables be integrated in the optimization algorithm, since it doesn't allow graphical programming?

2 Problem statement

A Plant P is controlled by its input u and disturbed by d. y is the observed measurement. In an early control development stage the plant can be represented by a simulation model. The control task is to follow a given reference y_{ref} so that an objective function $J(y, y_{ref})$ is minimized. For linear systems and quadratic objective

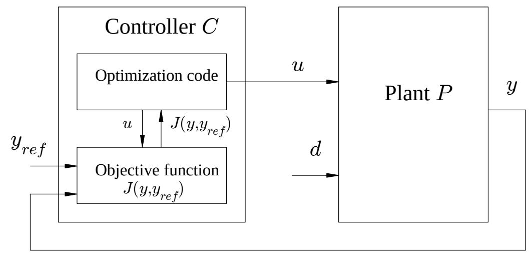


Figure 1: Control optimization problem

functions the choice of controller is well understood. A linear state feedback control can be directly derived from the linear plant given by the system matrices A, B, C, D and by the coefficients of the quadratic objective function.

For nonlinear objective functions the optimization can be carried out by an optimization algorithm. In each optimization step the algorithm calls the objective function, iterating the control signal u to generate the optimal solution u^*.

In our case the plant is a *Modelica* model. The control using the optimization algorithm is also integrated in *Modelica*. A tutorial example of such an optimization is shown in section 3.2.

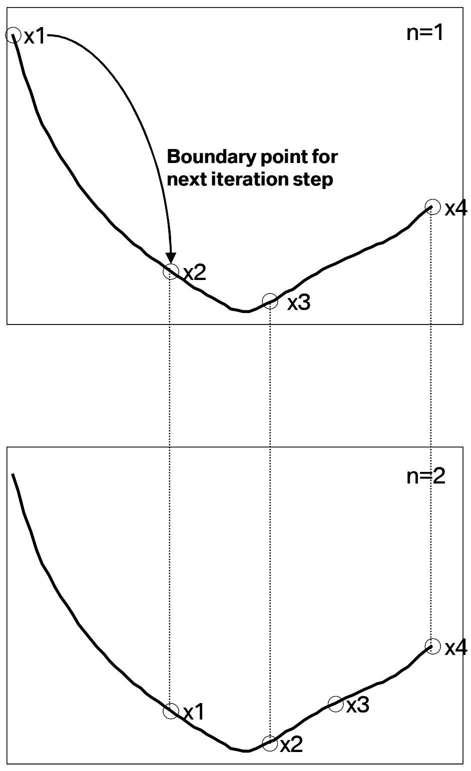


Figure 2: Principle of Golden Section Search Algorithm

3 Online optimization

An optimization algorithm used for the given problem has to be robust, i.e. it needs to come up with a solution after a finite number of iterations. Such an algorithm is golden section search. In this paper its integration into the *Modelica* framework is shown.

3.1 Optimization algorithm - Golden section search

The golden section search derives its name from the fact that it narrows its search interval with the golden ratio $\frac{1}{2}(1+\sqrt{(5)})$ in each step. The technique is effective only for unimodal functions, where a maximum or minimum is known to exist within a given interval. As starting points the lower and upper limit of the search interval are chosen. Using the golden section, two new points within the interval are evaluated and compared. The point with the highest functional value is chosen as a new boundary point, and points outside of this are no longer considered. The algorithm continues to search until the maximum number of iterations is reached or the termination condition suggested in [4]: $\mid x4 - x1 \mid > \tau(\mid x2 \mid + \mid x3 \mid)$ is satisfied. τ is a tolerance parameter. *Modelica* code 1 describes the golden section search algorithm:

```
function goldenSectionSearch
  extends Modelica.Icons.Function;
  parameter Real tau=0.001;
  ...
  constant Real C=0.5*(3 - sqrt(5));
  constant Real R=1-C;
  ...
algorithm
    x1:= xLowerLimit;
    x4:= xUpperLimit;
    x2:= R*x1 + C*x4;
    x3:= C*x1 + R*x4;
    fx2:=optFunction(x2,alpha,IbatDes,
    Ri,Iload,gammaI);
    fx3 :=optFunction(x3,alpha,IbatDes,
    Ri,Iload,gammaI);

while abs(x4-x1)>
    tau*(abs(x2)+abs(x3)) loop
  if (fx3<fx2) then
        x1:=x2;
        x2:=x3;
        x3:=R*x3 + C*x4;
        fx2:=fx3;
        fx3:=optFunction(x3,alpha,
          IbatDes,Ri,Iload,gammaI);
  else
        x4:=x3;
        x3:=x2;
        x2:=R*x2 + C*x1;
        fx3:=fx2;
        fx2:=optFunction(x2,alpha,
          IbatDes,Ri,Iload,gammaI);
    end if;
end while; if
  (fx2<fx3) then
    xmin:=x2;
    fxmin:=fx2;
else
    xmin:=x3;
    fxmin:=fx3;
end if;

end goldenSectionSearch;
```

Modelica Code 1: Golden Section Search Algorithm

3.2 Optimization example

The following example (see fig. 3) illustrates the control problem: A time varying electric load $I_{load}(t)$ is to be supplied with power from an energy storage device (e.g. a battery) in such a way that the power losses are minimal and the State-of-Charge (SOC) is kept at a fairly constant level (to optimize the lifetime of the energy storage device). The system can be influenced from an external current source I_{opt}, which can deliver power at all times but with losses that are time-dependent. This means at times it can be efficient to charge the battery and to use the stored energy at a later time when the losses of the current source are high. α is a control variable which we choose in order to weigh the importance of the SOC-control.

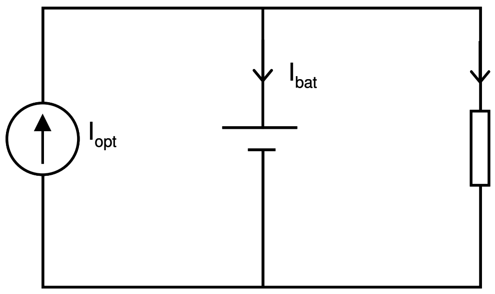


Figure 3: Example Electric Circuit

From these control objectives we define the objective function to be minimized as:

$$Cost = \underbrace{\alpha \mid I_{bat,des}(SOC) - I_{opt} \mid}_{SOCControl} + \underbrace{R_i(I_{opt} - I_{load})^2}_{BatteryLoss} + \underbrace{\gamma_I(t) I_{opt}}_{CurrentCost} \tag{1}$$

I_{opt} is our control variable; the current of the external current source. The battery losses are assumed to be a quadratic function of the current through the battery internal resistance. The SOC-optimal battery current $I_{bat,des}$ is a function of the battery SOC and is chosen to the following curve:

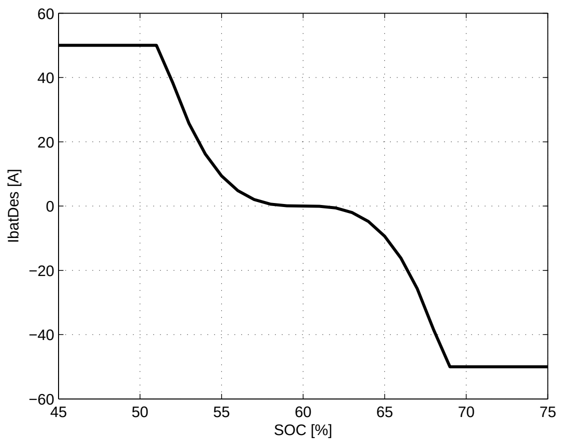


Figure 4: $I_{bat,des}$ as a function of battery state of charge

$\gamma_I(t)$ is a time-varying function that decides the loss power of the external current source. In this example, we have chosen it to be sinodial (see figure 6).

3.2.1 Results

We let the optimization algorithm defined in chapter 3.1 find the optimal solution to the objective function (1). The variable $Iopt$ is computed through a function call of `goldenSectionSearch`.

Figure 5 shows the calculated optimal current, as well as the load current and the resulting battery current. We can see that high (battery discharging) peaks in the load current have been compensated for with the current source in order to minimize the battery losses.

In figure 6, the optimized current has been compared to a control strategy that only considers the battery SOC (as described in figure 4). We can conclude that the optimization chooses to charge the battery at times when the current is inexpensive, but at the same time manages to keep the SOC at levels similar to the SOC-controlled strategy, not very far from the target value of 60%.

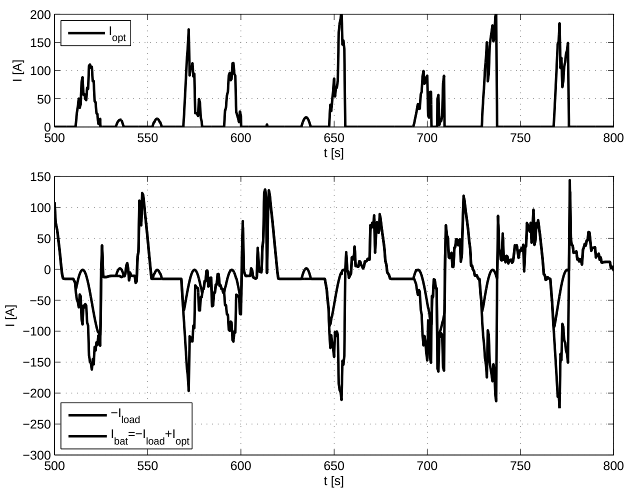


Figure 5: Optimization result: Controlled current I_{opt}, load current I_{load} and resulting battery current I_{bat}

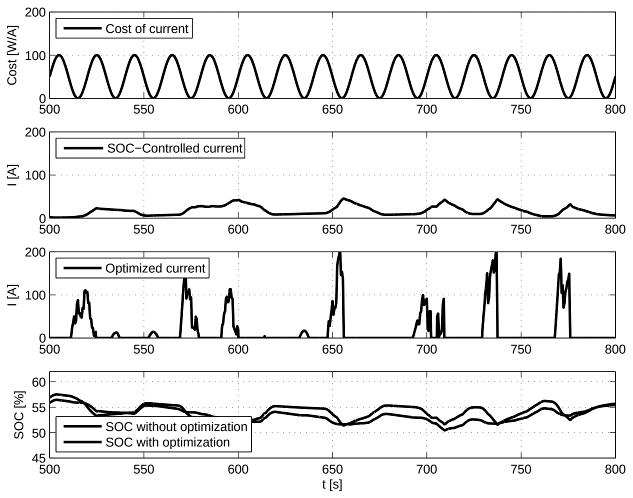


Figure 6: Optimization result: Cost of current, SOC-Controlled current, Optimized current and SOC

As a measurement on how good the optimization has worked, we compute the total system losses (battery losses and losses of the external current source). By integration of the loss power, as shown in figure 7, we see that the energy lost in the optimized system is only about half of the SOC-controlled strategy. The heat developed in the battery is proportional to the loss power, and the operating temperature of the battery rises over time. However, with the optimal control the battery losses are kept down, and the temperature remains at a lower level than the SOC-controlled strategy.

3.2.2 Implementation of tables in *Modelica* text

A difficulty in the implementation of the online optimization is the use of table look-ups for the objective function within *Modelica* text algorithm sections. In order to do this, one must initialize the table using *dymTableInit*. The table/mapping can then be called from a function using the function *dymTableIpo1* or *dymTableIpo2*.

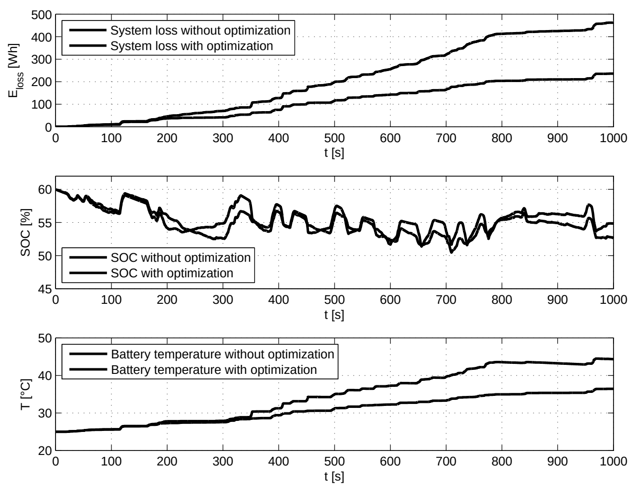


Figure 7: Comparison optimized system with SOC-controlled system: System losses, battery SOC and temperature

```
...
equation
  when initial() then
    Data.EngineFuelFlow=dymTableInit
    (2.0, smoothness, "FuelFlowAllCyl",
    engineFuelFlowTable, table, 0.0);
  end when;
...
```

Modelica Code 2: Table Interpolation in Modelica Text

3.2.3 Comments on simulation time

In a simple example like the one given above, the simulation time of a model containing an optimization algorithm is good, only somewhat slower than an equal model using a traditional control approach. However when combined with a complex vehicle model, generating a lot of events due to system state changes, a fixed-step optimization algorithm can slow the simulation time down considerably. In these cases, it has been shown that time-discrete sampling of the optimization algorithm increases the computation speed. A well considered sampled optimization algorithm delivers virtually the same result as the non-sampled, but without recomputing the optimal solution for each event triggered by the plant. Using this method, we have achieved simulation performance comparable to our traditional control concepts.

4 Hybrid vehicle application

This section will present a simulation model of a hybrid electric vehicle using a control strategy based on online optimization. In this case, the optimization only governs the choice of engine torque, but it could also be employed for the choice of gear, or in EVT-mode (Electrically Variable Transmission) the speed of the internal combustion engine. The advantage of such an implementation would be that the vehicle would adapt its gear strategy depending on the current conditions. However such a strategy also has the disadvantage that the gear choice is not always comprehensible to the driver.
The following control objectives are considered in our objective function [5]:

- Combustion engine losses
- Battery losses
- Electric machine losses
- Battery SOC control

Below simulation results from an FTP72[1] simulation of a hybrid electric vehicle are shown. In figure 8 the vehicle speed is plotted with our control signal, the optimal combustion engine torque. T_{ICE} is available for us to choose at all times except the phases where the vehicle is powered electrically. It has been chosen to minimize the listed control objectives.
Figure 9 shows the resulting power and SOC of the battery. At a given engine speed the battery power is proportional to the combustion engine torque, and therefore also directly connected to our control signal. We can conclude that even albeit a high portion of pure electrical driving in this cycle, the SOC remains around the target SOC of 60%.

5 Discussion and conclusion

This paper shows that it is possible to implement optimization algorithms for the control of a plant, e.g. a hybrid electric vehicle, in *Modelica*. Using online optimization, a fixed-step optimization algorithm can find a solution to a number of complex and interconnected control objectives. Although the optimization algorithm has to be called at each step of the simulation, the simulation time was comparable to models using traditional control strategies.

[1]The Federal Test Procedure legislation fuel cycle

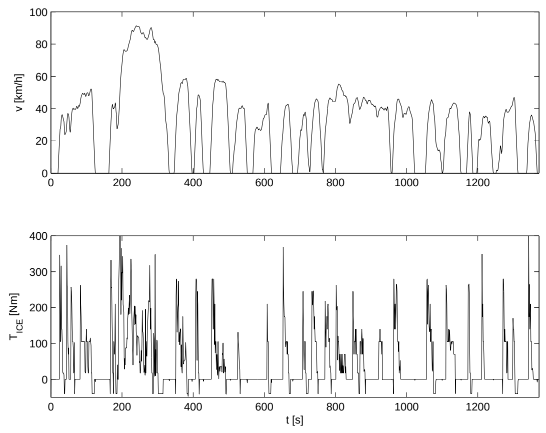


Figure 8: Vehicle speed (above) and combustion engine torque (below) as a function of time

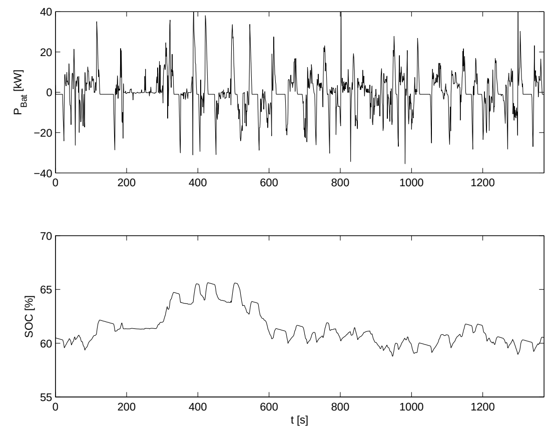


Figure 9: Battery power (above) and state of charge (below) as a function of time

References

[1] Eborn J. On Model Libraries for Thermohydraulic Applications. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2001.

[2] Tummescheit H. Design and Implementation of Object-Oriented Model Libraries using Modelica. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.

[3] Tummescheit H, Eborn J. Chemical Reaction Modeling with ThermoFluid/MF and Multi-Flash. In: Proceedings of the 2th Modelica Conference 2002, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002.

[4] Press, W. H.; Teukolsky, S. A. & Vetterling, W. T. et al. (1999), Numerical Recipes in C, The Art of Scientific Computing (second ed.), Cambridge University Press, Cambridge, ISBN 0-521-43108-5.

[5] US 2007/0032926 A1 FORD GLOBAL TECHNOLOGIES: Optimal Engine Operating Power Management Strategy for a Hybrid Electric Vehicle Powertrain. 8.2.2007.

Model Embedded Control: A Method to Rapidly Synthesize Controllers in a Modeling Environment

E. D. Tate   Michael Sasena†   Jesse Gohl†   Michael Tiller†

Hybrid Powertrain Engineering, General Motors Corp.
1870 Troy Tech Park, Troy, Michigan, 48009

†Emmeskay, Inc, 47119 Five Mile Road
Plymouth, Michigan, 48170

ed.d.tate@gm.com msasena@emmeskay.com jbgohl@emmeskay.com mtiller@emmeskay.com

Abstract

One of the challenges in modeling complex systems is the creation of quality controllers. In some projects, the effort to develop even a reasonable prototype controller dwarfs the effort required to develop a physical model. For a limited class of problems, it is possible and tractable to directly synthesize a controller from a mathematical statement of control objectives and a model of the plant. To do this, a system model is decomposed into a controls model and a plant model. The controls model is further decomposed into an optimization problem and a 'zero-time' plant model. The zero-time plant model in the controller is a copy or a reasonable representation of the real plant model. It is used to evaluate the future impact of possible control actions. This type of controller is referred to as a Model Embedded Controller (MEC) and can be used to realize controllers designed using Dynamic Programming (DP).

To illustrate this approach, an approximation to the problem of starting an engine is considered. In this problem, an electric machine with a flywheel is connected to crank and slider with a spring attached to the slider. The machine torque is constrained to a value which is insufficient to statically overcome the force of the spring. This constraint prevents the motor from achieving the desired speed from some initial conditions if it only supplies maximal torque in the desired direction of rotation. By using DP, a control strategy that achieves the desired speed from any initial condition is generated. This controller is realized in the model using MEC.

The controller for this example is created by forming an optimization problem and calling an embedded copy of the plant model. Furthermore, this controller is calibrated by conducting a large scale Design of Experiments (DOE). The experiments are processed to generate the calibrations for the controller such that it achieves its design objectives when used for closed loop control of the plant model.

It is well understood that Modelica includes many language features that allow plant models to be developed quickly. As discussed previously, the development of quality control strategies generally remains a bottleneck. In this paper we show how existing features along with appropriate tool support and potential language changes can make a significant impact on the model development process by supporting an automated control synthesis process.

Keywords: Control, Dynamic Programming, Model Embedded Control, Model Based Control, Optimal Control

1 Introduction

The use of modeling is well established in the development of complex products. Modern tools have significantly reduced the effort required to model and tune physical systems. Acausal or topological modeling reduces the effort required to model a system's physics. The use of optimization allows systematic tuning of parameters to improve a design. The combination of parameter optimization and rapid modeling allows a large set of potential designs to be quickly evaluated. However for systems which include controls, the development is, in general, a man-power intensive process subject to large uncertainty in development time and optimality. The optimization of both controls and design must be solved in many problems [1-3]. One way to address this problem is to use numerical techniques to construct controllers. For certain classes of problems, tractable numerical techniques can be used to develop an approximately minimizing controller [4]. A minimizing controller is a controller which achieves the best possible performance from a system as measured against an objective. There may exist more than one controller able to achieve this minimum, but no controller can perform better than a minimizing controller. For this work, the terms minimizing controller and optimal controller are used interchangeably.

To construct a minimizing controller, an object cost, J, is defined. This is a function which maps the state and input trajectory of the system to a scalar:

$$J = C(x,u). \tag{1}$$

Consider the special case of a plant described by ordinary differential equations with inputs that are piecewise constant. These piecewise constant inputs are updated periodically at the 'decision instances' by a controller at intervals of Δt. The total operating cost is calculated as a sum over an infinite time horizon. Furthermore, the sum of costs is discounted by the term α which is greater than zero and less than or equal to one. The total cost is calculated by an additive function that operates on the instantaneous state and the control inputs. This cost may take a form similar to

$$J\left(x(0)\right) = \sum_{k=0}^{\infty} \alpha^k \cdot \left(\int_{t=t_k}^{t_{k+1}} c_{cont}\left(x(\tau),u_k\right)\cdot d\tau \right). \tag{2}$$

The total cost in (2) is a function of the initial state of the system. To simplify notation, let the state at the decision instances be represented by

$$x_k = x(t_k). \tag{3}$$

Let the discrete time samples occur at

$$t_k = k\cdot\Delta t. \tag{4}$$

Furthermore, let the continuous-time instantaneous cost, c_{cont}, in (2) be represented in discrete time notation as an additive cost over an interval,

$$c(x_k,u_k) = \int_{t=t_k}^{t_{k+1}} c_{cont}\left(x(\tau),u_k\right)\cdot d\tau. \tag{5}$$

Using the notation developed in (2) through (5), the continuous-time system's total cost is expressed in discrete time notation as

$$J(x_0) = \sum_{k=0}^{\infty} \alpha^k \cdot c(x_k,u_k). \tag{6}$$

To simplify the continuous-time dynamics, let

$$f_d(x,u) = \left\{ \int_0^{\Delta t} f\left(\zeta(\tau),u\right)\cdot d\tau, \zeta(0)=x \right\}. \tag{7}$$

Hence,

$$x_{k+1} = f_d(x_k,u_k). \tag{8}$$

An optimal control choice for each time step can be found using the dynamic programming equations,

$$u^*(x) \in \arg\min_{u\in U(x)} \left\{ c(x,u) + \alpha\cdot V\left(f_d(x,u)\right) \right\}. \tag{9}$$

The function $V(x)$ is known as the value function. By using the dynamic programming (DP) equations to find the value function, a minimizing controller is obtained. The DP equations are

$$V(x) = \min_{u\in U(x)} \left\{ c(x,u) + \alpha\cdot V\left(f_d(x,u)\right) \right\}, \tag{10}$$

where

$$U(x) = \left\{ u \middle| g(x,u) \le 0 \right\} \tag{11}$$

defines the set of feasible actions, $U(x)$. For the case where the total cost is considered over an infinite horizon and $u_k = u^*(x_k)$ (see eq (9)), the value function is the same as the total cost function, $J(x) = V(x)$. Equation (10) can be solved through value iteration, policy iteration, or linear programming. See [5-23] for discussion of solution methods. For discussion of using DP to find value functions for automotive control application, see [24-29]. The formulation of equation (11) is chosen to simplify management of constraints throughout the model and to conform to a standard form used in the optimization community, the negative null form [30].

One problem with solving (10) is that when the state space consists of continuous states, $V(x)$ is a function from one infinite set to another. Except in special cases, this requires approximation to solve. One common approach is to use linear bases to approximate the value function. Possible linear bases include the bases for multi-linear interpolation, the bases for barycentric interpolation, b-splines, and polynomials. See the appendices in [25] for a discussion of linear bases for dynamic programming. In the case where $V(x)$ is approximated by a linear basis,

$$V(x)=\Phi^T(x)\cdot w, \tag{12}$$

where

$$\Phi(x)=\begin{bmatrix} f_1(x) & f_2(x) & \cdots & f_N(x)\end{bmatrix}. \tag{13}$$

An approximate solution to (10) is found by finding the weights, w, which solve

$$\Phi^T(x)\cdot w=\min_{u\in U(x)}\left\{c(x,u)+\alpha\cdot\Phi^T\left(f_d(x,u)\right)\cdot w\right\}. \tag{14}$$

See [5-7] for a discussion of using linear bases to form the value function.

It is important to understand that this controller is an optimal controller for the discrete time case only, when the controller updates every Δt seconds. In other uses, the controller will generally be suboptimal. Additionally, any development algorithm based on this methodology will suffer from the curse of dimensionality [31]. In other words, the time to find an optimal controller will increase geometrically with the size of the plant state space. As a point of reference, using a single commercially available PC from 2005, a five state controller was found in less than twenty four hours.

2 Controller Development

To use equations (2) through (14) to develop a controller, it is necessary to have a plant model which includes the dynamics (f), cost function (c), and constraints (g) all coupled to an integrator which can be invoked as a function call by a Control Design Algorithm (CDA). In addition, the set of states for the plant model and the set of controller actions must be specified to the CDA. For this work, a custom wrapper was developed that allowed batches of states and actions to be efficiently evaluated. Each evaluation returned the state at the next interval, the cost of operation for the interval, and the constraint activity over the interval.

To understand the structure of the equations involved in this work, consider a system consisting of a plant and a controller. Without loss of generality, assume the plant dynamics are described by ordinary differential equations

$$\dot{x}=f(x,u), \tag{15}$$

where f is a function that describes the plant dynamics. For notational simplicity consider a continuous time controller. Let the controller be a full state feedback controller implemented as a static mapping, M, from the state, x, to the action set, u:

$$u=M(x). \tag{16}$$

Assuming only a single global minimum exists, the dynamic programming equations in (9) can be directly used for the static mapping (16). The autonomous dynamics of this system are then described by the following equation

$$\dot{x}=f\left(x,\underset{u\in U(x)}{\arg\min}\left\{c(x,u)+\alpha\cdot V\left(f_d(x,u)\right)\right\}\right) \tag{17}$$

This equation is then integrated to solve for $x(t)$,

$$x(t)=\int_0^t f\left(x(s),\underset{u\in U(x)}{\arg\min}\left\{\begin{matrix}c(x(s),u)\\ +\alpha\cdot V\left(f_d\left(x(s),u\right)\right)\end{matrix}\right\}\right)\cdot ds \tag{18}$$

where

$$x(0)=x_0 \tag{19}$$

defines the initial conditions. To evaluate f_d from (7), a nested integrator, which is independent of the primary simulation integrator, is required. This nested integrator executes in 'zero-time' from the perspective of the primary integrator. We refer to this as an embedded or nested simulation. Because the nested integrator is used inside a numeric optimization, it will potentially be called multiple times at each primary integrator evaluation. If f_d in (18) is expanded using (7), the plant dynamics function, f, from (15) occurs in two locations in

$$x(t)=\int_0^t f\left(x(s),\underset{u\in U(x)}{\arg\min}\left\{\begin{matrix}c(x(s),u)+\\ \alpha\cdot V\left(\int_0^{\Delta t} f(\zeta(\tau),u)\cdot d\tau,\zeta(0)=x(s)\right)\end{matrix}\right\}\right)\cdot ds \tag{20}$$

where

$$x(0)=x_0 \tag{21}$$

The nested copy of the plant dynamics equations, f, is referred to as the embedded or nested model. In the case where the controller is modeled as updating periodically, rather than continuously, the solution to the optimization problem is held constant between controller updates.

The equation structure in (20) and the reuse of the plant dynamics function, f, offer the ability to quickly synthesize controllers using numerical techniques. However, existing tools make the implementation of this type of model problematic. There are two primary issues in implementation. The first is execution efficiency. Few commercial tools have been developed with the goal of efficiently solving this class of equations. Secondly, several commercial modeling environments make the definition and reuse of the plant model cumbersome, requiring significant efforts during development and maintenance. Fortunately, the features of Modelica make the definition and reuse of a plant model manageable. The examples that follow have been developed in Dymola ®, however this general approach has also been used with Simulink® and AMESim®.

To systematically generate a system with an optimal controller, a model of the plant is generated. This plant model is 'wrapped' with an application programming interface (API) so a control design algorithm can determine the state space, the action space, the state at the next time step, the constraint activity, and the cost for a given state and action. This interaction between the plant model, the API and the control design algorithm is illustrated in Figure 1. The CDA queries the API to determine the structure of the state and action space. Given this structure and the configuration of the CDA, a sequence of DOEs is executed. The DOE data are used to find a solution to (10). For this work, the value function was modeled using multi-linear interpolation and a solution to (14) was found. To simplify coding, value iteration was used [5, 6] to find $V(x)$.

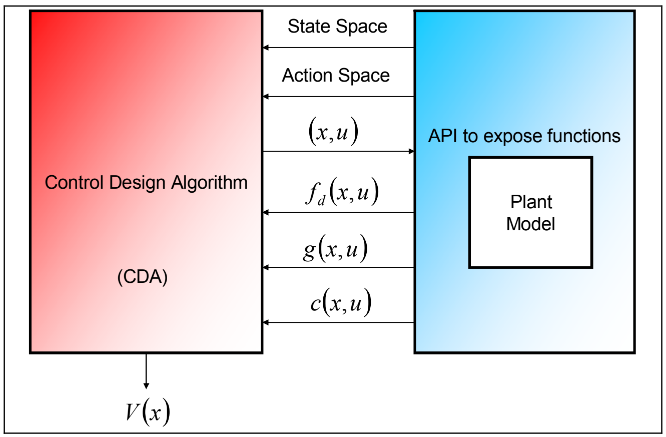


Figure 1 - Plant Model API

Once the value function is generated, the system model is formed by one of two methods. The first method is by generating a lookup table that maps the state variables to an action as in (16). The process of generating a value function, finding a mapping equivalent to (9), and realizing a controller as a mapping (or lookup table) is referred to as Indirect Model Embedded Control (IMEC). This method is appropriate for some systems. Another approach, which is more computationally expensive, is referred to as Direct Model Embedded Control (DMEC). For DMEC, the controller is realized by forming an optimization statement around an embedded copy of the plant model. This structure is illustrated in Figure 2.

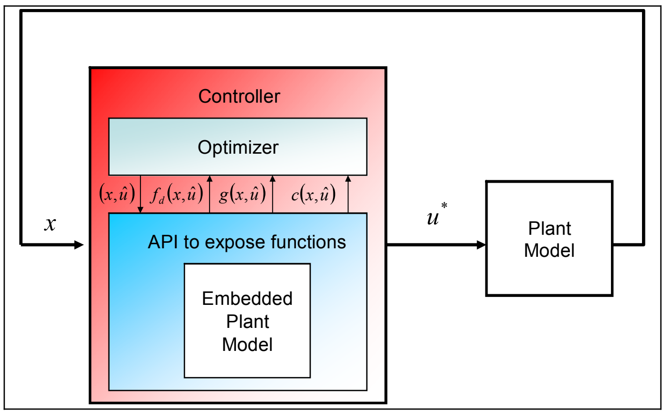


Figure 2 - Direct Model Embedded Controller Structure

To realize a Direct Model Embedded Controller (DMEC), two pieces are added to the system model. The first piece is an optimizer which solves (9). This optimizer can be as simple as a Design of Experiments (DOEs) which considers a fixed set of actions, and selects one which minimizes (9). For more sophistication, if the nature of the problem permits it, a gradient-based optimizer can be employed [30, 32, 33]. If the nature of the problem does not allow solution using these types of approaches, global solvers can be used [34-36]. Ideally, an optimization library should support both gradient and non-gradient methods for constrained optimization problems. As part of this project, libraries for performing both DOEs and gradient-based optimizations were implemented entirely in Modelica. However, there are currently no comparable commercial or public domain libraries available. The second piece required to implement a DMEC is the ability to invoke a function which efficiently initializes and simulates, over a 'short' time horizon, a set of models which are copies of the plant model with modified parameters. Because of the structure of the problem, each time the controller executes, multiple embedded simulations will execute. Depending on the nature of the action set, the number of embedded simulations may vary from as few as two embedded simulations to several thousand embedded simulations.

3 Example – Simple Engine Start

To illustrate how these concepts are used to build a controller, consider the problem of starting an internal combustion engine using an electric machine with insufficient torque to guarantee the engine completes a revolution from all possible stationary starting points. If the initial position of the engine is in a range of angles, the electric machine will stall. To simplify the modeling, let us assume the engine can be approximated using a crank slider connected to a spring. The system model, shown in Figure 3, consists of an electrical motor connected to the crank which connects through the crank slider mechanism to a piston which is subject to damping from friction. Inertia is present in the motor rotor, crankshaft and piston. The electric machine is subject to constraints on minimum and maximum torque.

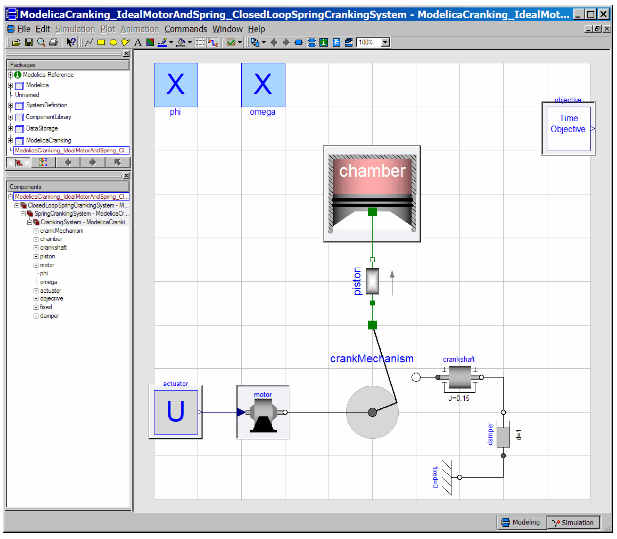

ModelicaCranking_IdealMotorAndSpring_ClosedLoopSpringCrankingSystem - ModelicaCranking_IdealMot...
File Edit Simulation Plot Animation Commands Window Help
Packages
Components
X
phi
X
omega
Time Objective
chamber
piston
crankMechanism
crankshaft
J=0.15
actuator
U
motor
damper
d=1
fixed
Modeling
Simulation

Figure 3 - Engine Starting Model

The objective of the control system is to ensure the engine will overcome the initial compression torque from any initial state and minimize engine start time. The total cost of operation (what is being minimized) is expressed mathematically as the total time taken to achieve a speed greater than or equal to five hundred RPM. Once this speed is achieved, the controller is deactivated and another scheme is used to manage the engine. The total cost of operation for this system is considered over an infinite time horizon and is computed as

$$J\left(x(0)\right)=\int_{0}^{\infty}\begin{Bmatrix}0 & ,\omega(t)\ge 500\text{ rpm}\\ 1 & ,\text{otherwise}\end{Bmatrix}\cdot dt\,. \qquad (22)$$

The instantaneous cost for this system is

$$c(x)=\begin{Bmatrix}0 & ,\omega > 500\text{ rpm}\\ 1 & ,\text{otherwise}\end{Bmatrix}. \qquad (23)$$

This type of cost generates a 'shortest-path' controller. The controller will minimize the total time to achieve 500 rpm. The total cost in (22) is undiscounted. Therefore the discounting factor, α, which is visible in (2) is assigned a value of one and omitted from the expression.

While it is clear that the system has exactly two states, they can be selected somewhat arbitrarily. For this example, the engine angle and engine speed were selected. With these variables as the states, the controller is represented as a static map from the engine angle and engine speed to the electric machine torque.

$$u=M\left(\omega,\theta\right) \qquad (24)$$

The feasible action set is a single real number, the motor torque, bounded by the constraints on motor torque and power. The set of feasible actions is defined by

$$U\left(x\right)=\left\{u\,\middle|\,\begin{matrix}-100\le u\le 100,\\ -10000\le u\cdot\omega\le 10000\end{matrix}\right\}. \qquad (25)$$

The value function was represented using multi-linear interpolation, see equation (12).

The plant model was implemented in Modelica. The Controller Design Algorithm (CDA) was implemented in MATLAB®. The CDA invoked function calls to a custom API, similar to Figure 1, applied to the plant model in Dymola®. The CDA solved for the weights, w, in the value function (equation (12)). This value function was used to generate an Indirect Model Embedded Controller (IMEC) and a Direct Model Embedded Controller (DMEC). The value function generated by the CDA is shown in Figure 4.

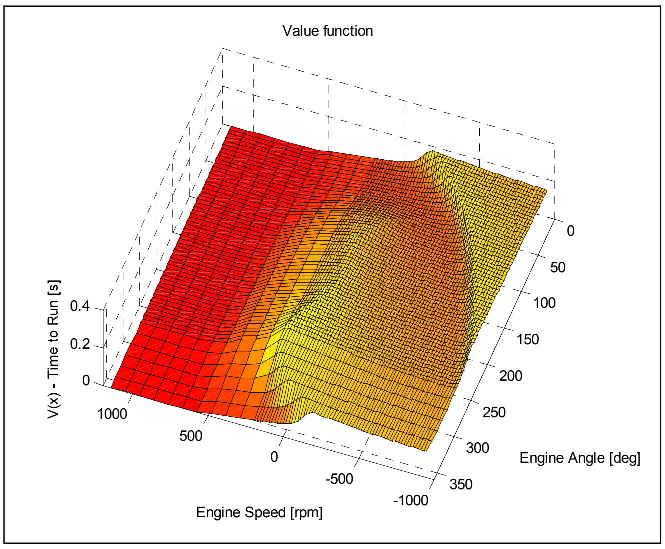


Figure 4 - Value function

The IMEC was realized as a two input lookup table with multi-linear interpolation on a regular grid. The grid points in the table were found by solving (9) using the value function generated by the CDA. This controller was implemented using standard Modelica components. The actuator commands for the IMEC controller are shown in Figure 5 as a function of engine speed and angle.

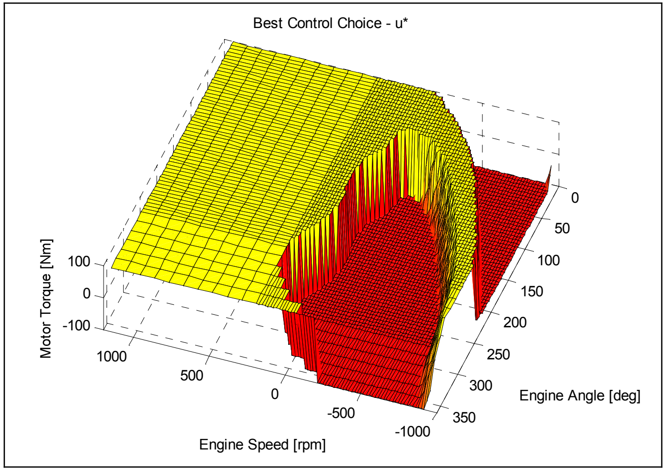


Figure 5 - IMEC control table

The DMEC was realized by wrapping a copy of the plant model with an API similar to the one used for the CDA. A DOE was used to search feasible actions. The resulting code structure is identical to Figure 2. The optimal action was chosen to minimize (9).

For both of these controllers, the problem of starting the engine from any initial condition was solved. The solution involved the counter-intuitive approach of spinning the engine backwards, then reversing direction to allow enough energy to be stored in the inertia to overcome the spring force. From a model and a control objective, an optimal controller with very complex behaviors was numerically generated in less than 10 minutes on a single PC (3GHz, 2Gb RAM). Furthermore, a similar problem with four states was solved in less than three hours. Of course the power of this approach can only be realized once a sufficient level of tool support is available so that the time required to set up the analysis is on the same order as the solution time.

3.1 Direct vs Indirect MEC

Ideally, both an IMEC and DMEC will result in identical behaviors. However, differences in approximation schemes and interpolation can results in appreciable differences. In many cases, while Indirect MEC is simpler to realize in a model, there are good reasons to implement a controller with the complexity and computational cost of a Direct MEC.

As an example, consider the previous problem. The value function, V(x), was found using the Control Design Algorithm (CDA). The IMEC controller was designed by solving for the best electric machine torque for a set of engine angles and speeds on a regular grid. For engine states which occur off this grid, multi-linear interpolation was used to calculate the control action. When the IMEC was used in an engine start simulation, if the optimal torque transitioned between positive and negative, the interpolation caused a smooth change in the torque because of the continuity imposed by interpolation.

Alternatively, consider a Direct MEC. Because of the characteristics of the dynamic programming equations and the value function, the optimal choices are either full positive or full negative torque. This results in an instantaneous, non-continuous change in torque. When plotted as in Figure 6, the difference between the control inputs and the state evolution of the system can be seen. The interpolation due to the approximation in the IMEC results in artifacts in the control actions and a slight loss of performance in the system. Mathematically this means that more detail is required to resolve $u^*(x)$, the function that we are ultimately trying to formulate, than to resolve $V(x)$.

There are cases where an IMEC is superior to a DMEC approach (*e.g.* [26] illustrates just such a case). In general, an IMEC implementation is superior when both the action set is continuous and the optimal actions are continuous. The DMEC approach is superior when either the action set is discrete or the optimal actions are not continuous with respect to the state. One example where DMEC is clearly superior is where the motor is controlled by selecting the state of a switch inverter. In this case, the action set consists of a finite set of choices for switch configuration and the optimal actions are not continuous with respect to the state.

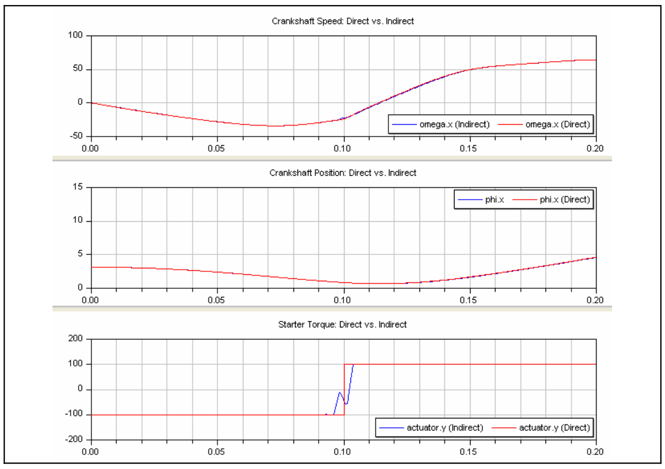


Figure 6 - Comparison of IMEC and DMEC results

4 Implementation of optimization algorithms

One of the challenges in Direct Model Embedded Control is the implementation of an optimizer. While this work was performed using a Design of Experiments (DOE) to select optimal actions, this approach becomes intractable when equality constraints and larger dimensional actions sets are considered. Towards the goal of supporting these classes of problems, a gradient-based optimizer was developed. One of the goals in developing this optimizer was to fully implement the optimizer in Modelica. By fully implementing in Modelica, all of the information used by the optimizer would be accessible for speed improvements by the compiler. Should native support for model embedding become available, all equations associated with a Direct MEC would be accessible to the compiler for speed improvement. Additionally, since the embedded simulations in a DMEC can be completely decoupled from each other, simulation tools could easily exploit the coarse grained parallelism on multi-core CPUs by running several embedded simulations concurrently when conducting searches in the optimizer (e.g. line searches and numerical gradients).

The optimizer was developed in Modelica to solve a constrained optimization problem which is generally stated in negative null form [30] as

$$\left\{\begin{array}{l} \min\limits_{\chi} f_{objective}(\chi) \\ s.t. \\ g_{inequalities}(\chi) \le 0 \\ h_{equalities}(\chi) = 0 \end{array}\right\}. \qquad (26)$$

To implement a gradient optimizer, the optimizer functionality was separated from the objective function ($f_{objective}$), the inequality constraint functions ($g_{inequalities}$), and the equality constraint functions ($h_{equalities}$). The optimizer was designed under the assumption that the inequality constraint functions are all in negative null form: feasible inequality constraints are less than or equal to zero. The objective function was assumed to be a minimization objective. Since Modelica does not (yet) support the concept of methods or passing of functions as arguments, the optimizer was designed to use static inheritance. For this reason, the objective and constraint functions are replaceable functions within an optimizer package.

One feature of this library, that is not commonly available, is the ability to handle functions which are undefined over some region. The domain of the objective and constraints may not be known a priori. This occurs with MEC applications because the objective (e.g. equation (9)) and constraint functions (e.g. equation (11)) are typically evaluated using a solver. The solver may not find a solution. Hence, classical algorithms must be modified to recover from undefined evaluations.

Implementation of this capability was problematic because of the lack of numeric support for a real value which represents the concept of an undefined quantity. Either a native capability similar to Matlab's ® 'NaN', or operator overloading with the ability to extend a class from real numbers would have simplified implementation.

In this library, Modelica.Constants.inf was used to indicate that a function call was undefined. However, the language specification does not define behavior for operations (e.g. addition, subtraction, multiplication, division) on Modelica.Constants.inf. Therefore, all functions and statements which operated on variables that might be assigned a value of Modelica.Constants.inf required conditional expressions to ensure expected behavior.

While this optimization library will not be publicly released, it is available for further development. Contact the lead author for a copy.

5 Recommendations

While it is possible to realize both IMEC and DMEC controllers using Modelica 2.2, the addition of a standard optimization library and native support for embedded model simulation would vastly simplify implementation and maintenance.

Towards the goal of simplifying implementation of MEC, a recommended language improvement is the addition of a 'model simulate' function. The function would accept arguments that specify the model to simulate, the parameter values to use in each simulation, the outputs to return, and any solver specific settings. The solver should be able to be configured to solve both initialization problems and simulation problems. For efficiency in evaluation, the function should support both a scalar and vector lists of parameters. In addition to results which are associated with the model, there should be results associated with the solver. These results should be sufficient to diagnose solver failures. At a minimum, these should include the final time in the evaluation and an indication of whether the simulation successfully completed. A sample function definition along with an example invocation are shown in Figure 7.

```
function simulateModel
 input String modelName;
 input String paramNames[:];
 input String resultNames[:];
 input Real
        paramValues[:,size(paramNames,1)];
 input SettingsRecord solverSettings;
 output Real
      results[size(paramValues,1),
             size(resultNames,1)];
 …
end simulateModel;

// example call
[angle, speed, exitCondition, exitTime] =
    simulateModel(
        modelName="Library.PlantModel",
        paramNames{"w0", "theta0","u"},
        resultNames=
           {"w", "theta",
            "exitCondition", "exitTime" },
        paramValues=
              [0, 0, -100;
               1, 0, -100;
               …;
               2, 2*pi, 100],
        solverSettings =
            SettingsRecord(
                stopTime=1.0,
                fixedStep=0.1)
        );
```

Figure 7 - Model evaluation

It is important to point out that the goal is to be able to invoke such a function from within a running model and not simply as a command line analysis option. As previously mentioned, the ability to directly express such nested simulation relationships makes posing MEC problems much easier. If the MEC problem could also directly express the “optimization problem” associated with MEC then tools could also bring the underlying symbolic information to bear on efficient gradient evaluation as well.

One remaining issue for DMEC problems is the initialization of state variables in the embedded model. For DMEC problems we typically want the embedded model to start at the current state of the parent simulation. Said another way, the current values of the states in the parent simulation should be used as initial conditions in the nested simulation. Of course, it is possible using the function in Figure 7 to establish such a mapping but hopefully the language design group will consider alternatives that would be less tedious and error prone.

6 Conclusions

It is tractable to numerically synthesize near optimal (or approximately minimal) controllers for many systems. While in most cases the state feedback required for the controllers may make them impractical to deploy, they can certainly be used as prototype controllers that establish performance limits for a given design as well as provide insights into control laws for production controllers. Furthermore, this approach can easily integrate into a combined plant-controller optimization process. This can be done by making the optimal controller a function of the plant parameters. These optimal controllers can be realized as lookup tables (IMEC) or through the use of optimization and embedded models (DMEC). An algorithmic approach to controls synthesis was presented. For this paper, the IMEC and DMEC approaches were applied to an engine starting problem to generate an optimal controller in an automated fashion.

As this work has shown, Modelica is a promising technology for rapid prototyping of subsystem designs and prototype controllers. However, lack of support for ‘model embedding’ makes development and long term maintenance problematic because considerable work must be done to implement this embedding. Lacking any language standard, this work will always be tool specific. Furthermore, implementation of controllers which rely on optimization suffer from the lack of a standard optimization library. While an optimization library was developed for this work, it isn’t practical for most users to make such an investment. By adding both language support to express the essential aspects of model embedding and optimization discussed in this paper, Modelica can evolve into a powerful technology for system development and optimization.

References

[1] H. K. Fathy,"Combined Plant and Control Optimation: Theory, Strategies, and Applications," Mechanical Engineering, University of Michigan, Ann Arbor, 2003.

[2] H. K. Fathy, P. Y. Papalambros, A. G. Ulsoy, and D. Hrovat, "Nested Plant/Controller Optimization with Application to Combined Passive/Active Automotive Suspensions."

[3] H. K. Fathy, J. A. Reyer, P. Y. Papalambros, and A. G. Ulsoy, "On the Coupling between the Plant and Controller Optimization Problems," in *American Control Conference*, Arlington, Va, 2001.

[4] P. R. Kumar and P. Variaya, *Stochastic Systems: Estimation, Identification and Adaption*. Englewood Cliffs, New Jersey: Prentice Hall, 1986.

[5] D. Bertsekas, *Dynamic Programming and Optimal Control: Vol 2*. Belmont, Mass: Athena Scientific, 1995.

[6] D. P. Bertsekas, *Dynamic Programming and Optimal Control: Vol 1*. Belmont, Mass: Athena Scientific, 1995.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, Mass: Athena Scientific, 1996.

[8] M. A. Trick and S. E. Zin, "A Linear Programming Approach to Solving Stochastic Dynamic Programs," Carnegie Mellon University 1993.

[9] M. A. Trick and S. E. Zin, "Spline Approximations to Value Functions: A Linear Programming Approach," *Macroeconomic Dynamics,* pp. 255-277, 1997.

[10] D. P. de Farias and B. Van Roy, "The Linear Programming Approach to Approximate Dynamic Programming," *Operations Research,* vol. 51, pp. 850-865, November-December 2003.

[11] V. F. Farias and B. Van Roy, "Tetris: Experiments with the LP Approach to Approximate DP," 2004.

[12] D. P. de Farias,"The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application," Ph.D. Dissertation, Department of Management Science and Engineering, Stanford University, Palo Alto, Ca, 2002.

[13] D. Dolgov and K. Laberteaux, "Efficient Linear Approximations to Stochastic Vehicular Collision-Avoidance Problems," in *Proceedings of the Second International Conference on Informatics in Control, Automation, and Robotics (ICINCO-05)*, 2005.

[14] G. J. Gordon, "Stable Function Approximation in Dynamic Programming," January 1995.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Mass: MIT Press, 1999.

[16] R. Munos and A. Moore, "Barycentric Interpolators for Continuous Space and Time Reinforcement Learning," *Advances in Neural Information Processing Systems,* vol. 11, pp. 1024-1030, 1998.

[17] R. Munos and A. Moore, "Variable Resolution Discretization in Optimal Control," *Machine Learning,* vol. 1, pp. 1-24, 1999.

[18] J. M. Lee and J. H. Lee, "Approximate Dynamic Programming Strategies and Their Applicability for Process Control: A Review and Future Directions," *International Journal of Control, Automation, and Systems,* vol. 2, pp. 263-278, September 2004.

[19] D. P. de Farias and B. Van Roy, "Approximate Value Iteration with Randomized Policies," in *39th IEEE Conference on Decision and Control* Sudney, Australia, 2000.

[20] D. P. de Farias and B. Van Roy, "Approximate Value Iteration and Temporal-Difference Learning," in *IEEE 2000 Adaptive Systems for Signal Processing, Communications and Control Symposium*, 2000, pp. 48-51.

[21] B. Van Roy and J. N. Tsitsiklis, "Stable Linear Approximations to Dynamic Programming for Stochastic Control Problems with Local Transitions," *Advances in Neural Information Processing Systems,* vol. 8, 1996.

[22] P. W. Keller, S. Mannor, and D. Precup, "Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning."

[23] V. C. P. Chen, D. Ruppert, and C. A. Shoemaker, "Applying Experimental Design and Regression Splines to High Dimensional Continuous State Stochastic Dynamic Programming," *Operations Research,* vol. 47, pp. 38-53, January-February 1999.

[24] C.-C. Lin, H. Peng, and J. W. Grizzle, "A Stochastic Control Strategy for Hybrid Electric Vehicles," in *Proceedings of the 2004 American Control Conference*, 2004, pp. 4710-4715 vol. 5.

[25] E. D. Tate,"Techniques of Hybrid Electic Vehicle Controller Synthesis," Electrical Engineering: Systems, University of Michigan, Ann Arbor, Michigan, 2007.

[26] E. Tate, J. Grizzle, and H. Peng, "Shortest Path Stochastic Control for Hybrid Electric Vehicles," *Internation Journal of Robust and Nonlinear Control,* 2006.

[27] I. Kolmanovsky, I. Siverguina, and B. Lygoe, "Optimization of Powertrain Operating Policy for Feasibility Assessment and Calibration: Stochastic Dynamic Programming Approach," in *Proceedings of the American Control Conference*, Anchorage, AK, 2002, pp. 1425-1430.

[28] J.-M. Kang, I. Kolmanovsky, and J. W. Grizzle, "Approximate Dynamic Programming Solutions for Lean Burn Engine Aftertreatment," in *Proceedings of the 38th Conference on Decision & Control*, Phoenix, Arizona, 1999, pp. 1703-1708.

[29] C.-C. Lin, H. Peng, J. W. Grizzle, and J.-M. Kang, "Power Management Strategy for a Parallel Hybrid Electric Truck," *IEEE Transactions on Control Systems Technology,* vol. 11, pp. 839-849, November 2003.

[30] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design: Models and Computation*, 2 ed. New York, New York: Cambridge University Press, 2000.

[31] J. Rust, "Using Randomization to Break the Curse of Dimensionality," 1996.

[32] S. Boyd and L. Vendenberghe, *Convex Optimization*. New York, N.Y.: Cambridge University Press, 2004.

[33] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. New York, N.Y.: Academic Press, 1981.

[34] D. R. Jones, C. D. Peritunen, and B. E. Stuckman, "Lipschitzian Optimization without the Lipschitz Constant," *Journal of Optimization Theory and Applications,* vol. 79, pp. 157-181, 1993.

[35] A. J. Booker, J. Dennis, J. E. , P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, "A Rigorous Framework for Optimization of Expensive Functions by Surrogates."

[36] M. J. Sasena,"Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations," Mechanical Engineering, University of Michigan, Ann Arbor, 2002.

Session 4d

Mechanical Systems & Applications

High-Accuracy Orbital Dynamics Simulation through Keplerian and Equinoctial Parameters

Francesco Casella   Marco Lovera
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract

In the last few years a Modelica library for spacecraft modelling and simulation has been developed, on the basis of the Modelica Multibody Library. The aim of this paper is to demonstrate improvements in terms of simulation accuracy and efficiency which can be obtained by using Keplerian or Equinoctial parameters instead of Cartesian coordinates as state variables in the spacecraft model. The rigid body model of the standard MultiBody library is extended by adding the equations defining a transformation of the body center-of-mass coodinates from Keplerian and Equinoctial parameters to Cartesian coordinates, and by setting the former as preferred states, instead of the latter. The remaining parts of the model, including the model of the gravitational field, are left untouched, thus ensuring maximum re-usability of third-party code. The results shown in the paper demonstrate the superior accuracy and speed of computation in the reference case of a point-mass gravity field.
Keywords: Spacecraft dynamics; Orbit dynamics; Numerical integration; State selection.

1 Introduction

The Modelica Spacecraft Dynamics Library ([6, 7, 10]) is a set of models (based on the already existing and well known Multibody Library, see [9]) which is currently being developed with the aim of providing an advanced modelling and simulation tool capable of supporting control system analysis and design activities for both spacecraft attitude and orbit dynamics. The main motivation for the development of the library is given by the significant benefits that the adoption of a systematic approach to modelling and simulation, based on modern a-causal object-oriented languages such as Modelica, can give to the design process of such advanced control systems.

At the present stage, the library encompasses all the necessary utilities in order 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 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 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, control algorithms).

More precisely, the generic spacecraft simulator consists of an Extended World model and one or more Spacecraft models. The Extended World model is an extension of Modelica.MultiBody.World which provides all the functions needed for a complete representation of the space environment as seen by a spacecraft: gravitational and geomagnetic field models, atmospheric models, solar radiation models. 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. The Spacecraft model, on the other hand, is a completely reconfigurable spacecraft including components to describe the actual spacecraft dynamics, the attitude/orbit control sensors and actuators and the relevant control laws. In this paper we are specifically concerned with the Spacecraft model; this component has been defined by extending the already available standard model Modelica.Mechanics.MultiBody.Parts.Body. The main modifications reside in the selectable evaluation of the in-

teractions 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. The main drawback associated with the adoption of the standard Body model as the core of the Spacecraft model is related to the intrinsic use this component makes of the Cartesian coordinates in the World reference frame for the state variables associated with the motion of the Body's center of mass. Indeed, for spacecraft work it is well known that significant benefits, both in terms of simulation accuracy and computational performance, can be obtained by using different choices of state variables, such as Keplerian and Equinoctial parameters (see, e.g., [11, 8]).
Therefore, the aims of this paper, which extends preliminary results presented in [2] are the following:

- to demonstrate improvements in terms of simulation accuracy and efficiency which can be obtained by using Keplerian and Equinoctial parameters instead of Cartesian coordinates as state variables in the spacecraft model;
- to illustrate how Keplerian and Equinoctial parameters can be included in the existing multibody spacecraft model by exploiting the object-oriented features of the Modelica language and the symbolic manipulation capability of Modelica tools.

The paper is organised as follows: first an overview of the available choices for the state representation of satellite orbits is given in Section 2; subsequently, the use of Keplerian and Equinoctial orbital elements for the simulation of orbit dynamics will be described in Section 3, while the corresponding Modelica implementation will be outlined in Section 4 and the results obtained in the implementation and application of the proposed approach to the simulation of a Low Earth and Geostationary orbits will be presented and discussed in Section 5.

2 Satellite State Representations

The state of the center of mass of a satellite in space needs six quantities to be defined. These quantities may take on many equivalent forms. Whatever the form, we call the collection of these quantities either a state vector (usually associated with position and velocity vectors) or a set of elements called orbital elements (typically used with scalar magnitude and angular representations of the orbit). Either set of quantities is referenced to a particular reference frame and completely specifies the two-body orbit from a complete set of initial conditions for solving an initial value problem class of differential equations.
In the following subsections, we will deal with spacecraft subject only to the gravitational attraction of the Earth considered as a point mass (*unperturbed Keplerian conditions*) and we will refer mainly to the Earth Centered Inertial reference axes (ECI), defined as follows. The origin of these axes is in the Earth's centre. The X-axis is parallel to the line of nodes. The Z-axis is parallel to the Earth's geographic north-south axis and pointing north. The Y-axis completes the right-handed orthogonal triad.

2.1 Position and Velocity Coordinates

In the ECI reference frame, the position and velocity vectors of a spacecraft influenced only by the gravitational attraction of the Earth considered with punctiform mass will be denotated as follows

$$r = \begin{bmatrix} x & y & z \end{bmatrix}^T, \tag{1}$$

$$v = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T = \frac{dr}{dt}. \tag{2}$$

The acceleration of such a spacecraft satisfies the equation of two-body motion

$$\frac{d^2 r}{dt^2} = -GM_\oplus \frac{r}{\|r\|^3} \tag{3}$$

where $\mu = GM_\oplus$ is the gravitational coefficient of the Earth. A particular solution of this second order vector differential equation is called an orbit that can be elliptic or parabolic or hyperbolic, depending on the initial values of the spacecraft position and velocity vectors $r(t_0)$ and $v(t_0)$. Only circular and elliptic trajectories are considered in this study.
The state representation by position and velocity of a spacecraft in unperturbed Keplerian conditions is

$$x_{ECI} = \begin{bmatrix} r^T & v^T \end{bmatrix}^T \tag{4}$$

at a given time t. Time t is always associated with a state vector and it is often considered as a seventh component. A time used as reference for the state vector or orbital elements is called the epoch.

2.2 Classical Orbital Elements

The most common element set used to describe elliptical orbits (including circular orbits) are the classical orbital elements (COEs), also called the Keplerian parameters, which are described in the sequel of this Section. The COEs are defined as follows:

- a : semi-major axis, [m];
- n : mean motion, [rad/s]
- e : eccentricity, [dimensionless];
- i : inclination, [rad];
- Ω : right ascension of the ascending node, [rad];
- ω : argument of perigee, [rad];
- ν : true anomaly, [rad];
- E : eccentric anomaly, [rad];
- M : mean anomaly, [rad];

(see Figures 1 and 2). The definitions of the COEs are referenced to the ECI frame. The semi-major axis a specifies the size of the orbit. Alternatively, the mean motion

$$n = \sqrt{\frac{GM_{\oplus}}{a^3}} \tag{5}$$

can be used to specify the size.
The eccentricity e specifies the shape of the ellipse. It is the magnitude of the eccentricity vector, which points toward the perigee along the line of apsis.
The inclination i specifies the tilt of the orbit plane. It is defined as the angle between the angular momentum vector $h = r \times v$ and the unit vector Z.
The right ascension of the ascending node Ω is the angle from the positive X axis to the node vector n pointing toward the ascending node, that is the point on the equatorial plane where the orbit crosses from south to north. The argument of perigee ω is measured from the ascending node to the perigee, i.e., to the eccentricity vector e pointing towards the perigee.
The eccentric anomaly E is defined on the auxiliary circle of radius a, that can be drawn around the elliptical orbit, as shown in Figure 2. Finally, the mean anomaly M is defined as $M = n(t - t_p)$, where t_p denotes the time of perigee passage, i.e., the instant at which the eccentric anomaly vanishes. As is apparent from its definition, the mean anomaly for an ideal Keplerian orbit increases uniformly over time. E and M are related by the well known Kepler equation

$$E - e\sin(E) = M. \tag{6}$$

In this work, satellite state representation in terms of classical orbital elements (Keplerian parameters) will be denoted as

$$x_{COE} = \begin{bmatrix} a & e & i & \Omega & \omega & M \end{bmatrix}^T \tag{7}$$

with the implicit choice of adopting M as a parameter to represent the spacecraft anomaly; the advantages and disadvantages of this choice will be discussed in the following.

2.3 Equinoctial Orbital Elements

COEs suffer from two main singularities. The first is when the orbit is circular, i.e., when the eccentricity is zero ($e = 0$). In this case the line of apsis is undefined and also the argument of perigee ω. The second occurs when the orbit is equatorial, i.e., when the inclination is zero ($i = 0$). In this case the ascending node is undefined and also the right ascension of the ascending node Ω. See Figure 1.

It is nevertheless possible to define the true, eccentric and mean longitude (L, K and l, respectively) as

$$L = \omega + \Omega + \nu, \tag{8}$$
$$K = \omega + \Omega + E, \tag{9}$$
$$l = \omega + \Omega + M; \tag{10}$$

these quantities remain well-defined also in the singular cases of circular and/or equatorial orbits.

The equinoctial orbital elements (EOEs) avoid the singularities encountered when using the classical orbital elements. EOEs were originally developed by Lagrange in 1774. Their definitions in terms of Keplerian elements are given by the following equations

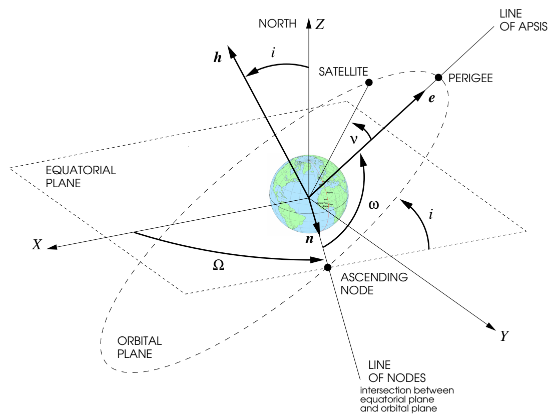


Figure 1: Classical Orbital Elements (COEs).

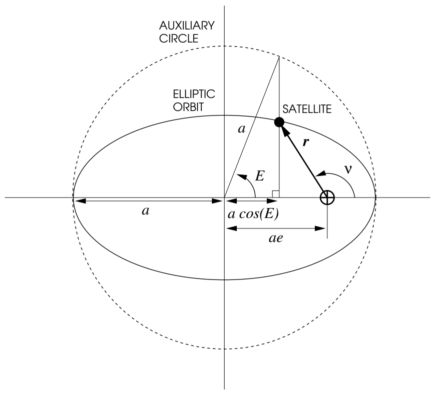


Figure 2: True and eccentric anomalies for elliptic motion.

(see, e.g., [1, 5, 8] for details)

$$a, \tag{11}$$
$$P_1 = e\sin(\omega + I\Omega), \tag{12}$$
$$P_2 = e\cos(\omega + I\Omega), \tag{13}$$
$$Q_1 = \tan(i/2)\sin\Omega, \tag{14}$$
$$Q_2 = \tan(i/2)\cos\Omega, \tag{15}$$
$$l = \Omega + \omega + M. \tag{16}$$

True retrograde equatorial orbits ($i = 180°$) cause problems because Q_1 and Q_2 are undefined. This problem is solved by introducing a retrograde factor I which is $+1$ for direct orbits and -1 for retrograde orbits. In this work, dealing with geostationary satellites, I is equal to $+1$ and the mean longitude net of the Greenwich Hour Angle $\Theta(t)$

$$l_\Theta = l - \Theta(t) \tag{17}$$

will be used instead of the mean longitude l given by equation (16). GEO satellite state representation in terms of equinoctial orbital elements will be denoted as follows

$$x_{EOE} = \begin{bmatrix} a & P_1 & P_2 & Q_1 & Q_2 & l_\Theta \end{bmatrix}^T. \tag{18}$$

The definitions of the EOEs are referenced to the equinoctial reference frame, which can be obtained from the ECI reference frame by a rotation through the angle Ω about the Z axis, followed by a rotation through the angle i about the new X axis (which points in the same direction as the node vector n pointing the ascending node), followed by a rotation through the angle $-I\Omega$ about the new Z axis (which points in the same direction as the h vector). In the equinoctial frame the elements P_1 and P_2 represent the projection of the eccentricity vector onto the Q and E directions, respectively (see Figure 3). The elements Q_1 and Q_2 represent the projection of the vector oriented in the direction of the ascending node with magnitude $\tan(i/2)$, onto the Q and E directions, respectively. Note that in the singular cases of circular (or equatorial) orbits, the vector P (or Q) becomes zero; the indetermination in the two components of each vector is thus not a problem.

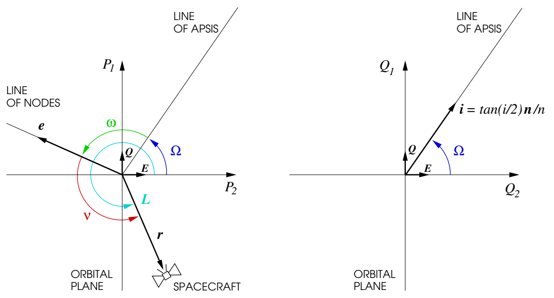


Figure 3: Eccentricity and inclination equinoctial components and true longitude.

2.4 Conversion formulae: COEs to Cartesian

The position coordinates in the orbital plane, centered in the Earth (Figure 2) are related to the COEs by the following equations

$$\begin{bmatrix} x_{orb} \\ y_{orb} \end{bmatrix} = \begin{bmatrix} a\cos(E) - ae \\ a\sin(E)\sqrt{1-e^2} \end{bmatrix}. \tag{19}$$

while the corresponding velocities can be computed as

$$\begin{bmatrix} v_{x,orb} \\ v_{y,orb} \end{bmatrix} = \begin{bmatrix} -\frac{a^2 n}{|r|}\sin(E) \\ \frac{a^2 n}{|r|}\sqrt{1-e^2}\cos(E) \end{bmatrix}, \tag{20}$$

with $|r| = \sqrt{x_{orb}^2 + y_{orb}^2} = \sqrt{r^T r}$. As depicted in Figure 1, the orthogonal basis RTN of the Gaussian coordinate system can be obtained from the orthogonal basis XYZ of the ECI frame by means of three successive rotations

$$\begin{bmatrix} x_{orb} \\ y_{orb} \\ 0 \end{bmatrix} = \mathscr{R}_{ZXZ}(x_{COE}) \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{21}$$

with

$$\mathscr{R}_{ZXZ}(x_{COE}) = \mathscr{R}_Z(\omega)\mathscr{R}_X(i)\mathscr{R}_Z(\Omega) \tag{22}$$

where matrix

$$\mathscr{R}_Z(\Omega) = \begin{bmatrix} \cos\Omega & \sin\Omega & 0 \\ -\sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

describes the first rotation around the Z axis of an angle Ω, matrix

$$\mathscr{R}_X(\Omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos i & \sin i \\ 0 & -\sin i & \cos i \end{bmatrix} \tag{24}$$

describes the second rotation around the X of an angle i, matrix

$$\mathscr{R}_Z(\omega) = \begin{bmatrix} \cos(\omega) & \sin(\omega) & 0 \\ -\sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{25}$$

describes the third rotation around the Z axis of an angle ω. Thanks to the orthonormal property of rotation matrices, equation (21) can be easily inverted, giving

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathscr{R}_Z^T(\Omega)\mathscr{R}_X^T(i)\mathscr{R}_Z^T(\omega) \begin{bmatrix} x_{orb} \\ y_{orb} \\ 0 \end{bmatrix}; \tag{26}$$

following the same reasoning, the Cartesian velocity vector can be expressed as

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \mathscr{R}_Z^T(\Omega)\mathscr{R}_X^T(i)\mathscr{R}_Z^T(\omega) \begin{bmatrix} v_{x,orb} \\ v_{y,orb} \\ 0 \end{bmatrix}. \tag{27}$$

Further details can be found, e.g., in [11]. Summarizing, it is possible to compute x_{COE}, given x_{ECI}, by first solving the scalar implicit equation (6) for E, and then the explicit vector equations (19)-(20), (26)-(27).

2.5 Conversion formulae: EOEs to Cartesian

The conversion formulae from EOEs to Cartesian coordinates in ECI are slightly more involved. The results are summarised here; for further details, the reader is referred to, e.g., [1, 5, 8].

The eccentric longitude K can be computed by solving the implicit equation

$$l_{theta} + \Theta(t) = K + P_1\cos(K) - P_2\sin(K). \tag{28}$$

The ECI coordinates are then given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \rho \begin{bmatrix} (1+Q_2^2-Q_1^2)\cos(L) + 2Q_1Q_2\sin(L) \\ (1+Q_1^2-Q_2^2)\sin(L) + 2Q_1Q_2\cos(L) \\ 2Q_2\sin(L) - 2Q_1\cos(L) \end{bmatrix}, \tag{29}$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = n \begin{bmatrix} \frac{x}{|r|}\frac{d|r|}{dl} + \sigma\left[(1+Q_2^2-Q_1^2)\frac{dcos(L)}{dK} + 2Q_1Q_2\frac{dsin(L)}{dK}\right] \\ \frac{y}{|r|}\frac{d|r|}{dl} + \sigma\left[(1+Q_1^2-Q_2^2)\frac{dsin(L)}{dK} + 2Q_1Q_2\frac{dcos(L)}{dK}\right] \\ \frac{z}{|r|}\frac{d|r|}{dl} + 2\sigma\left[Q_2\frac{dsin(L)}{dK} - Q_1\frac{dcos(L)}{dK}\right] \end{bmatrix}, \tag{30}$$

where

$$\begin{aligned} |r| &= a(1-P_1\sin(K) - P_2\cos(K)), \\ \rho &= \frac{|r|}{1+Q_1^2+Q_2^2}, \\ \sigma &= \frac{a}{1+Q_1^2+Q_2^2}, \\ \gamma &= 1+\sqrt{1-P_1^2-P_2^2} \\ \sin(L) &= \frac{a}{\gamma|r|}\left[(\gamma-P_2^2)\sin(K) + P_1P_2\cos(K) - \gamma P_1\right] \\ \cos(L) &= \frac{a}{\gamma|r|}\left[(\gamma-P_1^2)\cos(K) + P_1P_2\sin(K) - \gamma P_2\right]. \end{aligned}$$

3 COEs and EOEs for simulation of orbit dynamics

When orbital control problems are studied, it is usually necessary to integrate the equations of motion of the satellite under the action of gravity (due to the Earth or any other celestial body), of the space environment and of the actuators' thrust. The usual approach, known as Cowell's method (see [3]), is to integrate the equations of motion in cartesian coordinates

$$\dot{r} = v \tag{31}$$

$$\dot{v} = a_g(r) + \frac{F}{m} \tag{32}$$

where a_g is the acceleration of gravity, F is the sum of all the other forces, and m is the satellite mass. applied by the actuators. First-cut models assume a point-mass model

$$a_g = -GMr/\|r\|^3, \tag{33}$$

while accurate simulations require more detailed models of the gravitational field, usually in the form of a series expansion (see, e.g., [12]). In both cases, the differential equations are strongly non-linear; therefore, despite the use of high-order integration algorithm, tight tolerances end up in a fairly high number of simulation steps per orbit.

If the satellite motion is described in terms of COEs or EOEs, it is easy to observe that the variability of the six orbit elements is much smaller than that of the Cartesian coordinates. In particular, it is well-known that in case of a point-mass gravity field with no other

applied forces, the first five parameters are constant, while the mean anomaly and the mean longitude increase linearly with time. All existing high-order integration methods have error bounds which depend on Taylor expansions of the state trajectory. One can then conjecture that if the COEs/EOEs are used as state variables, instead of the Cartesian vectors r and v, the state trajectories will be smoother, and therefore the integration algorithm will be able to estimate them with with a higher relative precision using much larger time steps, compared to the Cartesian coordinates case.
Recalling the definition of vector x_{ECI} in (4), letting $z = x_{COE}$ or $z = x_{EOE}$ depending on the choice for the new state variables and denoting by $g(\cdot)$ the transformation relating z and x, equations (31)-(32) can be written in compact form as

$$\dot{x} = f(x) \tag{34}$$

$$x = g(z). \tag{35}$$

If a state variable change from x to z is now performed, the following equations are obtained

$$\frac{\partial g(z)}{\partial z}\dot{z} = f(g(z)) \tag{36}$$

which can be solved for $\dot{z}$ provided that the new state variables z are uniquely defined

$$\dot{z} = \left(\frac{\partial g(z)}{\partial z}\right)^{-1} f(g(z)) \tag{37}$$

$$x = g(z). \tag{38}$$

The Jacobian for g_{COE} is generically well defined and becomes singular only in the case of a circular and/or equatorial orbit. In this case the EOEs are needed, as the Jacobian for g_{EOE} is well defined in this case.
The model (37)-(38), which is now in standard state-space form, has two very important features:

- the right-hand side of (37) is much less variable than the right-hand side of (34), so it will be easier to integrate the equations with a higher accuracy;
- in case an accurate model of the gravity field is used, it is not necessary to reformulate it in terms of the COEs/EOEs, as the right-hand side of (37) uses the compound function $f(g(z))$.

Remark 1 *The accurate computation of long-term solutions for dynamical problems associated with pure orbital motion has been a subject of extensive research for decades. In particular, the so-called class of symplectic integration methods (see, e.g., [4] and the references therein) provides an effective and reliable solution to the problem. In the framework of the present study, however, the aim is to improve accuracy in the computation of orbital motion while retaining the advantages associated with the use of a general-purpose object-oriented modelling environment, in which not only orbital dynamics can be simulated, but also the coupled attitude motion, as well as the associated mathematical models of sensors, actuators and controllers for orbital and attitude control. This more general framework requires the use of general-purpose integration algorithms for ODEs/DAEs.*

4 Modelica implementation

The concepts outlined in Section 3 are easily implemented with the Modelica language. The starting point is the Body model of the standard Modelica.Mechanics.MultiBody library [9]: this is a 6 degrees-of-freedom model of a rigid body, which can be connected to other components to form a multibody system model. The original model has six degrees of freedom, corresponding to 12 state variables: the three cartesian coordinates and the three velocity components of the center of mass, plus three suitable variables describing the body orientation and the three components of the angular velocity vector. Assuming that the gravitational field is applied exactly at the center of mass (the gravity gradient effect is computed in a separate model and thus not included here), the translational and rotational equations are completely decoupled, so it is possible to focus on the former ones, leaving the latter ones untouched.
First of all, the equations to compute the gravity acceleration *as a function of the cartesian coordinates* using accurate field models are added by inheritance to the standard World model of the MultiBody library, which only offers the most basic options of no gravity, constant gravity and point mass gravity (see [7, 10]). Then, the standard Body model must be enhanced by:

1. adding the COEs a, e, i, ω, Ω, M or the EOEs a, P_1, P_2, Q_1, Q_2, l_Θ as new model variables;
2. adding the equations relating COEs/EOEs to the cartesian coordinates;
3. switching the stateSelect attribute for the r and v vectors of the Body model to StateSelect.avoid, and for the COEs/EOEs to StateSelect.prefer.

The Modelica compiler tool will then perform the transformation from (34)-(35) to (37)-(38) automatically, using symbolic manipulation algorithms.

A first implementation option is to extend the Body model by inheritance, adding the above-mentioned features, and thus deriving two enhanced models BodyKepler and BodyEquinoctial; this approach is documented in [2].

A second option is to put the additional variables and equations in a separate model with a multibody flange interface, and then connect it to the unmodified Body model within a wrapper model that also sets the preferred state variables. This option perfectly fits the architecture of the Spacecraft Dynamics library, where such a structure was already used in order to include the models of the interaction of the satellite with the space environment: gravity gradient torque, aerodynamic drag, solar radiation, etc. (see [10], Fig. 3). In fact, the library described in [10] already contained a similar model to compute the orbital parmeters; that model, however, contained explicit inverse conversion formulae (from cartesian coordinates to COEs), and was designed to be used with cartesian coordinates as states. Since either the COEs or the EOEs can be used, the wrapper model must actually contain two conditionally declared, mutually exclusive models (one for each choice of coordinates), which are both connected to the standard Body model; a flag in the wrapper model decides which of the two will actually be activated in the simulation model.

The Modelica code defining the new models is very compact and easy to check, which is an important feature to ensure the correctness of the resulting model. As already noted, the accurate models of the gravity field, previously implemented in [7, 10], can still use the Cartesian coordinates as inputs, and are thus left unchanged.

As to the computational efficiency, the workload at each time step is increased, compared to the standard ECI formulation, by the conversion formulae, the Jacobian computation and the solution of the linear system (37). However, as will be demonstrated in the next section, this additional overhead is more than compensated by the fact that the differential equations are much easier to integrate in the new state variables, resulting in a faster simulation time and in a much tighter accuracy.

5 Simulation examples

In this Section, the results obtained in comparing the accuracy obtained by simulating the orbit dynamics for two Low Earth orbiting (LEO) spacecraft and a GEO one will be presented. As previously mentioned, for the purpose of the present study we focus on the simulation of the unperturbed dynamics, i.e., only the gravitational acceleration computed from a point-mass model for the Earth is considered. In this case, the orbit is an ellipse (closed curve), having well-defined features. Therefore, this assumption allows us to introduce two simple criteria in order to evaluate the accuracy of the performed simulations, namely:

- The period of an unperturbed elliptical orbit can be computed a priori and is given by $T = 2\pi\sqrt{\frac{a^3}{\mu}}$, so a first measure of simulation accuracy can be given by the precision with which the orbit actually closes during the simulation. To this purpose, the following stopping criterion has been defined for the simulation: the integration is stopped when the position vector crosses a plane orthogonal to the initial velocity and passing through the initial position. Then, the final time is compared with the orbit period and the final position is compared with the initial one.
- Furthermore, for an unperturbed orbit the angular momentum $h = r \times v$ should remain constant, so a second measure of accuracy for the simulation is given by the relative error in the value of h, i.e., the quantity

$$e_h = \frac{\|h - h(0)\|}{\|h(0)\|}. \tag{39}$$

The considered orbits have been simulated using the Dymola tool, using Cartesian and Keplerian/Equinoctial coordinates, in order to evaluate the above-defined precision indicators. The DASSL integration algorithm has been used, with the smallest feasible relative tolerance 10^{-12}. The RADAU algorithm has also been tried with the same relative tolerance, yielding similar results which are not reported here for the sake of conciseness.

5.1 A near-circular, LEO orbit

The first considered orbit is a LEO, near circular one (see Figure 4), characterised by the following initial state, in Cartesian coordinates:

$$r(0) = \begin{bmatrix} 6828.140 \times 10^3 \\ 0 \\ 0 \end{bmatrix},$$

$$v(0) = \begin{bmatrix} 0 \\ 5.40258602956241 \times 10^3 \\ 5.40258602956241 \times 10^3 \end{bmatrix}$$

The results obtained in the comparison of Cartesian and Keplerian coordinates are summarised in Table 1. As can be seen from the Table, the precision achieved in the actual closure of the orbit improves significantly when using Keplerian coordinates as states: the simulated period is very close to the actual one and both the period error and the position error are significantly smaller.

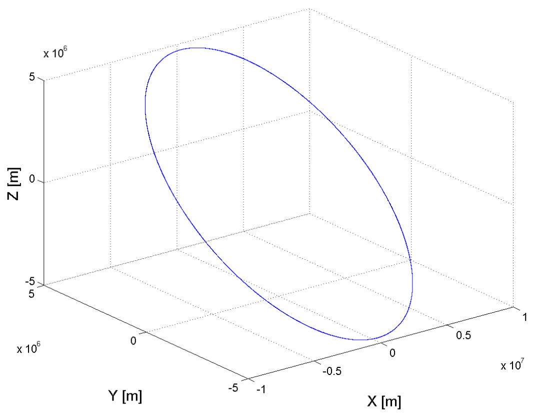


Figure 4: The considered LEO, near circular orbit.

Similarly, in Figure 5 the time histories of the relative error on the value of the orbital angular momentum are illustrated, for a simulation of about one day: the results are clearly very satisfactory in both cases, however while in the case of Cartesian states the relative error is significantly larger than machine precision and is slowly increasing, in the case of Keplerian states the relative error is much smaller and appears to be more stable as a function of time (see also the mean value of the relative angular momentum error, given in Table 1). Finally, note that the use of Keplerian parameters also gives significant benefits in terms of simulation efficiency, as can be seen from the last column of Table 1.

5.2 A highly elliptical, LEO orbit

The second considered orbit is again a LEO one, but it is characterised by a high value of the eccentricity

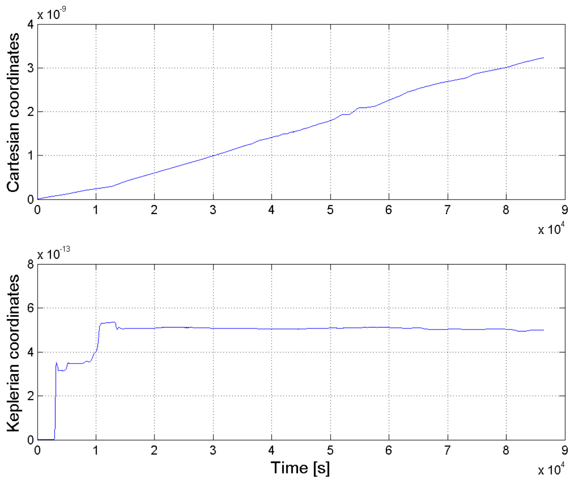


Figure 5: Relative errors on the orbit angular momentum - near circular orbit: Cartesian (top) and Keplerian (bottom) coordinates.

(see Figure 6, where it is also compared with the circular orbit considered in the previous case) and by the following initial state, in Cartesian coordinates:

$$r(0) = \begin{bmatrix} 6828.140 \times 10^3 \\ 0 \\ 0 \end{bmatrix},$$

$$v(0) = \begin{bmatrix} 0 \\ 5.40258602956241 \times 10^3 \\ 7.29349113990925 \times 10^3 \end{bmatrix}$$

As in the previous case, Table 2 shows the precision achieved in the actual closure of the orbit: as can be seen, the errors on the simulated period are of the same order of magnitude for both choices of state variables. The position errors, on the other hand are significantly smaller when simulating the orbital motion using Keplerian rather than Cartesian states.

Similarly, in Figure 7 the time histories of the relative error on the value of the orbital angular momentum are illustrated, for a simulation of about one day. In this case, the results show that using Cartesian states the relative error is again significantly larger than machine precision and is slowly increasing, while using Keplerian states the relative error is of the order of machine precision.

Finally, the gain in terms of simulation efficiency can be verified from the last column of Table 2.

Table 1: Orbit closure errors, relative angular momentum error and number of steps using Cartesian and Keplerian coordinates - near circular orbit.

States	ΔT [s]	$\|\Delta r\|$ [m]	Mean e_h	Number of steps
Cartesian	-1.00332×10^{-6}	1.69711×10^{-3}	1.5373×10^{-9}	959
Keplerian	2.38369×10^{-8}	2.17863×10^{-5}	4.7528×10^{-13}	376

Table 2: Orbit closure errors, relative angular momentum error and number of steps using Cartesian and Keplerian coordinates - highly elliptical orbit.

States	ΔT [s]	$\|\Delta r\|$ [m]	Mean e_h	Number of steps
Cartesian	-1.17226×10^{-5}	4.39241×10^{-3}	1.2927×10^{-10}	3650
Keplerian	1.48665×10^{-5}	2.67799×10^{-7}	2.5223×10^{-16}	1120

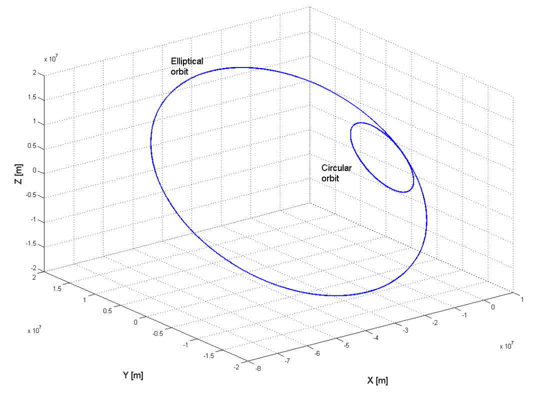


Figure 6: The considered LEO, highly elliptical orbit, compared with the circular one considered in Section 5.1.

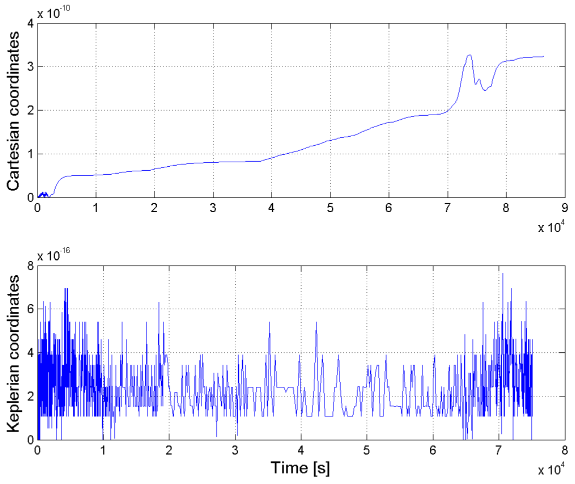


Figure 7: Relative errors on the orbit angular momentum - highly elliptical orbit: Cartesian (top) and Keplerian (bottom) coordinates.

5.3 A GEO orbit

The last considered orbit is a GEO one, characterised by the following initial state, in Cartesian coordinates:

$$r(0) = \begin{bmatrix} 4.21641 \times 10^7 \\ 0 \\ 0 \end{bmatrix},$$

$$v(0) = \begin{bmatrix} 0 \\ 3074.66 \\ 0 \end{bmatrix}$$

Table 3 shows the accuracy improvement achieved when simulating the orbital motion using Equinoctial rather than Cartesian states. As in the previous case, also for the simulation of GEO orbits it appears from the inspection of the time histories of the relative error on the orbital angular momentum (depicted in Figure 8) that in the case of Cartesian states the relative error is slowly increasing over time, while in the case of Equinoctial states the relative error appears to be more stable (see also Table 3).

Finally, the advantages provided by the use of Equinoctial parameters in terms of simulation efficiency are confirmed by the data provided in the last column of Table 3.

6 Concluding remarks

A method for the accurate simulation of satellite orbit dynamics on the basis of the Modelica MultiBody library has been presented. The proposed approach is based on the use of Keplerian and Equinoctial parame-

Table 3: Orbit closure errors, relative angular momentum error and number of steps using Cartesian and Equinoctial coordinates - GEO orbit.

States	ΔT [s]	$\|\Delta r\|$ [m]	Mean e_h	Number of steps
Cartesian	-2.79186×10^{-5}	1.88208×10^{-2}	1.0323×10^{-10}	793
Equinoctial	-2.92057×10^{-8}	8.91065×10^{-5}	6.8574×10^{-16}	20

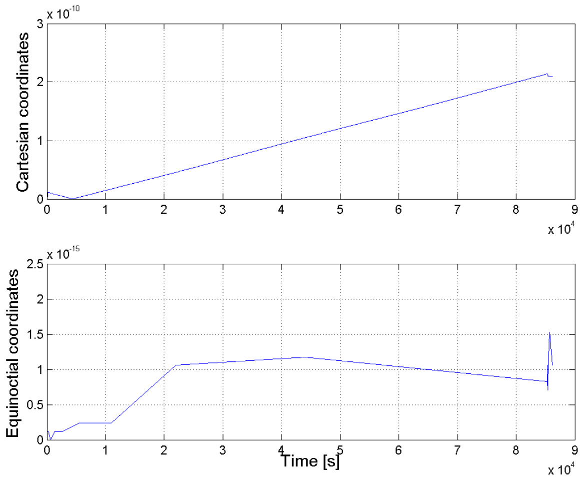


Figure 8: Relative errors on the orbit angular momentum - GEO orbit: Cartesian (top) and Equinoctial (bottom) coordinates.

ters instead of Cartesian coordinates as state variables in the spacecraft model. This is achieved by adding to the standard Body model the equations for the transformation from Keplerian and Equinoctial parameters to Cartesian coordinates and exploiting automatic differentiation. The resulting model ensures a significant improvement in numerical accuracy and a reduction in the overall simulation time, while keeping the same interface and multibody structure of the standard component. Simulation results with a point-mass gravity field show the good performance of the proposed approach. The validation with higher order gravity field models is currently being performed.

References

[1] R.A. Broucke and P.J. Cefola. On the equinoctial orbit elements. *Celestial Mechanics and Dynamical Astronomy*, 5(3):303–310, 1972.

[2] F. Casella and M. Lovera. High accuracy simulation of orbit dynamics: an object-oriented approach. In *Proceedings of the 6th EUROSIM Congress, Lubiana, Slovenia*, 2007.

[3] V. Chobotov. *Orbital Mechanics*. AIAA Education Series, Second edition, 1996.

[4] V.V. Emel'yanenko. A method of symplectic integrations with adaptive time-steps for individual Hamiltonians in the planetary N-body problem. *Celestial Mechanics and Dynamical Astronomy*, 98(3):191–202, 2007.

[5] D. Losa. *High vs low thrust station keeping maneuver planning for geostationary satellites*. PhD thesis, Ecole Nationale Superieure des Mines de Paris, 2007.

[6] M. Lovera. Object-oriented modelling of spacecraft attitude and orbit dynamics. In *54th International Astronautical Congress, Bremen, Germany*, 2003.

[7] 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.

[8] O. Montenbruck and E. Gill. *Satellite orbits: models, methods, applications*. Springer, 2000.

[9] M. Otter, H. Elmqvist, and S. E. Mattsson. The new Modelica multibody library. In *Proceedings of the 3nd International Modelica Conference, Linköping, Sweden*, 2003.

[10] T. Pulecchi, F. Casella, and M. Lovera. A Modelica library for Space Flight Dynamics. In *Proceedings of the 5th International Modelica Conference, Vienna, Austria*, 2006.

[11] M. Sidi. *Spacecraft dynamics and control*. Cambridge University Press, 1997.

[12] J. Wertz. *Spacecraft attitude determination and control*. D. Reidel Publishing Company, 1978.

Rotational3D — Efficient modelling of 3D effects in rotational mechanics

Johan Andreasson
Magnus Gäfvert
Modelon AB
Ideon Science Park
SE–223 70 Lund, Sweden
E-mail: {johan.andreasson,magnus.gafvert}@modelon.se

Abstract

In this work, it is described how effects of rotational mechanics represented in three dimensions can be described in an efficient way. The ideas have been implemented in the Rotational3D library which is introduced in this paper. Special attention is payed to the influence of joints and how 1D rotational and multibody representations can be combined. Comparison of accuracy and efficiency with MultiBody is shown as well as typical application examples.
Keywords: Rotational mechanics, Rotational3D, Multi-Body, Driveline, Shaft

1 Introduction

There is large class of rotational mechanical systems where 1-dimensional revolution, or spin, is the dominating motion. Examples of such systems are automotive drivelines, transmissions, and steering mechanisms [1, 2]. These rotational systems are often part of general 3-dimensional multi-body systems with which they interact. This may be reaction forces and torques from brakes or motors, or gyroscopic moments resulting from motion orthogonal to the spin direction. It can also be kinematic effects on the spin motion from universal joints with bending angle coupled to 3-dimensional motion of multi-body parts.
Systems of this class may in parts be described with the Modelica Standard Library (MSL) [3] by using the the 1-dimensional formalism of the Rotational library, or by the full multi-body formalism of the MultiBody library [4]. The 1-dimensional approach gives a low-complexity representation that allows for fairly straightforward modeling of components such as clutches and gears. Simple reaction torques can be applied to multi-body parts by using the Mounting1D component. Gyroscopic effect may be included by using the Rotor1D component instead of the normal rotational inertia. These composite Rotational and MultiBody models in the Modelica Standard Library were introduced in [5]. They were designed for modeling of automatic transmissions, and are useful in contexts where the complete rotational mechanism is mounted on one multi-body part. In more general cases they have the limitations of not describing all interaction effects. Other drawbacks are the lack of information on mechanism geometry and the very rudimentary visualization.
Using MSL, a full model that includes all interaction effects requires that also the spin motion is described with MultiBody models. This results in overly complex and inefficient models where the 1-dimensional rotation is hidden in transformation matrices. The rotation angle is difficult and computationally expensive to extract. It is also only available in the range $[-\pi, \pi]$ and it is therefore tricky to track revolutions. Components such as shafts introduces a lot of constraint equations when defined with standard multi-body joints.
The Rotational3D library was designed to combine the advantages of the Rotational and MultiBody approaches. In particular, this means:

- Efficient description of the spin motion
- Reaction torques and forces
- Geometry and kinematic effects
- Visualization
- Interfacing to both MultiBody and Rotational

As a result, Rotational3D is very suitable for e.g. automotive applications and it is an established part of the VehicleDynamics Library [6].

2 Modelling principles

The main idea is to describe the spin motion similarly as in the Rotational library, and provide references that are coupled to MultiBody frames.

2.1 Connector definition

To represent the reference and the shaft motion, the following information must be communicated in a connector.

1. The rotation angle around the axis of rotation
2. The torque around the axis of rotation
3. The forces and torques acting on the reference frame
4. A reference frame with position and orientation
5. The direction of the axis of rotation, n
6. The axis that defines zero rotation, q

Items 1 and 2 can be described by a Rotational flange and items 3 and 4 by a MultiBody frame. In addition, the reference axes n and q are required. The n and q axes were explicitly defined in the connector in early versions of Rotational3D. Now they are instead implicitly defined as the x and y axes of the reference frame, resulting in the following connector definition[1]:

```
connector Flange
  MB.Interfaces.Frame frame
    "Reference frame";
  Rt.Interfaces.Flange flange
    "Rotation around frame x-axis relative y-axis";
end Flange;
```

In Figure 1, the representation of the connector is seen.

2.2 Coupling 1D and 3D effects

Consider the fundamentals of an inertia, the angular acceleration of the inertia is a sum of the contributions of the flange and the frame so that the resulting torque at the flange, τ_{flange} is defined[2] as

$$\tau_{flange} = J\left(\dot{\omega}_{flange} + \hat{n}\cdot\dot{\overline{\omega}}_{frame}\right) \qquad (1)$$

[1] A similar connector definition is also used in [7] and has been introduced in the Modelica Standard Library as "FlangeWithBearing". The scope of this is different as it essentially is intended as a Rotational connector with an optional "bearingFrame" that allow for reaction torques to be applied.

[2] This can also be found in [5].

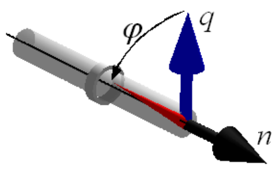


Figure 1: Animation view of an axle in a bearing with a connector reference shown. Axis-of-rotation (n), reference rotation (q), actual rotation (red arrow) and the angle φ.

and the torque at the frame, $\overline{\tau}_{frame}$ is defined by

$$\overline{\tau}_{frame} + \hat{n}\tau_{flange} = J\left(\hat{n}\dot{\omega}_{flange} + \overline{\omega}_{frame} \times \hat{n}\omega_{flange}\right). \qquad (2)$$

Here ω_{flange}, $\overline{\omega}_{frame}$, $\hat{n}$, and J refer to the angular velocity of flange and frame, spin axis, and inertia, respectively. As a result, the torque at the flange, depends on the motion of the frame and vice versa.
Another example of how information from both connectors are required to formulate a model is a tyre model. Typically, the longitudinal force generated by the tyre is dependent on the wheel's spin velocity. Unlike for example a wheel speed sensor that measures the speed relative to the hub, the total rotational velocity around the spin axis, ω_{tot}, is required.

$$\omega_{tot} = \omega_{flange} + \hat{n}\cdot\overline{\omega}_{frame} \qquad (3)$$

2.3 Avoiding over- and under-determined systems

Consider Figure 2: With two mounted bearings (1), each bearing is mounted to a MultiBody frame and by supplying parameters for n and q, the rotation axis and reference is defined in the Rotational3D connector. This model is over-determined since two connectors, both specifying the reference is connected. Correspondingly, if none of the connectors in a set would have the reference specified, the model would be under-determined (2).
This is avoided by a connection rule using two connectors with different colours, the grey is unconstrained and the white is constrained: *Each connection set must have one and only one white connector and there must never be a loop containing only white connectors.*
This rule is somewhat simplified, consider for example the under-determined example from Figure 2 (3). Here, both ends of the shaft are attached to white connectors and there is no loop consisting on only white connectors. Still, the model is over-constrained

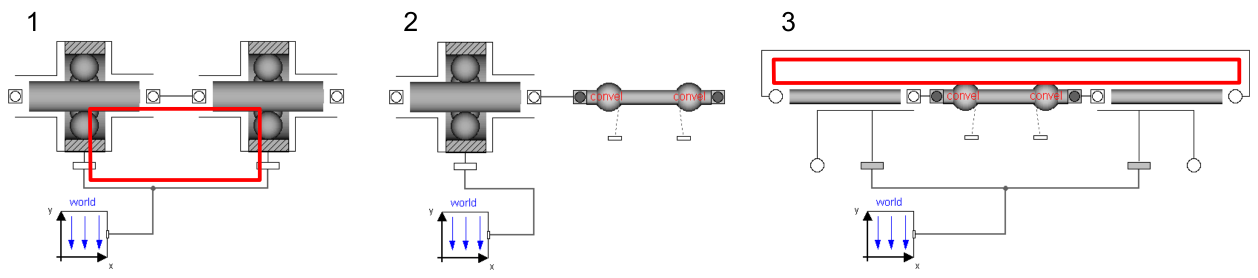


Figure 2: Example of under- and over-determined systems: Loop consisting on only white connectors (1). The right-hand flange of the shaft is un-connected and under-determined (2). A model that is over-constrained through the Rotational part (3).

since the Rotational part is forming a loop. This of course could be handled by the rule above. It would require that there were four Rotational3D connectors to cover the combinations of constrained and unconstrained connectors.

Still, however, this would not be sufficient for the general case since there are cases where some of the MultiBody variables are constrained and some not, which ultimately would lead to an unreasonable amount of connectors.

3 Library contents

Unlike for many other libraries, Rotational3D components are not intended to be used entirely on its own, but together with Rotational and MultiBody components. Only parts that cannot be modelled in Rotational are implemented. This includes shafts, inertias, visualizers, and other models where either the geometry or inertial effects are considered. In addition, there are models that allow both Rotational and MultiBody models to be used with the library, especially mounts and housings.

Figure 3 shows some highlighted components: The MultiBodyMount (1) translates a Rotational3D connector to a rotating MultiBody frame and the RotationalMount (2) allow Rotational models with action and reaction torques to be connected. The SupportedRotationalMount (3) is a version of (2) including a bearing and thus has an additional MultiBody connector to define the orientation of the flange. The Flange visualization (4) visualizes the vectors n and q as well as the rotation angle ϕ as seen in Figure 1 which is useful both for debugging and comprehension. The Shaft component (5) defines the motion between two flanges without reference. It imposes no constraints between

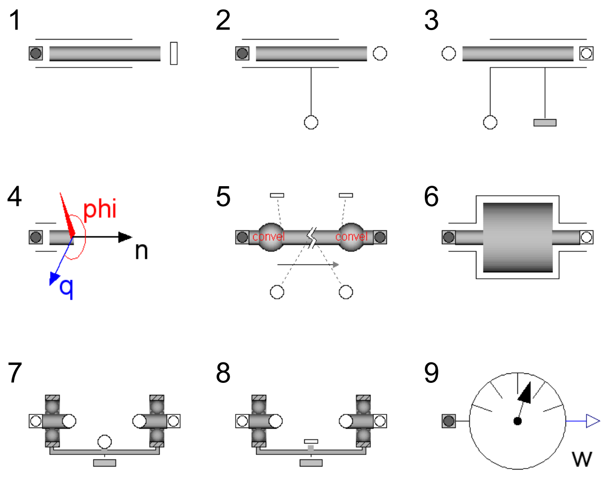


Figure 3: A selection of components from Rotational3D.

the flanges since it is assumed to be flexible also in the length direction. The twist characteristics is defined by connecting components to the Rotational connectors and additionally, inertial effects can be added via the MultiBody connectors. The rotation angle is defined based on the choice of joint type made, either universal and constant velocity. The Inertia (6) corresponds to the Rotor component but additionally contains geometric information as well as mass properties and the corresponding visualization.

As an extension to the basic mounts (1-3), there are also housings suitable for components such as gear boxes. The SupportedHousing (7) allow for ideal gears to be directly connected to the Rotational flanges. The SupportedHousing2 (8) is an adaption to handle gearboxes from e.g. the PowerTrain library [8] requiring a MultiBody support frame for internal inertias and animation. There are also sensors, such as the AbsoluteRotationalVelocity (9), described in Section 2.2.

4 Examples

As explained earlier, Rotational3D is intended to be used together with both Rotational and MultiBody components. This section explains and exemplifies when Rotational3D is suitable and how component and subsystem models can be designed.

4.1 Bevel gear

The first example shows two bevel gear components. Consider first the upper diagram layer in Figure 4, showing a bevel gear with an external support frame (1). To this frame, two SupportedRotationalMount components (2) are connected, each containing a bearing that specifies the orientations of the two axles. These mounts are connected to a Rotational.GearR2R component (3) that contains the 1D gear characteristics. Each mount is also connected to a connector (4) and visualization for gear wheel and axle (5). Both connectors are white since their orientation is fully specified and as seen in the animation view (6), there are two bearings except for the axles and gear wheels.

Now consider the lower diagram, showing the same bevel gear but without the support frame. Instead, the left connector is grey, indicating that this component has to be connected to an outside bearing. The left mount is here a RotationalMount, and as a consequence, no bearing is present in the visualization.

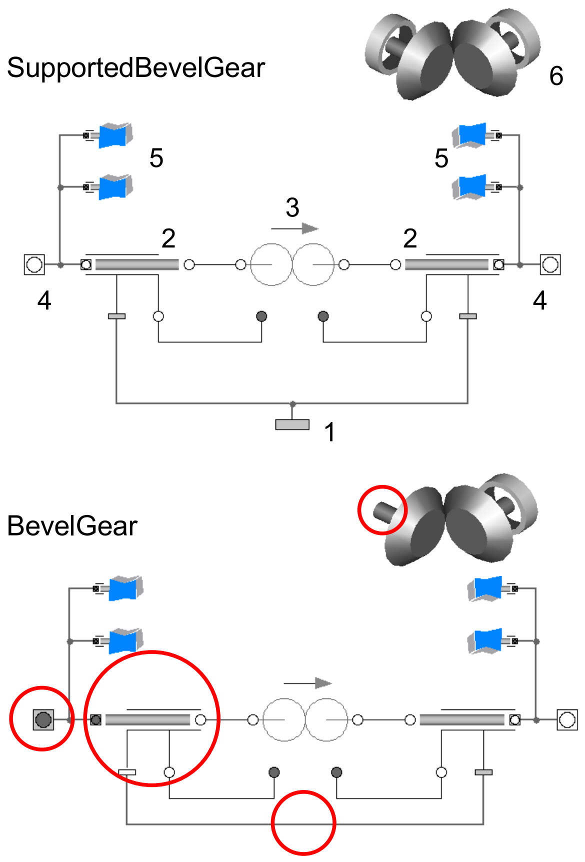


Figure 4: Diagram layer and visualization of two bevel gear models.

4.2 Steering system

Steering systems for passenger cars are typically designed as illustrated in Figure 5. The steering wheel is attached to the steering column (1) which in turn is connected to a shaft that is connected to the pinion of the rack-and-pinion mechanism (3). Each connection requires a joint (4) with two degrees of freedom to allow the shaft to rotate around its length axis. The universal joint is the most common type, consisting of two revolute joints in series forming a cross. Unlike more advanced joints, the universal joint has a varying ratio over a revolution, depending on the bend angle (α). This require careful design of the geometric layout as well as the rotation of the cross and depending on these two factors, the resulting gear ratio of the mechanism will vary. Figure 6 shows the difference in rack

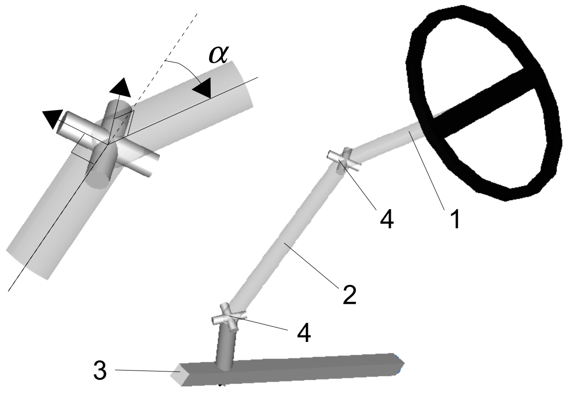


Figure 5: Typical steering system layout (right) and universal joint geometry (left).

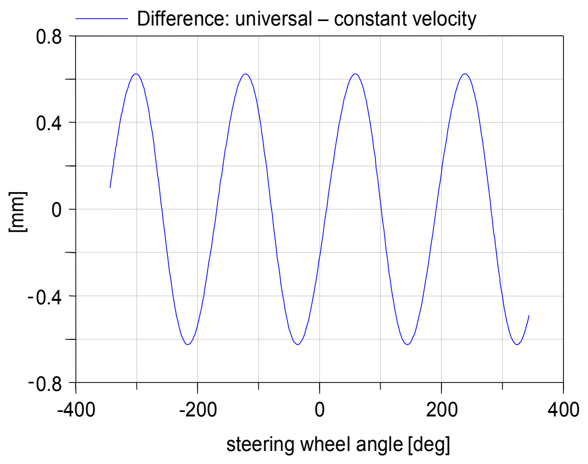


Figure 6: Difference in rack position as function of steering wheel angle.

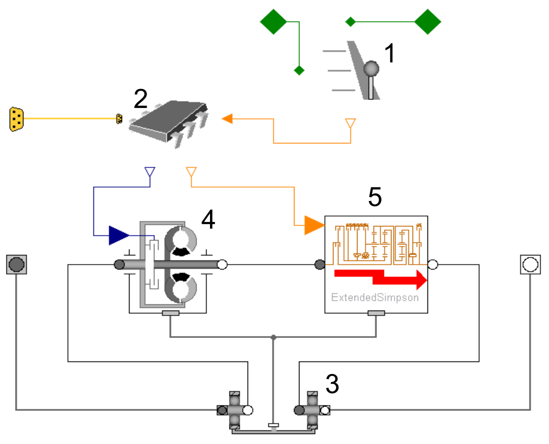


Figure 7: Transmission model for the VehicleDynamics library using PowerTrain library components.

position for the steering system in Figure 5, compared to the same system but with constant velocity joints.

4.3 Automatic Transmission

Another advantage of Rotational3D is its compatibility with other Modelica libraries. Figure 7 shows an example of a seamless integration of two libraries for an automatic transmission. It is defined from a standard interface in the VehicleDynamics library and completed with shift mechanism (1), shift controller (2) and a housing (3). The torque converter (4) and the gear box (5) are from the PowerTrain library [8]. Note that the PowerTrain components use a MultiBody support frame as discussed in Section 3.

Table 1: Translation and simulation results from Dymola [9].

	MB	Rt3D
linear systems	$\{10,3,3,2\}$	$\{3,3\}$
non-linear systems	$\{10\}$	$\{1\}$
simulation time	1.91	0.047

4.4 Comparison with MultiBody

Two implementations of a drive shaft are compared. A drive shaft is typically used to transfer axial rotational motion between two moving bodies. Using MultiBody primitives to define a non-elastic shaft with universal joints yields the series `R-U-P-U-R`. `R` denotes a revolute joint, here representing the bearings, `P` is a prismatic and `U` is a universal (two orthogonal revolute joints).

Figure 9, shows an animation screen shot of the performed simulation where both the MultiBody (MB) and the Rotational3D (Rt3D) representations are included on top of each other. The trajectory of the first universal joint is shown as well as vectors for the resulting forces and torques. The first shaft end is driven with a constant speed relative to its bearing while the required drive torque as well as the speed at the other shaft end is shown, Figure 9.

Table 1 shows the number of linear and non-linear systems of equations and the simulation time for the two cases described above. By avoiding the constraints imposed by the MultiBody primitives, the number of non-linear equations are drastically decreased which makes the Rotational3D implementation about a factor 40 faster and less sensitive to the specified accuracy of the integrator.

5 Limitations

Although the library concept is proven to be efficient, it requires that some fundamental rules are followed. As already stated, there is a connection rule to avoid over- and under-constrained models. Additional, relating to the problems with the representation of multiple revolution in MultiBody, it is required that the relative rotation between two reference frames is less than one revolution.

6 Conclusions

This work presents a new library that combines the advantage with Rotational and MultiBody representation

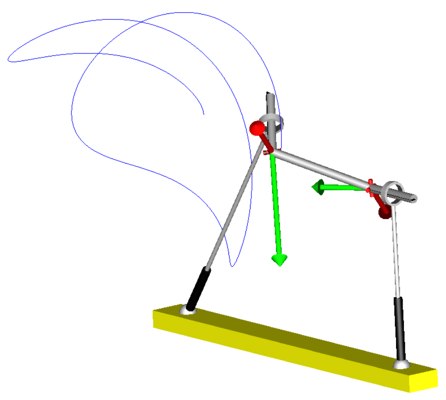
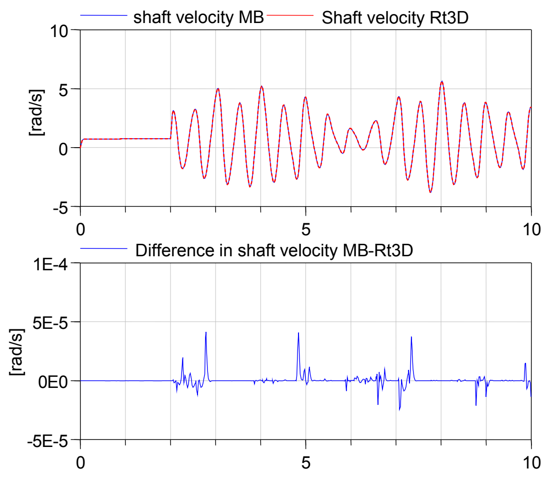


Figure 8: Animation view of the comparison and validation example with motion, forces and torques indicated. The MultiBody and the Rotational3D models are overlayed.



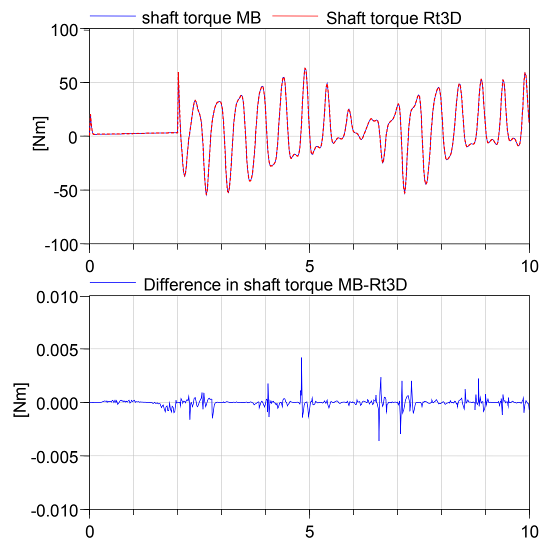


Figure 9: Comparison of the MultiBody and the Rotational3D implementations. Speed at the second shaft end (upper two plots) and required drive torque at first shaft end (lower two plots).

of rotational mechanics. It gives the same simulation results as MultiBody representations, often at a fraction of the cost. An example shows a factor of around 40. More complex mechanisms such as drivelines are often difficult/impossible to get to work in MultiBody. These are built seamlessly in Rotational3D.

References

[1] Andreasson, J., Gävert, M., The VehicleDynamics Library - Overview and Applications. In: *Proceedings of the 5th Modelica Conference*, Vienna, Austria, Modelica Association, 4-5 September 2006.

[2] Andreasson, J., On Generic Vehicle Motion Modelling and Control, Ph.D. thesis, ISBN 91-7178-527-2, 2006.

[3] Modelica Association. The Modelica Standard Library, http://www.modelica.org

[4] Otter, M. et al. The New Modelica MultiBody Library. In: Proceedings of the 3rd Modelica Conference 2003, Linköping, Sweden, Modelica Association, October 2003.

[5] Schweiger, C., Otter, M., Modelling 3D Mechancisl Effects of 1D Powertrains. In: Proceedings of the 3rd Modelica Conference 2003, Linköping, Sweden, Modelica Association, October 2003.

[6] Modelon AB, Lund, Sweden. The VehicleDynamics library, User's Guide, Version 1.2, 2007.

[7] Dempsey M. et al. Coordinated automotive libraries for vehicle system modelling. In: Proceedings of the 5th Modelica Conference 2006, Vienna, Austria, Modelica Association, September 2006.

[8] Schweiger, C. et al. The PowerTrain Library: New Concepts and New Fields of Application. In: Proceedings of the 4th Modelica Conference 2005, Hamburg, Germany, Modelica Association, March 2005.

[9] Dymola - Dynamic Modelica Laboratory, http://www.dynasim.se

Methods of Sensitivity Calculation Applied to a Multi-Axial Test Rig for Elastomer Bushings

Susann Wolf Joachim Haase Christoph Clauß
Fraunhofer-Institute for Integrated Circuits, Design Automation Division (IIS/EAS)
Zeunerstraße 38, 01069 Dresden, Germany
{Susann.Wolf, Joachim.Haase, Christoph.Clauss}@eas.iis.fraunhofer.de

Michael Jöckel Jürgen Lösch
Fraunhofer-Institute for Structural Durability and System Reliability (LBF)
Bartningstraße 47, 64289 Darmstadt, Germany
{Michael.Joeckel, Juergen.Loesch}@lbf.fraunhofer.de

Abstract

This paper presents methods of sensitivity calculation applied to a multi-axial test rig for elastomer bushings. In this context, the effect of parameter variations on system variables is analysed by using different tools. Aspects like modelling, modelling effort, computing time and accuracy are discussed.

This paper is based on results, which were developed in the Fraunhofer collaborative project "Computer Aided Robust Design (CAROD)".

Keywords: parameter sensitivity; Monte-Carlo simulation; elastomer test rig; DAE; Dymola; DASPK; MATLAB

1 Introduction

Tolerances of material quality, manufacturing processes and assembly operations lead to scattering product properties. In mass production therefore more or less significant deviations of the desired "ideal" properties occur. Resulting from wear and degradation, also during product life cycle continuous changes of component and system characteristics take place.

The named uncertainties are only covered insufficiently in traditional development workflows of mechatronical products. In most cases, simulations without any parameter scatter are performed to optimise mechanical and mechatronical systems and to analyse their durability and reliability.

The objective of Robust Design is to analyse the effects of scattering component behaviour in early development phases and to optimize products. This presentation will focus on sensitivity analysis, which typically is an initial task in robust design studies. The prior objective is to determine parameters, which highly affect the product behaviour.

The paper starts with a short description of the chosen technical example: A 3 DOF test rig for elastomer bushings. Subsequently, two multi-body-simulation models of the test rig using MATLAB and Modelica are presented, including a cross-comparison of the particular simulation results. In the following, the method of "sensitivity calculation" is introduced, which has conceivable potentials to lower the effort for sensitivity analyses. To validate the results, the implementation of a Monte-Carlo (MC) analysis is treated, which is based on repeated calls of the Dymola simulator. The results of this MC analysis are again compared with an MC analysis performed with MATLAB. Finally, based on the analysis results, the application and performance of the methods for sensitivity evaluation are discussed.

2 Technical Example: Test Rig for Elastomer Bushings

2.1 Application and Technical Description

The methods described in this paper are presented by example of a multi-axial test rig (figures 1 and 2), which is owned by Fraunhofer LBF. The test rig is mainly used for sign-off tests of automotive elastomer bushings considering service loads. Further on, the bushing's dynamic transmission behaviour can be characterised.

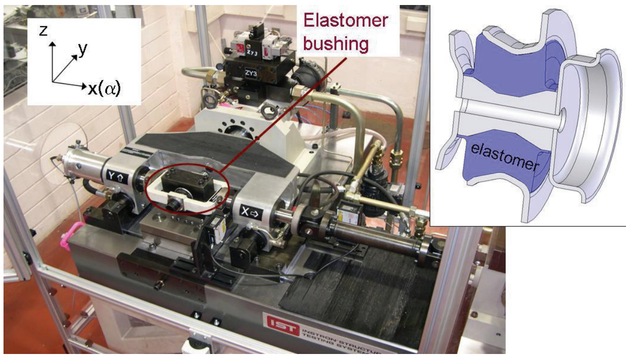


Fig. 1: Multi-axial test rig for characterisation and testing of elastomer bushings at Fraunhofer LBF and CAD model of a typical elastomer bushing (right)

Using servo-hydraulic actuation, variable cyclic loading of more than 100 Hz is feasible. Based on a transmission design with pre-stressed cardan joints, the load directions “axial (x)”, “lateral (y)” and “torsion (α)” at the bolt can be realised isolated or in combination, respectively. To minimise wear and friction, hydrostatic linear guides between bolt and bail as well as bail and base plate are used. The movement of the bolt – and therefore the displacement of the elastomer bushing – is measured by sensors, which are positioned near the bushing’s clamping device. The reaction forces are measured by a piezoelectric measurement platform.

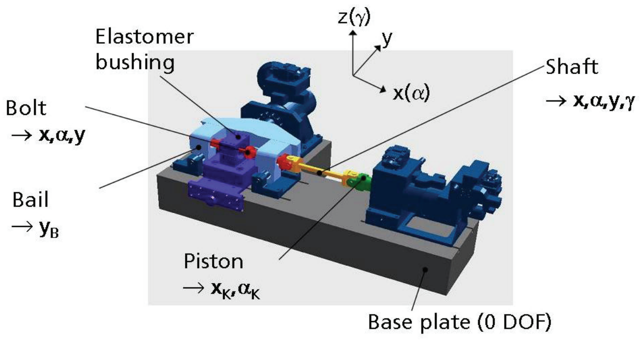


Fig.2: CAD model of the test rig, declaration of bodies and degrees of freedom (“absolute”)

2.2 Multi-Body-Simulation Model of the Test Rig and Simulation Tools

To analyse the dynamic behaviour of test rig mechanics, multi-body-simulations (MBS) are used (cp. [1]). In this context, the models typically include rigid bodies, which are linked by joints and force elements.

Corresponding, the MBS model of the elastomer test rig includes the rigid bodies “bolt”, “bail”, “piston” and “shaft”, while following joints and force elements are applied between the bodies (cp. figures 2 and 3):

- Bolt-bail: cylindrical joint x/α, damping x/α (hydrostatic bearings)
- Bail-base plate: prismatic joint y, damping y (hydrostatic bearing and friction of hydraulic cylinder)
- Bolt-base plate: 6 DOF, stiffness $x/\alpha/y$ and damping $x/\alpha/y$ (elastomer bushing)
- Bolt-shaft: cardan joint, no force element
- Shaft-piston: cardan joint, no force element
- Piston-base plate: cylindrical joint x/α, damping x/α (friction of hydraulic cylinder)

The transmission characteristic of the elastomer bushing is modelled by using the approaches

$$F_{E,x}=c_{Ex3}\cdot x^3+c_{Ex1}\cdot x+d_{Ex1}\cdot dx/dt,$$

$$F_{E,y}=c_{Ey3}\cdot y^3+c_{Ey1}\cdot y+d_{Ey1}\cdot dy/dt \text{ and}$$

$$M_{E,\alpha}=c_{E\alpha3}\cdot\alpha^3+c_{E\alpha1}\cdot\alpha+d_{E\alpha1}\cdot d\alpha/dt,$$

which describe nonlinear stiffness and linear damping behaviour for each load component. The parameter settings are based on measurement data derived by tests with a commercial elastomer bushing.

Comparable to the physical test rig, the MBS model is actuated by axial forces F_x, torsion M_α (piston) and lateral forces F_y (bail).

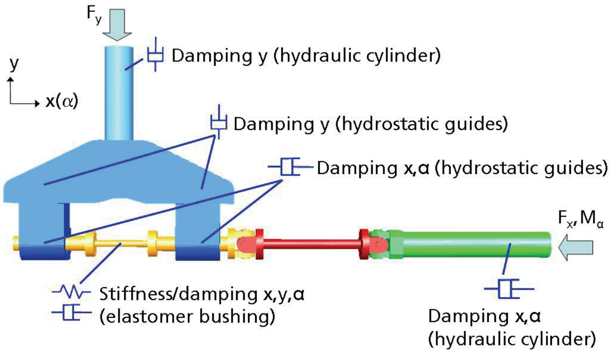

F_y
Damping y (hydraulic cylinder)
y
x(α)
Damping y (hydrostatic guides)
Damping x,α (hydrostatic guides)
F_x, M_α
Stiffness/damping x,y,α (elastomer bushing)
Damping x,α (hydraulic cylinder)

Fig.3: Multi-body simulation model of the test rig („top view“)

The MBS model was set up in MATLAB/SimMechanics as well as in Modelica (using a Dymola solver).

2.3 Comparison of Modelica and MATLAB Model

To compare the MBS models built in the MATLAB and Modelica environment, two test cases were defined. The first test case is characterised by sinusoidal forces and moments:

- $F_x=3.0\cdot10^6\cdot\sin(314\cdot t)$ [N]
- $F_y=3.4\cdot10^8\cdot\sin(t)$ [N]
- $M_\alpha=10^4\cdot\sin(150\cdot t)$ [Nm]

In contrast, the second test case includes the application of noise signals (figure 4). These signals were generated with a MATLAB script, which contains the rand command.

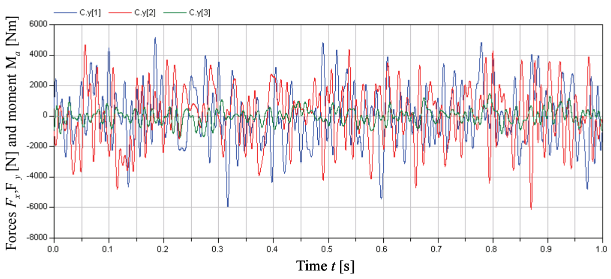


Fig. 4: Applied noise signals in test case 2

Figures 5 and 6 show the calculated displacements x and y [mm] as well as the torsion α [rad] of the body "bolt" for the two test cases.

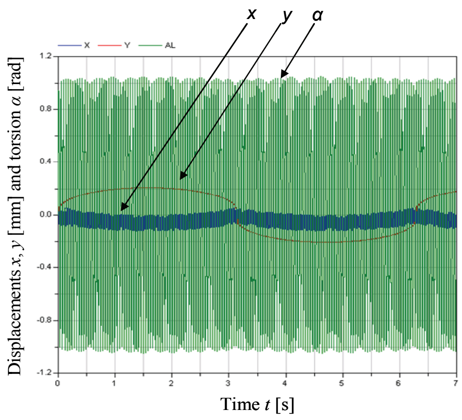


Fig. 5: Test case 1 - Solution (Dymola)

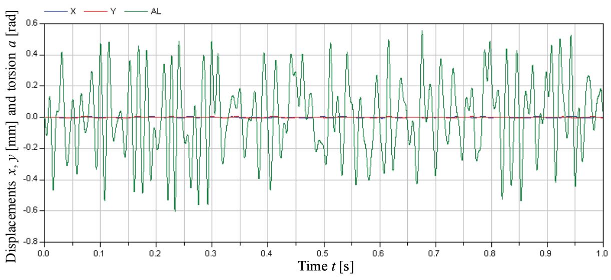


Fig. 6: Test case 2 – Solution (Dymola)

To compare the MBS models created in Modelica and MATLAB, the displacements x and y as well as the torsion α of the body "bolt" were analysed in the time domain. Figure 7 shows the calculated displacements and torsion for test case 1, exemplarily. It is obvious, that both models lead to nearly identical results. This conclusion is affirmed by the results concerning case 2.

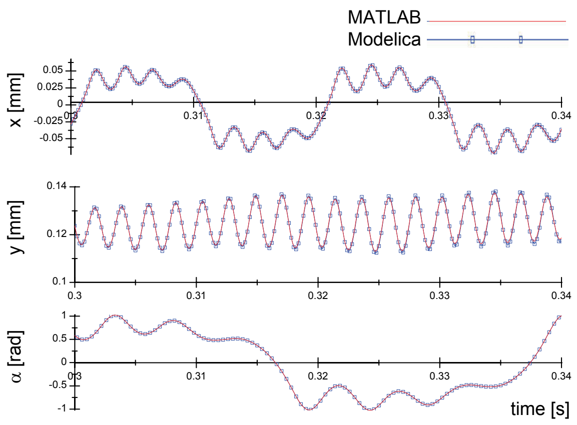


Fig. 7: Comparison of results from Modelica and MATLAB model (case 1)

2.4 Treated Scenario

The investigations target a virtual scenario, which describes the production of a small series of (only) theoretically identical test rigs. Due to manufacturing tolerances the test rig components will differ more or less, leading to scatter of the test rig's behaviour.

To examine the scenario, prior parameters of the MBS model have to be defined, which are affected by assumable manufacturing tolerances. In this context, 13 parameters have been identified:

- Masses of all rigid bodies
- Inertias "α" of bodies "piston", "shaft" and "bolt"
- Inertia "γ" of body "shaft"
- Damping coefficients concerning hydrostatic linear guides "bail-base plate (y)" and "bail-bolt (x,α)"
- Damping coefficients concerning friction of hydraulic cylinders ("piston": x/α, "bail": y)

In the following, the sensitivity of the test rig dynamics on variations of these parameters is examined by sensitivity calculation and MC analyses.

3 Sensitivity Calculation

3.1 Method of Sensitivity Calculation

The main idea is to pre-evaluate the sensitivity of the test rig performance due to variations of single parameters. Beside information concerning the performance scatter to be expected, promising "adjusting screws" for system optimisation can be derived.

The performance and dynamics of the test rig are analysed using multi-body-simulation. To perform sensitivity calculation, the analytical equations of motion have been set up explicitly using the Lagrange approach. Thus, the system equations are available in symbolic form. The DAE/ODE system of the elastomer test rig is described by 3 equations of motion (1) to (3) with 3 state variables *x(t)*, *y(t)* and $\alpha(t)$ as well as 23 system parameters (13 parameters to be varied, 10 fixed parameters). The system is excited by sinusoidal forces and moments (compare (1) to (3), case 1) as well as noise signals (case 2).

$$\begin{aligned}
3.0e6\,\mathrm{Sin}(314\,t) &= cEx1\,lW + 4\,cEx3\,lW^3 + cEx1\,X(t) + 6\,cEx3\,lW^2X(t) + 3\,cEx3\,lW\,X(t)^2 + \\
&cEx3\,X(t)^3 - 3\,cEx3\,lW\,Y(t)^2 - 3\,cEx3\,X(t)\,Y(t)^2 - \frac{cEx1\,lW^2}{\sqrt{lW^2-Y(t)^2}} - \frac{4\,cEx3\,lW^4}{\sqrt{lW^2-Y(t)^2}} - \frac{6\,cEx3\,lW^3X(t)}{\sqrt{lW^2-Y(t)^2}} - \qquad (1)\\
&\frac{3\,cEx3\,lW^2X(t)^2}{\sqrt{lW^2-Y(t)^2}} + \frac{cEx1\,Y(t)^2}{\sqrt{lW^2-Y(t)^2}} + \frac{5\,cEx3\,lW^2Y(t)^2}{\sqrt{lW^2-Y(t)^2}} + \frac{6\,cEx3\,lW\,X(t)\,Y(t)^2}{\sqrt{lW^2-Y(t)^2}} + \\
&\frac{3\,cEx3\,X(t)^2Y(t)^2}{\sqrt{lW^2-Y(t)^2}} - \frac{cEx3\,Y(t)^4}{\sqrt{lW^2-Y(t)^2}} + (dDx+dEx1+dKx)\,X'(t) + \frac{(dDx+dEx1)\,Y(t)\,Y'(t)}{\sqrt{lW^2-Y(t)^2}} + \\
&\frac{lW^2(2\,mD+mW)\,Y'(t)^2}{2\sqrt{\left(lW^2-Y(t)^2\right)^3}} + (mD+mK+mW)\,X''(t) + \frac{(2\,mD+mW)\,Y(t)\,Y''(t)}{2\sqrt{lW^2-Y(t)^2}}
\end{aligned}$$

$$3.4e8\,\mathrm{Sin}(t) = cEy1\;Y(t) + \ldots + \left(\begin{array}{l} mB+mD+\dfrac{mW}{4}+\dfrac{jWz}{lW^2-Y(t)^2}+ \\ Y(t)^2\left(\dfrac{4\,mD+mW}{4\,lW^2-4\,Y(t)^2}+\dfrac{jWx\,lW^2\;\mathrm{Tan}(AL(t))^2}{\left(lW^2-Y(t)^2\right)\left(\dfrac{-lW^2}{\mathrm{Cos}(AL(t))^2}+Y(t)^2\right)^2}\right)\end{array}\right)Y''(t) \qquad (2)$$

$$1.0e4\,\mathrm{Sin}(150\,t) = (cEa1+cEa3\;AL(t)^2)\;AL(t)\;+\ldots+\frac{jWx\,lW^2\;\mathrm{Tan}(AL(t))\;Y(t)\;Y''(t)}{\mathrm{Cos}(AL(t))^2\left(\dfrac{-lW^2}{\mathrm{Cos}(AL(t))^2}+Y(t)^2\right)^2} \qquad (3)$$

The performance and dynamics of the test rig are evaluated by analysing the resulting displacements *x* and *y* as well as the resulting torsion α of the body "bolt", which directly loads the elastomer bushing.

Sensitivity calculation examines the effects of minor parameter deviations from their nominal values for the behaviour of the dynamic systems.

In this case systems are regarded, which are described by differential-algebraic equations (DAEs) of the form

$$F(x, \dot{x}, p, t) = 0 \qquad (4)$$

, the equations of motion of the system, whereas $x(t) \in \mathbb{R}^n$ are state variables and $p \in \mathbb{R}^m$ summarises parameters of all types, which mean diverse determining factors on developing of the variables.

The factor of interest is the influence, which changes of parameters cause of developing of the variables, the so-called *sensitivities of parameter* $\dfrac{\partial x(t)}{\partial p_i}$ (the sensitivity of variable *x* concerning parameter p_i). These sensitivities are computed for the evaluation of the interesting influence. Sensitivities are functions of time *t*. Also these functions can be used as a basis for the determination of derived functions.

By differentiation of the system (4) according to all p_i the following system can be set up for the computation of first-order parameter sensitivities:

$$\frac{\partial F}{\partial x}\frac{\partial x}{\partial p_i}+\frac{\partial F}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p_i} = -\frac{\partial F}{\partial p_i} \qquad (i=1,...,m) \qquad (5)$$

(4) together with (5) can currently be solved by Dymola, if (5) is added explicitly. The code DASPK [2] can directly solve the DAE (4) and (5), where (5) is generated within the code automatically.

3.2 Results of Sensitivity Calculations

Dymola and DASPK were used to calculate both solution and sensitivities of the elastomer test rig. For the equations of motion the results for *x*, *y* and α were calculated. The solutions from Dymola and DASPK are in accordance. In the following, the influence of parameters on the axial displacement *x* and the torsion α of the bolt are illustrated.

The first-order sensitivities for case 1 are computed using DASPK (figures 8 to 15). The solutions are shown in 8 and 10 (see also figure 5 using Dymola). The figure 9 shows first-order parameter sensitivities of *x* regarding parameters of mass. The timeline corresponds to figure 8. Figure 11 where the timeline corresponds to figure 10 shows the first-order parameter sensitivities regarding parameters of inertia.

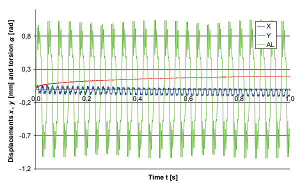


Fig.8: Detail of solution (DASPK)

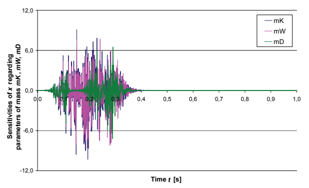


Fig.9: Detail of first-order parameter sensitivities of *x* (DASPK)

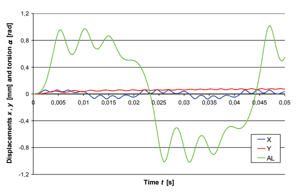


Fig.10: Detail of solution (DASPK)

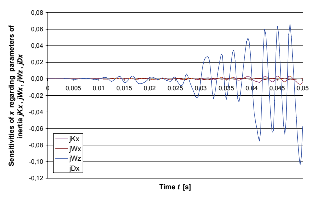


Fig.11: Detail of first-order parameter sensitivities of *x* (DASPK)

By classifying the amplitudes of the time-depending sensitivity functions (e.g. figures, 9, 11, 12, 13, 14, 17 and 18) it can be evaluated, which parameters have a large, a marginal or no influence on the solution of the system. Parameters with a significant influence to solution *x* are the parameters of mass *mK*, *mW*, *mD* and the parameters of inertia *jKx, jWz* and *jDx*, see as an example figure 12 and 13. Concerning sensitivities of *y,* also parameters of inertia and mass have an influence. Parameters with a significant influence to solution *α* are the inertias *jKx*, *jWz* and *jDx*.

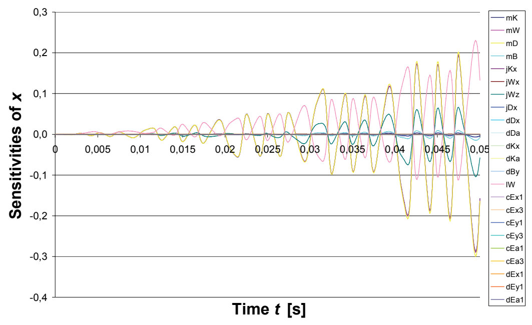


Fig.12: Detail of first-order parameter sensitivities of *x* regarding all parameters (DASPK)

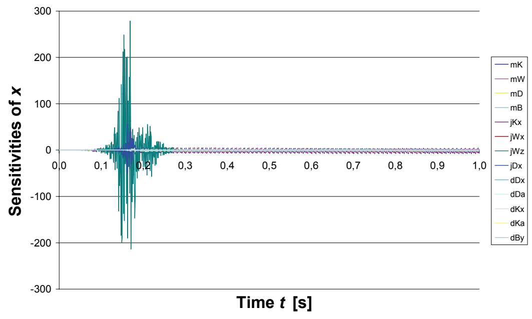


Fig.13: Detail of first-order parameter sensitivities of *x* regarding 13 specified parameters) (DASPK)

Again figure 14 illustrates the described results exemplarily of parameter of mass *mK* regarding *x* and *α*.

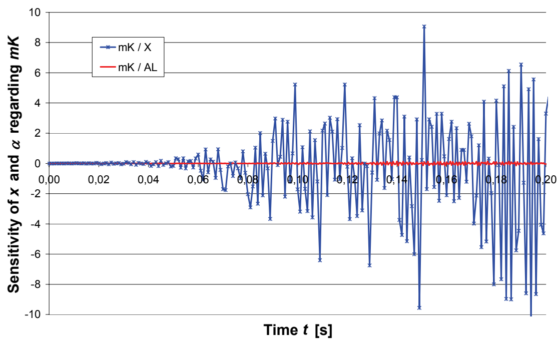


Fig.14: Comparison of influence of parameter of mass *mK* on *x* and *α*

Solutions of the original system using different parameter values confirm the sensitivity calculations. Figure 15 illustrates the range of tolerance of solution *x* regarding variation of all parameters (± 0,5% and ± 1%). The graphic shows that by increasing time the range of tolerance band is increasing. That is why parameter changing causes not only different amplitudes but also variations in the time behaviour.

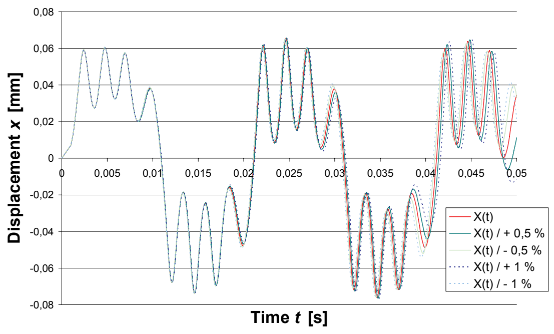


Fig.15: Detail of range of tolerance of solution *x* regarding variation of all parameters (DASPK)

In case 2 (figures 17 and 18), sensitivity calculations are carried out by means of Dymola. The resulting first-order parameter sensitivities are approximations by method of difference quotient. This method is explained by a scheme which is illustrated in figure 16. The original model was parallel instantiated with different parameter values, which are necessary for difference quotient calculation.

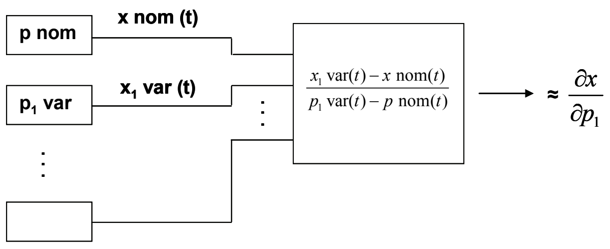

p nom
x nom (t)
p_1 var
x_1 var (t)
$\frac{x_1\,\mathrm{var}(t) - x\,\mathrm{nom}(t)}{p_1\,\mathrm{var}(t) - p\,\mathrm{nom}(t)}$
$\approx \frac{\partial x}{\partial p_1}$

Fig.16: Method of difference quotient using Dymola

Figure 17 shows that all considered parameters have nearly no influence on the solution *x* of the system, whereas parameters of mass *mK*, *mW* and *mD* are dominating parameters.

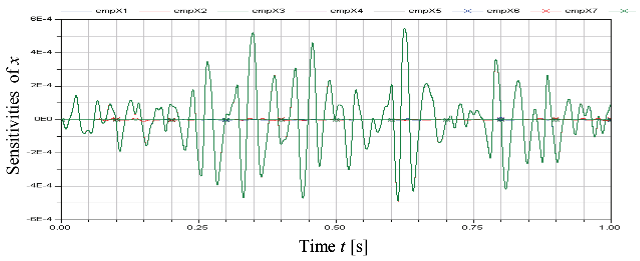


Fig.17: First-order parameter sensitivities of *x* for 13 specified parameters (difference quotient)

Parameter with a significant influence on solution *α* are parameter of inertia *jKx*, *jWx* and *jDx*, see figure 18. Regarding case 2, first-order parameter sensitivities will also be carried out by means of DASPK.

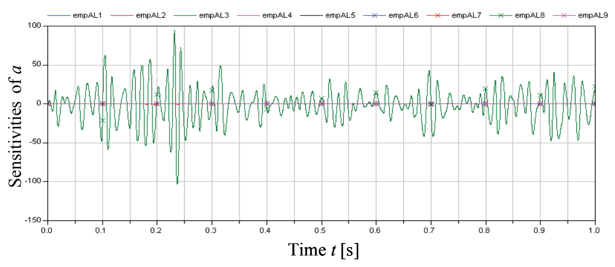


Fig.18: First-order parameter sensitivities of α for 13 specified parameters (difference quotient)

The results show, that sensitivity calculation leads to comparable results for test case 1 and test case 2.

Related to these sensitivity computations it can be summarised that additionally to the determination of the solution of a concrete DAE system also sensitivity computations are possible. As noted above, the results from Dymola and DASPK are in accordance. Concerning the results of sensitivity calculations, the evaluation of influence of the parameters on the targeted result values is feasible.

4 Monte-Carlo Analysis

To evaluate the results of sensitivity calculation MC analyses [3] were performed by usage of both Modelica and MATLAB models.

4.1 Modelica Model

Within the Dymola simulator MC simulation is offered as a tool-specific feature. In this paper, another possibility was used which will also be presented as a poster on the Modelica'2008 conference.

In the Modelica model the parameters which are to be varied randomly get their values via a function call. This function, which can be coded as a Modelica function or a C-Function, is parametrised by the user with parameters of the desired random distribution. Repeated Dymola calls (via the scripting language) cause the randomly choice of the value of the chosen parameter. The results of each simulator run have to be collected.

In this case, a uniform distribution with the nominal value 6.0 and the tolerance ±10% was used as an example, which is specified in the model in this way:

```
model elast
 // parameter of the system
    parameter Real mK = uniform(6,0.10)
    "mass of piston";
...
```

Sensitivity calculation described in section 3 yielded an evaluation of system parameters. Then MC analyses for located dominating parameters were determined.

The results of MC analyses for case 1 are illustrated by figures 19 to 23.

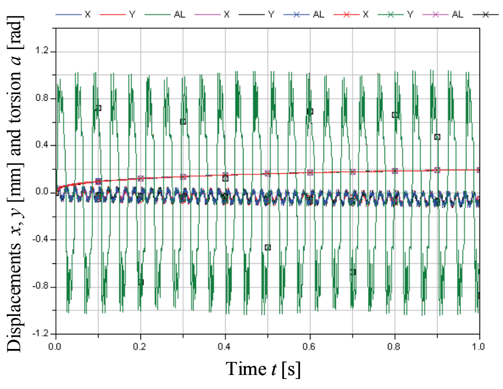


Fig.19: Tolerance band of x, y and α regarding parameters of mass

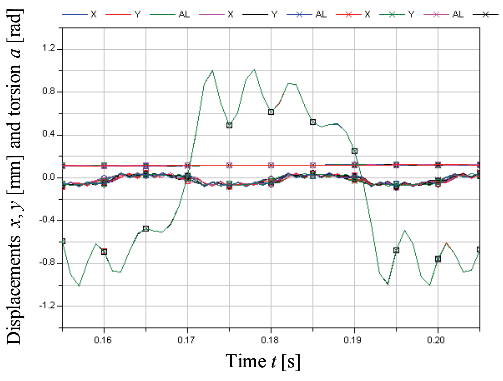


Fig.20: Detail of tolerance band of x, y and α regarding parameters of mass

Figure 21 shows that parameters of mass have only a marginal influence on solution α. These result verified the small tolerance band of α regarding parameters of mass.

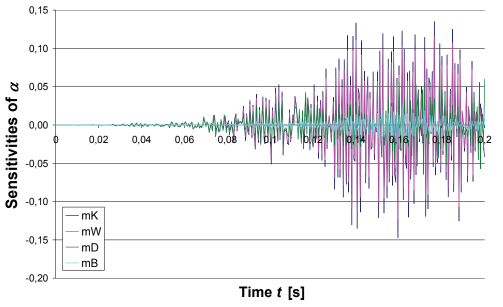


Fig.21: Sensitivity of α regarding parameters of mass (DASPK)

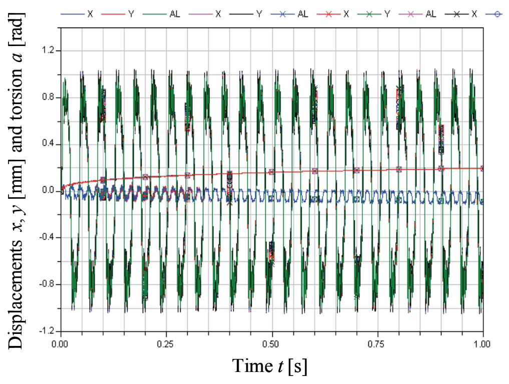


Fig.22: Tolerance band of x, y and α regarding parameters of inertia

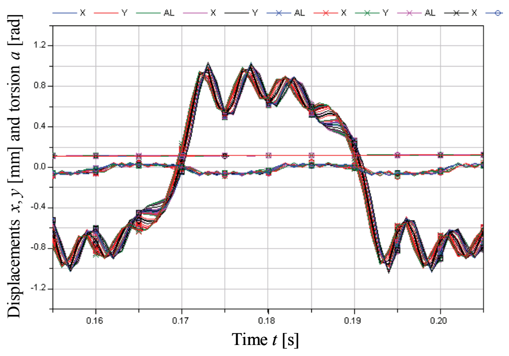


Fig.23: Detail of tolerance band of x, y and α regarding parameters of inertia

The results for case 2 are shown by figures 24 to 26.

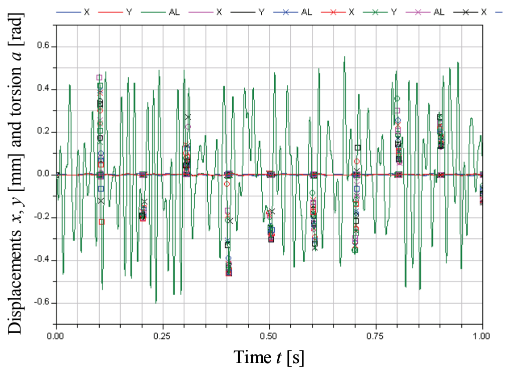


Fig.24: Tolerance band of x, y and α regarding parameters of mass

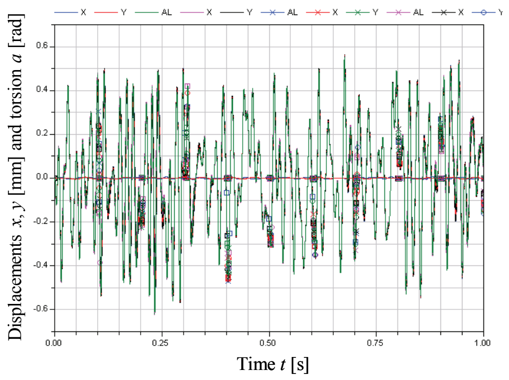


Fig.25: Tolerance band of x, y and α regarding parameters of inertia

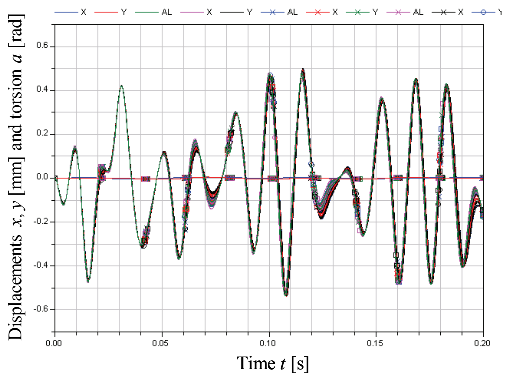


Fig.26: Detail of tolerance band of x, y and α regarding parameters of inertia

MC analyses verified the results of sensitivity calculation, see section 3.

4.2 MATLAB Model

The sensitivity analysis of the MATLAB model was performed with the Fraunhofer LBF inhouse software MASIMO. The software creates sample sets of user-defined parameters based on Latin-Hypercube-Sampling methods and automatically performs the needed simulations in MATLAB. MASIMO was, among other things, applied during the EC funded project "MODBOGIE" [4] to perform sensitivity analysis of a complex locomotive model.

The MC analyses each contained 100 simulations for test case 1 and test case 2. All 13 parameters (cp. ch. 2.2) were set to vary in a range of ±10% of their nominal value, while an equal distribution of parameter values was defined, respectively.

To analyse the resulting time series of the simulations, scalar evaluation quantities x_m, y_m and α_m were defined, taking the arithmetic mean value of the amount of displacements and torsions $x(t)$, $y(t)$ and $\alpha(t)$. Following, the parameter x_m is examined, exemplarily.

Figure 27 shows an qualitative Anthill plot of x_m as a function of the piston mass for test case 1. Each point represents the (converted) result of one single simulation of the MC analysis. The diagram shows the trend, that an increasing piston mass leads to decreasing values of x_m. In general, Anthill plots can be used to get a first impression of sensitivities and trends.

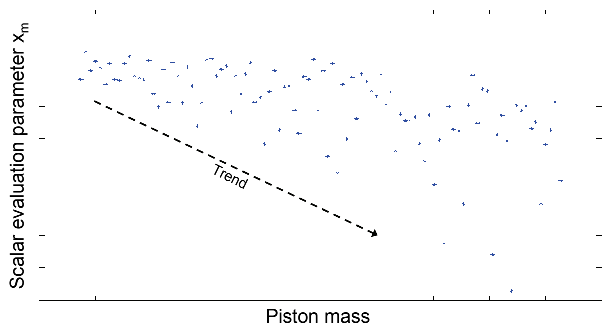


Figure 27: Anthill-plot of x_m as a function of piston mass

To derive further information, correlation coefficients between the result values x_m, y_m and α_m and the varied input parameters can be applied. Figure 28 exemplary shows the correlation between x_m and the input parameters. In this context, negative correlation coefficients point out, that an increasing parameter value leads to decreasing result quantities. It is obvious, that the masses of piston, shaft and bolt affect x_m dominantly.

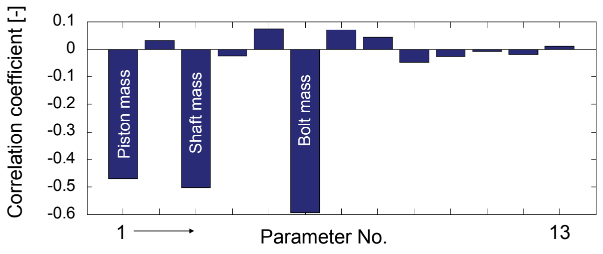


Figure 28: Correlation of x_m with the varied parameters (1…13), case 1

Similar investigations were performed for the result values x_m and α_m. The analysis of α_m showed prior sensitivities on the inertias of piston, shaft and bolt, while y_m is dominantly affected by the masses of all 4 bodies.

The analysis of test case 2 led to comparable results. Again, the body masses of piston, shaft and bolt affect x_m, while the torsion α_m is dominantly influenced by the inertias of these bodies. y_m again is dominantly affected by the masses of all 4 bodies.

5 Conclusions

In this paper, the application of sensitivity calculation was presented by example of a multi-body simulation model of an elastomer test rig. The results show, that sensitivity calculation has the potential to pre-evaluate prior parameters of a model, which exemplarily can be deeper analysed by a following MC analyses. An indispensable precondition for applying sensitivity calculation is the provision of the equations of motion in a symbolic representation.

Especially for complex models with a high number of DOF or long periods to be computed, the preselection of parameters can lead to a significant reduction of computational effort. Even in case of the test rig example, which only comprises 4 DOF and rather small time series to be computed (< 2 seconds), each simulation of the MC analysis took approximately 2 minutes (Pentium 4, 3 GHz). Resulting, a complete analysis with 100 simulations and 13 parameters took more than 3 hours.

A sensitivity calculation using DASPK respectively 13 parameters (until t_{end} 1 s) took approximately 10 minutes. Using Dymola a sensitivity calculation (as shown in figure 16) took maximal approximately 40 minutes and a MC analysis with 50 simulations and 4 varied parameters maximal approximately 12 minutes. Resulting, a complete MC analysis using Dymola with 100 simulations and 13 parameters (2 seconds) would take approximately 2.5 hours. These computation times point out, that sensitivity calculation is able to reduce effort considerably.

Within the Dymola simulator, MC simulation is offered as a tool-specific feature. In this paper a more common possibility was presented, which describes the MC method on the Modelica language totally. Both methods are very time consuming. Using Dymola the effects of parameter tolerances can be calculated by MC simulation (with a high computational effort), using the sensitivity system (5), which has to be added manually or using the finite difference approximation (see figure 16).

Using the code DASPK, system (5) is generated within the code automatically. It would be desirable, if this possibility (and also regarding a similar system for second-order parameter sensitivity) would be existant also in Dymola. So far, an operator of sensitivities like the existing operator *der (),* the derivation with respect to time, is absent.

Note, that the solver DASPK allows the computation of first-order parameter sensitivities. The interpretation of these results leads to a classification of the importance of the system parameters regarding the effect to the variables.

6 Outlook

The next steps will cover following topics:

Parameter Sensitivities of Second-Order using DASPK

In this section a short description is given to determine second-order parameter sensitivities by means of DASPK. By means of differentiation of the system (5) according to all p_i second-order parameter sensitivities are computed. As mentioned before, the code DASPK can solve the DAE (4) and (5). By differentiation of the system (4) according to all p_i and using of this system as a new system (4) in the source code, the second-order parameter sensitivities are generated automatically.

Another way to determine the second-order parameter sensitivities is the modification of source code of DASPK. Therefore, an aim is to extend the source code of DASPK to generate the system of second-order parameter sensitivities automatically by differentiation of system (5). Then DASPK could solve (4), (5) and also the system of second-order, where (5) and the system of second-order could be generated within the code automatically.

Introduction of scalar evaluation quantities

In continuative work, analyses concerning scalar evaluation quantities, which are derived from the results in the time domain, are planned. Examples for these scalar evaluation result quantities are the first eigenfrequency or the mean value of the amplitude spectrum in a defined frequency range. The first eigenfrequency can be computed in MATLAB directly from the condition matrix of the elastomer test rig model. Using Dymola, the condition matrix can be generated and denoted by linearisation of the original model. Within Modelica there are also matrix functions, which are useful for this context. The first eigenfrequency can be derived from simulations in the time domain and a following Fourier transformation. For each DOF x, y and α, then a scalar quantity can be calculated.

Regarding sensitivity calculation, problems can occur in this context, because scalar evaluation quantities are not directly available in the DAE system. If the evaluation quantity can be calculated during the simulation, sensitivities are automatically present. Otherwise, derived evaluation variables have to be calculated by post-processing. This challenge will be discussed in further publications.

References

[1] Jöckel, M.; Wallmichrath, M.; Bruder, M.; Lösch, J.; Landersheim, V.: Virtual Test Lab – Simulation Based Testing of Components and Systems. Proceedings of NAFEMS-Seminar 2006 "Virtual Testing – Simulation Methods as Integrated part of an Efficient Product Development", ISBN 1-874376-14-x, 2006

[2] Petzold, L.; Cao, Y.; Li, S.; Serban, R.: Sensitivity analysis of differential-algebraic equations and partial differential equations. Computers and Chemical Engineering 30, 2006, pp. 1553-1559.

[3] Robert, C. P.; Casella, G.: *Monte Carlo Statistical Methods*, Springer, 2004 (2nd ed.)

[4] Jöckel, M.; Bruder, T.; Lösch, J.; Kieninger, M.; Schmidt, H.: European research projects of LBF: HEMBOT / MODBOGIE. Proceedings of 4th Fraunhofer LBF Meeting on Structural Durability in Railway Engineering, May 16 and 17 2006, Fraunhofer LBF Darmstadt, 2006

Implementation of a Modelica Library for Simulation of High-Lift Drive Systems

Dipl.-Ing. Malte Pfennig Prof. Dr.-Ing. Frank Thielecke
Hamburg University of Technology, Institute of Aircraft Systems Engineering
Nesspriel 5, 21129 Hamburg

Abstract

The development and design of new high-lift drive systems is a complex and iterative process, which is often depending on experience. Especially results determined in the predevelopment phase and based on uncertain assumptions have decisive influence on the system specification and thus on the system design. In order to reduce development time and optimize the development process, a rapid generation and adaptation of simulation models for analysis of transient system behaviour is essential. This article presents an computer-integrated approach for further reduction of the high-lift development process. An interface to Modelia should enable an automated system model generation. A suitable component library is introduced and verified by simulation of the Airbus A380 flap actuation system.

The purpose of this article is to present the project of a computer-aided development process as well as an adequate component library for assembling simulation models of high-lift drive systems.

Keywords: high-lift system; power drive system; system development

1 Introduction

In order to reduce take-off and landing airspeed, modern transport aircraft are equipped with high-lift systems. The extension of slats and flaps at the wing's leading and trailing edges augments the effective wing area and also allows for higher angles of attack thus increasing the lift coefficient. Figure 1 depicts the high-lift surfaces, as well as the corresponding drive and actuation system.

A central power drive unit (PDU), mounted in the center fuselage, provides energy for driving a shaft transmission, which ensures mechanical synchronisation of the left and right actuation systems. The shaft transmission is routed across the wingspan by numerous bearings, while universal joints and gearboxes compensate changes in direction. Branch gears transmit the mechanical energy to rotary or ballscrew actuators which are coupled with the flap traverse mechanism. High actuator gear ratios reduce fast turning transmission inputs to slow panel movement.

As part of the secondary flight control, the high-lift drive system has to be fault-tolerant and fulfill high requirements regarding the reliability. While the power drive unit and the slat flap control computer are of redundant design, the shaft transmission system offers a single load path only. Sufficient mechanical strength of all elements in the actuation system is required for all possible system states. Peak loads occurring as a result of a system failure are often a design case for the mechanical components of the drive system. Thus, the analysis of transient system behaviour is of uttermost importance for the determination of strength requirements for the drive train's mechanical elements. As aerospace applications require certified components, no standard but custom-build components and assemblies have to be installed. In consequence, component parameters characterising their dynamical behaviour, e.g. the mass moment of inertia or the friction characteristics, are unknown in the early design phase. Thus, these parameters have to be estimated based on the knowledge of existing similar products.

Owing to numerous changes of the system architecture, requirements, constraints or parameters, the effort for installing and maintaining a complete simulation model in the early design and specification phase is not justified. For this reason, simplified models are used for a rough evaluation of peak loads, while adequate safety margins compensate uncertainties. However, increasing mechanical strength normally involves an increasing mass. Thus, considerable potentials in system weight reduction might be wasted.

In this report an integrated approach is presented that aims at an optimisation of the high-lift drive system, as well as its development process. Moreover, an au-

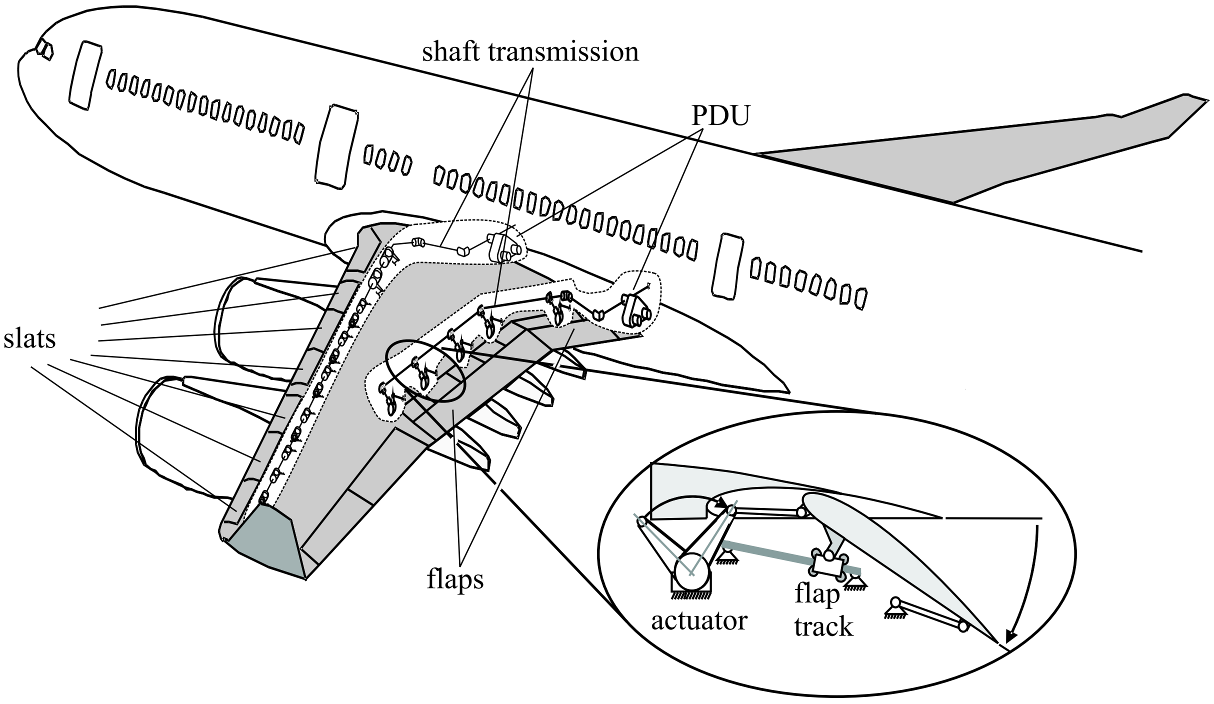


Figure 1: High-lift segments and power drive train at leading and trailing edge

tomated generation and easier maintenance of a complex simulation model for analysis of transient system behaviour should be realised in order to make simulation results available in the predevelopment phase. A software tool combining knowledge based methods for high-lift design and steady state calculations is to be extended to transfer available system information into a simulation model for analysis of transient behaviour. Modelica's characteristic of being object-oriented and providing a simple way to generate simulation models by combining library components makes it predestined for this task. In order to facilitate modelling a complex high-lift drive train, a library containing all required components has been created.

2 System description and modelling

The basic elements of a high-lift drive system, namely the power drive unit, the actuators and the shaft transmission connecting actuators and drive unit, were introduced in chapter 1. Besides gearboxes, shafts, joints and bearings that are essential for the shaft routing, there are further components required to react to mechanical failures. A shaft rupture leading to a separation of flap segments might result either in an asymmetric flap setting or even the complete transmission system might be decoupled from the PDU so that the aircraft looses its high-lift function in a critical situation. Furthermore, jamming in the flap tracks might cause an asymmetric flap setting as well as an overload in actuation system and wing structure. In order to avoid an unacceptable flap asymmetry that cannot be compensated by the rudders, safety brakes are installed at the spanwise ends of the shaft transmission. These wing tip brakes (WTB) are activated if the monitoring systems identifies a failure by comparing the position at the transmission ends, the drive units output angle and the commanded position.

Moreover, the installation of torque limiting devices reduces loads in the drive system and structure in case of jamming in one of the drive stations. High loads and load gradients result from rapid deceleration of the system by either jamming or brake activation.

The analysis of such transient behaviour requires a nonlinear model. Figure 2 exemplifies a flap drive system architecture and its elements. For the purpose of an acceptable simulation time, modelling each mechanical element separately is not practicable. Thus, adjacent parts are merged into a lumped model. The total inertia and torsional stiffness can easily be calculated from the elements connected in series. Other variables like friction coefficients or backlash can be determined accordingly.

While the system model in contrast to the real drive system posesses concentrated parameters, an appropriate discretisation must not change the dynamic behaviour of the system. Different approaches have

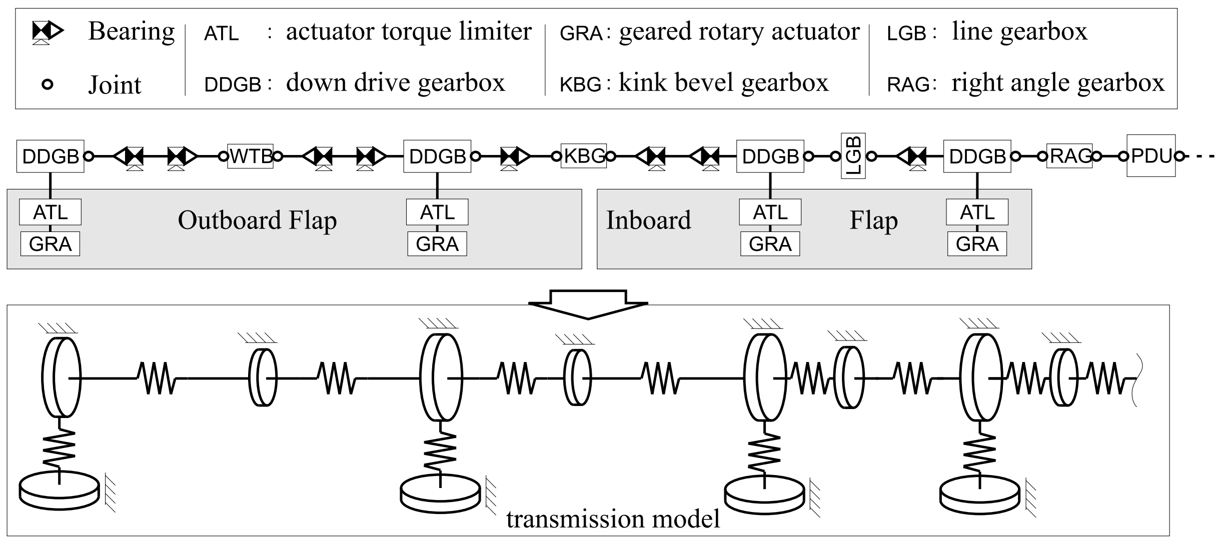


Figure 2: Schematic representation of a flap drive system and its model according to [4]

proven their value. In [2], Neumann proposes to substitute all transmission elements between the downdrives by at least two systems made up of spring, damper and mass. Adapting the distribution of the torsional stiffnesses, the relevant natural modes can be preserved.
A closer look at the distribution of drive system element masses and torsional stiffnesses reveals that installed gearboxes make up a major share of the total mass moment of inertia, while the torsional stiffness is mainly influenced by long shaft elements. Another approach, reducing the model order by summing up the mass moments of inertia around those areas that already show an accumulated inertia, like gearboxes, is presented in [4]. On the other hand, a single torsional spring represents the torsional stiffness of the elements between those inertias.
Both methods show a good correlation between simulation and measured data.

3 Development of high-lift actuation systems

The design and construction process of a new high-lift actuation system starts early in the overall development process of a new aircraft. Thus, only few and uncertain information is available at the beginning. Especially in the concept and preliminary design phase, the requirements, system constraints and component data often change. The data becomes more reliable and more detailed while the development process proceeds. However, mandatory inputs for the design process of the high-lift drive systems that have to be available from the start are:

- the number and type of actuators. The application of either a geared rotary actuator or a ballscrew actuator depends on the planned flap kinematics.
- effective airloads at the actuators.
- wing geometry and available installation space in the wing area.
- maximum travel of the actuators and required time for their adjustment.

The development and design of the mechanical transmission system, the hydraulic and/or electric power drive unit and the monitoring and failure detection system is complex and highly interdependent. The focus in this article is on the actuation system.
With the listed inputs, a first drive system architecture can be designed schematically. To guarantee a uniform motion of the actuators, gear ratios have to be determined accordingly. Mechanical properties of the components have to be estimated at first. Experience from the development of former drive systems is of tremendous value for this parameter estimation.
If gear ratios, characteristic friction coefficients and the architecture are known, steady state calculations, e.g. for determination of the drive torque required by the PDU and torque limiter settings, are possible.
For a rough evaluation of maximum loads resulting from transient changes in consequence of wing tip brake activation or torque limiter lock out, simple models seem practicable. As proposed in [4], the part of the transmission that is in focus of the analysis can

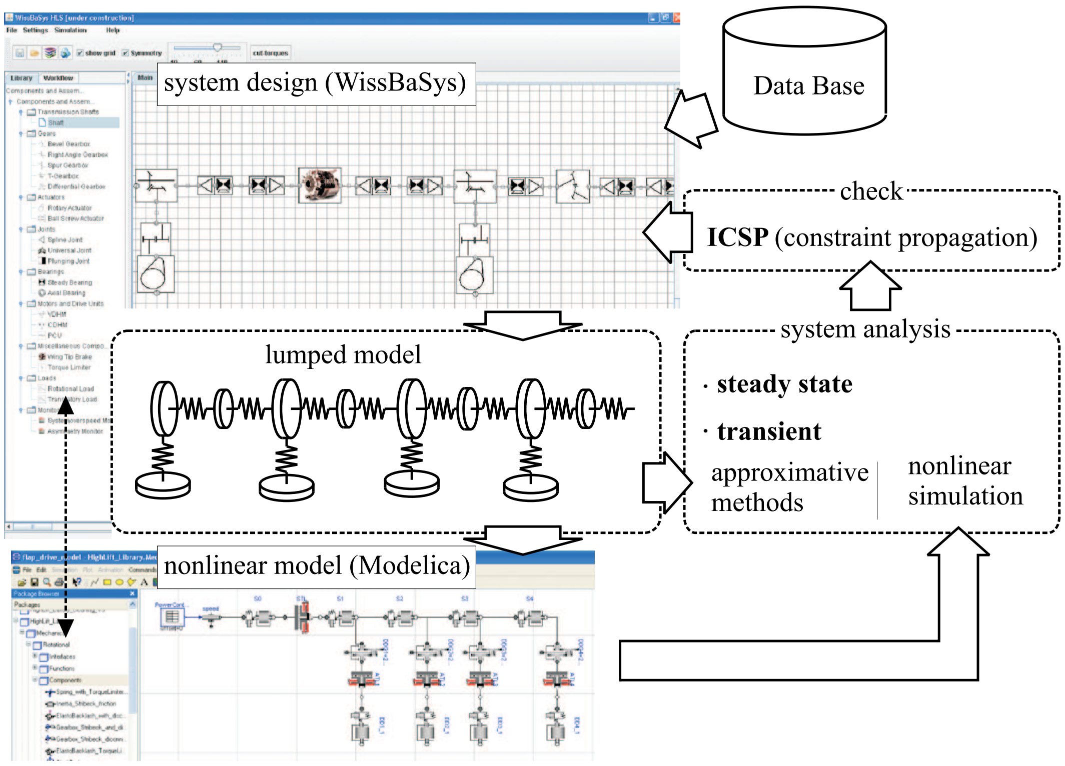


Figure 3: Concept for an computer-aided development process of high-lift drive systems

be transformed to a torsional oscillator with a single inertia J^*. For dynamic similarity, the torsional stiffness c^* of the vibrator is adjusted, such that the first eigen mode of the complete transmission system and the reduced model are identical. Presuming a sudden deadlock in the transmission and neglecting nonlinear influences, the kinetic energy E_{kin} of the transmission converts to potential energy E_{pot} in the spring, allowing the calculation of the peak load:

$$\begin{aligned} E_{kin} &= E_{pot} \\ \Rightarrow \frac{1}{2}\cdot J^*\cdot\omega^2 &= \frac{1}{2}\cdot c^*\cdot\Delta\varphi^2 = \frac{1}{2}\cdot\frac{\tau_s^2}{c^*} \\ \Rightarrow \tau_{s,max} &= \omega_{max}\cdot\sqrt{c^*\cdot J^*} \quad . \end{aligned} \tag{1}$$

Thus, the possibility to do rough system evaluations and trade-offs is provided. For example, the system dependency on the chosen gear ratio could be analysed.

Regarding equation (1), another problem seems obvious. The maximum transmission speed, especially after a mechanical disconnect, depends on nonlinear friction characteristics. While the effort of generating a complex simulation model and the time for running these simulations is not justified as long as most parameters are uncertain and many changes are necessary, the need for more detailed system analysis when the system specification reaches a mature level and reliable data are available is obvious.

In order to reduce development time, the Institute of Aircraft-Systems Technology at Hamburg University of Technology is working on a tool called WissBaSys to support the design process. Particularly, the efforts in early design and specification phases, that are in focus, could be reduced by numerous computer-aided features, which are introduced hereafter.

While the architecture of high-lift transmission systems may change, they generally consist of a relatively small number of different mechanical components. Thus, a library of generic, parameterised components has been created. A graphical user interface (GUI) offers the possibility to connect these generic elements to a complete transmission system. The resulting system layout can easily be changed by adding or removing components.

In order to support the difficult task of parameter estimation when reliable data are not available, not only default values are provided, but also functions describing an interdependence between variables are supported. Furthermore, the user has access to an external database containing extensive information about many existing aircraft components.

Another characteristic of the preliminary design phase is the handling of uncertain knowledge and checking the system requirements after every change. For this reason, continuous domains are attached to all variables. This is the basis for an interval constraint satisfaction problem (ICSP). Constraint propagation as it is presented in [5] enables the evaluation of nondirectional equations and inequalities containing variables with interval domains.

Establishing an ICSP brings further useful advantages. Enabling nondirectional evaluation, trade-off studies are encouraged. Furthermore, violations of system requirements or constraints are detected automatically within the constraint propagation process.

The concept of a computer-aided development process is illustrated in figure 3. The system architecture is assembled utilising generic library components. System and component parameters are estimated with help of data base information, default values and empirical estimation functions. An automated generation of simplified models enables approximation of maximum load result from transient behaviour. The system analysis is completed by steady state calculations. The ICSP automatically checks all system requirements so that the basis for a system synthesis is available. While synthesis methods allow for an evaluation and optimisation of slat and flap traverse mechanisms [1] an all-including high-lift optimisation on aircraft level is not available up to now.

Containing all relevant component data, the transfer to a complex nonlinear simulation model would complete the development process. The way Modelica uses for modelling by combining generic library components offers ideal possibilities for an interaction in this context.

WissBaSys supports design studies in early development phases and generates lumped models of reduced order. An appropriate Modelica model has to be named for general concentrated transmission sections. Presuming the allocation of available and model parameters is existent, model instances corresponding to the concentrated parameters can be generated. With the knowledge that some parts execute special function, e.g. the wing tip brake, additional models have to be inserted. If an allocation of simulation models for the mechanical elements in the transmission system is existent, the generation of the complete simulation model can be realised.

4 HighLift library for drive systems

The high-lift drive system consists of the mechanical actuation system, hydraulic drive units, as well as a control and fault detection system. Here, the actuation system and the power drive unit are considered in more detail. For a determination of maximum transmission loads, the mechanical components of the drive train can be modelled as one-dimensional rotational elements. These are characterised by their mass moment of inertia, a torsional stiffness, structural damping, mechanical backlash, gear ratio and the friction characteristic. While the models *Inertia*, *ElastoBacklash* and *IdealGear* of Modelica's standard library cover most of these attributes a new friction model is needed and introduced in this chapter.

Besides models representing a nonlinear torsional oscillator, some components fulfill additional tasks that have to be taken into account. These components are the safety brakes and mechanical torque limiters. The HighLift library contains models for a shaft brake, an ideal torque limiter, the general mechanical rotational part and a geared rotary actuator. Moreover, hydraulic components necessary for modelling hydraulic power drive units are available.

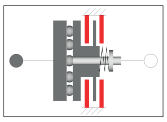
The focus is on the mechanical drive train and its relevant models are discussed in the following. All models are designed such that they need only the information that is relevant for a specification.

Shaft brake

This model represents the brake function of the wing tip brake, which is mounted in the wing structure. If the brake is activated, the compression of friction packages causes a friction torque that stops the transmission. Essential parameters describing the brake behaviour are the maximum dynamic brake torque, maximum static brake torque and the time for reaching the maximum dynamic torque. Thus these are the only input variables of the model which extends the interfaces *Rigid* and *FrictionBase*.

In order to allow different approaches for describing the transient change of the friction torque when the brake is activated, the model's input `u` is the normalized maximum dynamic brake torque `tauB_max`. After reaching a halt, the static friction torque might increase up to the brake's maximum limit load `tauB_lim`. In contrast to the models available in the standard library, friction coefficients are no longer needed here.

Ideal torque limiter



A coupling consisting of balls embedded along the circumferece of two flanges guarantees a positive connection in normal operation mode. In case a torque limit is passed, the balls start to move along a ramp thus pushing one of the flanges against a friction device. The increasing relative angle between the flanges results in an increasing brake torque.
The torque limiting function has two characteristics. First of all, a brake torque depending on the relative angle of the flanges is induced. Moreover, the torsional stiffness changes within the lock out process. While the balls are in motion, the stiffness decreases significantly compared to the normal operation mode. When the balls reach their end stop, the device is grounded and the torsional stiffness changes again.

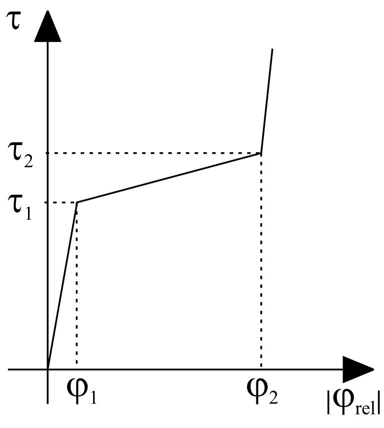


Figure 4: Nonlinear torsional stiffness characteristic of a mechanical torque limiter

For modelling these characteristics, a torsional spring with nonlinear stiffness, according to figure 4, is needed. Furthermore, the dynamic brake torque increases after lock out and reaches its maximum when the balls reach their end stop.
A new spring model has been created. Required inputs are the lock out torque τ_1 and the end stop torque τ_2 as well as the different torsional stiffnesses for all three states. Compared to the standard spring, this model has an additional output y describing a normalized brake torque:

$$y = \begin{cases} 0 & : \quad \varphi_{rel} < \varphi_1 \\ 1 & : \quad |\varphi_{rel}| \geq \varphi_2 \\ \dfrac{|\varphi_{rel}| - \varphi_1}{\varphi_2 - \varphi_1} & : \quad \varphi_1 \leq |\varphi_{rel}| < \varphi_2 \end{cases} \tag{2}$$

Combining the nonlinear spring with a shaft brake as figure 5 shows, an ideal mechanical torque limiter is modelled.

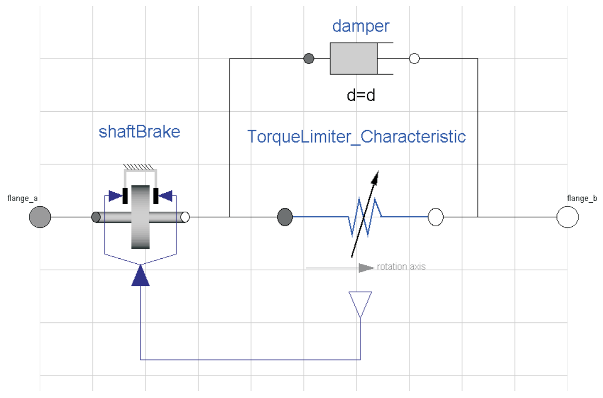
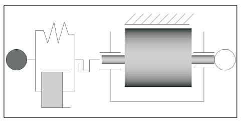


Figure 5: Ideal torque limiter model

General rotational element



As shown in its symbol the general rotational mechanical element consists of an *ElastoBacklash* model and a modified *Inertia* as well as of an *IdealGear*. The *LossyInerta* model takes friction losses into account. Most elements of the transmission system like bearings show friction behaviour corresponding to the Stribeck Friction Law:

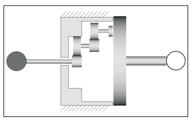
$$\tau_{fric,S} = \tau_{Coulomb} + d_{vis} \cdot \omega + \tau_{Stribeck} \cdot e^{-f_{exp} \cdot |\omega|} \tag{3}$$

However, the detailed analysis of single state gearboxes shows additional friction losses that highly depend on the transmitted loads [6]. This phenomenon is valid only when the unit is in motion and the breakout has occurred. Based on the results of sophisticated analyse of gearbox friction behaviour, a combined approach appears feasible. As discussed in [3] bearing losses and load dependent gear stage losses differ. For representation of a total drag torque, the friction torque is made up of the bearing friction according to the Stribeck law which is depending on ambient conditions and gearbox losses characterised by a gearbox efficiency η_{GE}:

$$\tau_{fric} = \tau_{fric,S} + (1 - \eta_{GE}) \cdot \tau_{load} \quad . \tag{4}$$

While η_{GE} varies between 0 and 1, it represents the dependence on the transmitted torque and is easily determined by measurement.

Geared rotary actuator



Most Airbus aircraft use planetary gears with high gear reduction for flap and slat actuation. Their dynamic behaviour has essential effects on the complete high-lift actuation system. Exact modelling of these components is of vital importance for the reliability of simulation results. Analysis of the friction behaviour of these actuator types also shows remarkable influence of the transmitted loads on the friction torque [2]. Furthermore, the load-dependent friction changes with the energy flow direction. Generally, driving against opposing load has better efficiency than in the case of aiding loads. The load-dependent friction does not occur stepwise as soon as the unit begins to move, but increases smoothly after a change in direction.

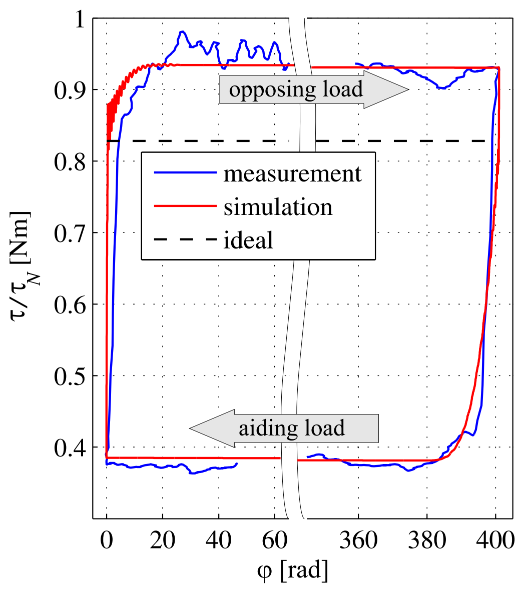


Figure 6: Normalised input torque of a geared rotary actuator with constant load

For validation a geared rotary actuator has been tested and its friction behaviour determined [2]. Simulation results using the model described above show good resemblance to test data as presented in figure 6. For validation of the actuation system measured data of the actuator loads as well as the power drive unit's speed is an inputs to the model. The contact of the gear wheel teeth is the reason for the load-dependent friction torque [3]. When a turnaround occurs, the wheels do not turn simultaneous but consecutively. Thus, the contact between the gear wheels establishes smoothly. Since the geared rotary actuator is modelled as a single stage gearbox in order to reduce the model order, this phenomenon can be represented by the gearbox efficiency η_{GE} as a function of the input angle φ_{in}. If the unit stops, η_{GE} increases linearly to 1 after a speed threshold is crossed. Consequently, load-dependent friction diminishes according to equation (4). When the unit starts to move again, η_{GE} is a function of φ_{in}, while its final value depends on the sign of the transmitted power. Figure 7 shows this characteristic.

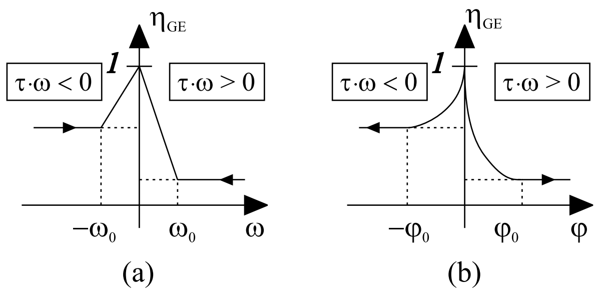


Figure 7: Gearbox efficiency for deceleration (a) and acceleration (b)

Further Models

The models presented in detail here are of vital importance for modelling a complete high-lift actuation system. Furthermore, the HighLift library contains models for inducing mechanical failures in the drive train. For this purpose, an element that can be used for a mechanical disconnection and another model that causes jamming at a specified time are included. Besides the transmission system, the power drive unit is of major interest. Hydraulic component models for turbulent resistances, servo valves, a differential cylinder as well as an example that uses these components for modelling a PDU's drive train with a variable displacement hydraulic motor (VDHM) are included.

5 Transient simulation of Airbus A380 flap actuation system

For a verification of the presented models the Airbus A380 flap actuation system is taken into account. The number of actuators and mechanical elements in total outnumbers that of all other flap actuation systems of Airbus aircraft. The actuation system utilises geared rotary actuators, a wing tip brake and a system torque limiter that is installed between the power drive unit and the first downdrive. A test rig replicating the A380 high-lift drive system of one wing only, has been installed at the Airbus facilities in Bremen in order to run certification tests. Utilising the models

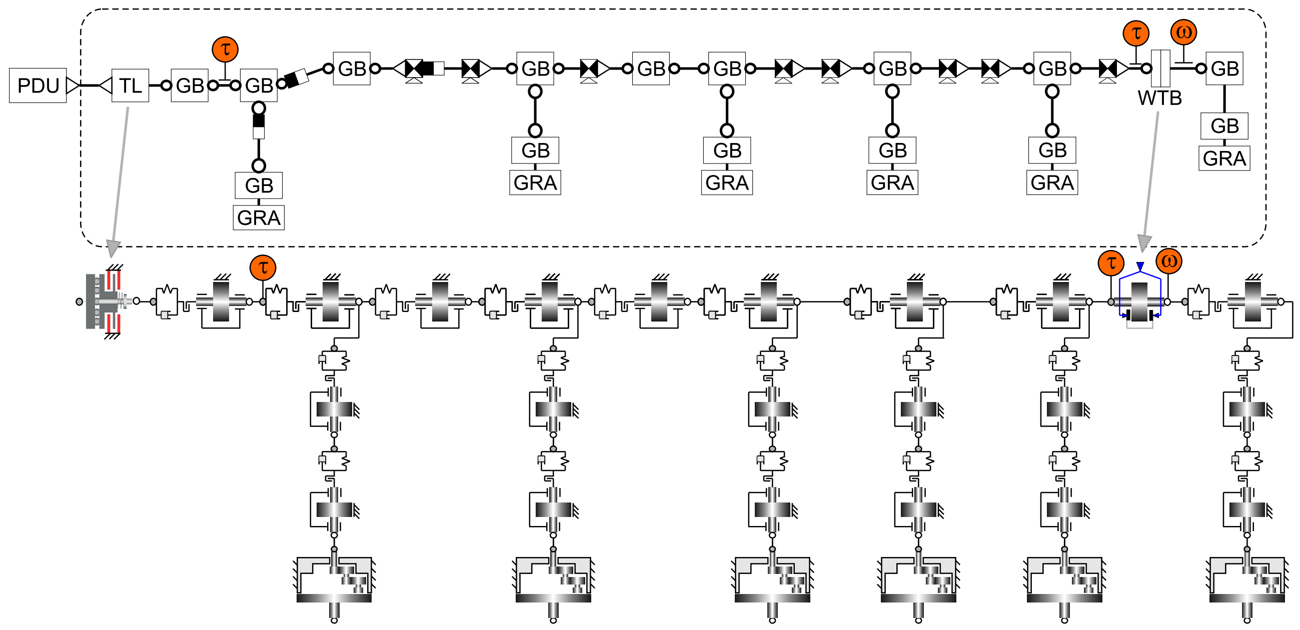


Figure 8: Airbus A380 flap actuation system and model

of the HighLift library the actuation system is modelled and verified by means of measured data. Figure 8 presents a schematic view of the transmission system and a lumped model of reduced order in Modelica. Furthermore, sensor positions are marked in figure 8. For modelling the approach presented in [4] and discussed in 2 is used.

For validation test data of the actuator loads and the PDU speed are used as input. The drive systems starts to operate against increasing opposing actuator loads. After an acceleration phase the system speed is almost constant until a position threshold is reached and the speed is reduced before the system stopps at its determined position.

Figure 9 shows that the speed within the shaft transmission system varies only slightly. Comparing simulation and test data for the input torque at the system torque limiter (STL) that depends on the exact modelling of the complete actuation system, the data show good conformability. While the simulated break out occurs 0.5 seconds earlier than in the test the simulation results are very accurate afterwards. Figure 10 compares simulation and test results for the specified sensor positions.

Now the introduced model is used to analyse a failure case scenario. At t_1 the disconnector model is used to simulate a shaft rupture between system torque limiter and first downdrive while the transmission system

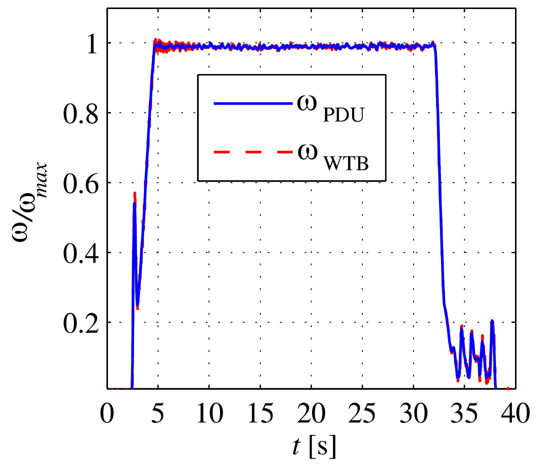


Figure 9: Transmission speed

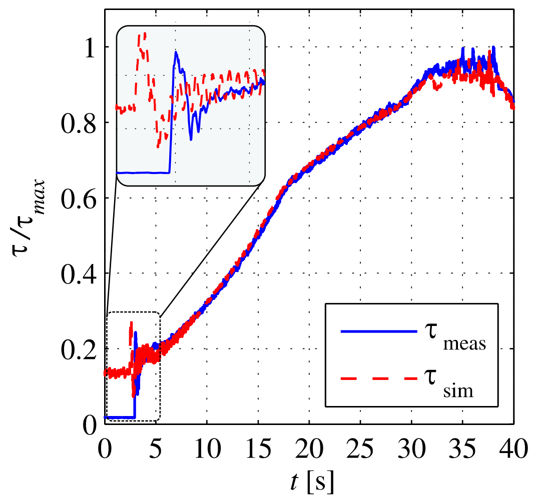


Figure 10: Simulated and measured actuation system input torque

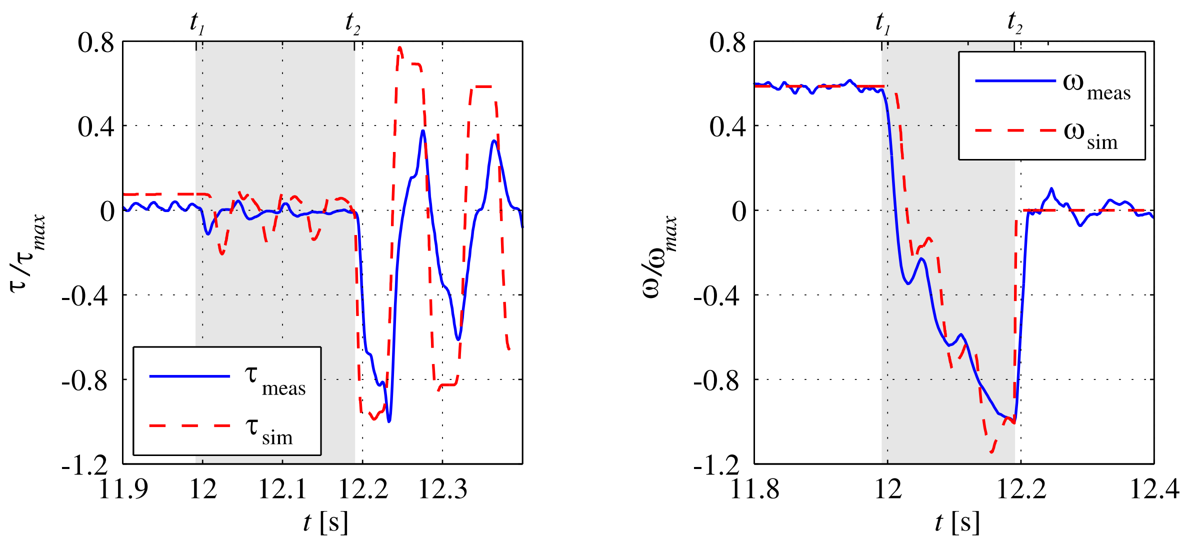


Figure 11: Transmission speed and torque at the wing tip brake after shaft rupture and brake activation

drives against opposing loads. After the mechanical disconnection the complete system is accelerated by the applied actuator loads. The failure is detected and the wing tip brakes are applied at t_2 and cause a system stop. In consequence of the rapid deceleration,
load peaks occur within the shaft transmission. The maximum is to be found at the safety brake.
Figure 11 compares test rig data and simulation results for transmission speed and torque at the wing tip brake. Although simulated and measured speed have different gradients during the acceleration phase, their oscillatory behaviour is similar and their value at t_2 is almost identical. The simulated deceleration phase is shorter as it was in the test. Nonetheless, the maximum transmission loads differ only slightly.

6 Conclusion and future work

This article presents the development and design of high-lift actuation systems and its implied challenges. For further reduction of development time for new high-lift systems a computer-aided approached is aspired. In order to enable an automated generation of nonlinear models for simulation of the complete drive train, a library containing all essential elements of the described drive system is introduced. With the help of the modelled components the Airbus A380 flap actuation system has been modelled. Simulating a normal extension cycle, the simulation model provides results that are close to measured data. The verified model is used for analysis of maximum loads when the safety brakes are applied after a shaft rupture.
While the basis for an interface between the design tool WissBaSys and the Modelica environment has been established by the implementation of the presented HighLift library, its execution is still outstanding. Furthermore a simulation of the complete system including the power drive unit as well as the slat flap control computer is necessary.

Acknowledgment

The authors thank the Airbus Deutschland GmbH for sponsoring and supporting this project.

References

[1] Holert, B.: Eine Methode zum mehrkriteriellen Entwurf der Führungsmechanismen in Hochauftriebssystemen von Transportflugzeugen. Hamburg: Dissertation, Institut für Flugzeug-Systemtechnik, TU Hamburg-Harburg, 2005.

[2] Neumann, U.; Holert, B.; Carl, U.B.: Für eine sichere Landung - Simulation von Landeklappenantriebssystemen. Antriebstechnik, 4/2004.

[3] Pelchen, C.;Schweiter, C.; Otter, M.: Modeling and Simulating the Efficiency of Gearboxed and of Planetary Gearboxes. 2nd International Modelica Conference, Oberpfaffenhofen 2002.

[4] Rechter, H.; Richter, M.: Die Simulation als Hilfsmittel bei der Entwicklung und Integration der A330/340-Hochauftriebssysteme. DGLR-Jahrestagung, Band 1, Göttingen 1993.

[5] Runte, W.: YACS: Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung. Diplomarbeit, Fachbereich Mathematik / Informatik, Universität Bremen, 2006.

[6] Ruprecht, T.; Thielecke, F; Recksiek, M: SIVA - A testrig for the validation of high lift component models. AST Workshop on Aircraft System Technologies, Hamburg 2007.

Session 5

Poster Session

4-DIMENSIONAL TABLE INTERPOLATION WITH MODELICA

Tobias Hirsch Markus Eck
German Aerospace Center (DLR)
Pfaffenwaldring 38-40, 70569 Stuttgart, Germany
tobias.hirsch@dlr.de, markus.eck@dlr.de

Abstract

The steady-state model for a solar field contains a large number of equations including conditional statements. For a yearly energy yield analysis the operational state (on duty, off duty) of the solar field may change from one time instant to the other. Due to the strongly varying boundary conditions a simulation run without convergence problems is not likely. For this reason a lookup-table model is designed to calculate the five output variables of the solar field depending on the four input variables. The interpolation model is based on the existing MODELICA model for 2D-interpolation and can be used for table interpolation tasks independent of the technical application. The structure of the model and a method for the automatic generation of the required interpolation data from the complex solar field model is described.

Keywords: solar power plant; look-up table; interpolation

1 Introduction

Solar thermal power plants are one of the most interesting options for renewable electricity production. For the calculation of the annual energy yield of these plants steady-state models are used. The calculation method which is based on mass and energy balances is called for every hour of the year with the corresponding weather data input and delivers an output of electric energy. This approach works well as long as transient effects in the plant can be neglected. When a thermal storage has to be considered an additional transient model has to be implemented. Since the solar field and the power block can still be represented as a steady-state block, the final plant model is composed of very complex steady-state models for the solar field and the power block and a rather simple transient model of the storage system. For an annual calculation on an hourly basis, the model is called 8760 times with input data that might be strongly varying from hour to hour. First tests with the complex steady-state models show that robustness of the simulation is not satisfying. Due to the large changes in input parameters and model dependencies it is very likely that an annual calculation might terminate before reaching the end time.

The reason for the complexity of the solar field model is the aspect that the model has to describe the operation in full load, part load and stand-by mode. While mass and energy balances are derived for regular field operation this is not the case for the stand-by mode. In order to determine the time instant with irradiation conditions sufficient for a switch from stand-by into part-load operation the set of balance equations has to be solved with a modified set of input parameters even if the field is shut-down. Implementing the equations within the MODELICA language yields a number of conditional statements that have to be operated by the solver. Robustness of the resulting system is hard to check and may differ from one field layout to the other.

A way to couple the complex steady-state field model with the simple transient thermal storage model is developed by replacing the equation-based solar field model by a table-based interpolation. When analyzing the system it is found that the solar field output is determined by just four independent inputs. Unfortunately, the existing interpolation model in MODELICA is limited to two independent variables. Within this paper, a MODELICA model is presented that allows a three dimensional interpolation using the MODELICA 2D-interpolation model. By an additional interpolation level the capability can easily be extended to an interpolation in four dimensions.

2 Solar field model characteristics

The solar field is composed of a large number of parabolic trough collector rows arranged in parallel. The water fed into the field at high pressure is preheated, evaporated and superheated by the solar irradiation. This kind of system is called a Direct Steam Generation parabolic trough power plant [1]. Apart from general parameters of the field, the output of the solar field is determined by the following input variables:

- Direct normal irradiation, *DNI*
- Ambient temperature, *T_amb*
- Feed water specific enthalpy, *h_in*
- Operating pressure of the field, *p_out*

All of these are a function of time with the first two taken from the weather data file and the last two being determined by the whole plant model. In addition to the generated mass flow, four more outputs have to be provided by the model, so the list of output variables reads:

- Steam mass flow, *m_out*
- Field inlet pressure, *p_in*
- Field outlet temperature, *T_out*
- Recirculation pump power, *P_rec*
- "Field in operation"-indicator, *FIO*

A MODELICA solar field model is available that describes the relation between input- and output parameters based on the physical equations. The model allows changes in the solar field configuration in an easy way by simply changing some parameters that e.g. determine the number or arrangement of collector rows. It is therefore suited for the design of a solar field but is not suited for annual energy yield analysis.

3 General approach

The physically based solar field model is replaced by a table interpolation model that calculates one output variable (e.g. *m_out*) based on a set of interpolation data and the three input variables (*h_in*, *p_out*, *DNI*). Extension to the forth input variable is done by linear interpolation in the ambient temperature (*T_amb*). For each of the five output variables the same interpolation model can be used with an individual set of interpolation data. The interpolation data are automatically generated by calling the physical solar field model from a MATLAB script for all nodes of the interpolation data. The outputs of the solar field are stored in MATLAB .mat files and can directly be read by the MOCELICA interpolation model. Within the following sections the automatic generation of the interpolation data and the structure of the interpolation model will be described.

4 Generation of interpolation data

Since a large number of solar field configurations, each described by one set of interpolation data, is to be analysed for the yearly output, an efficient method is needed to generate the interpolation data. For the interpolation routines in MODELICA one look-up table in three dimensions (variation of input variables *p_out*, *h_in*, *DNI*) has to be provided for each of the five output variables (*m_out*, *p_in*, *T_out*, *P_rec*, *FIO*).

This is realized by a MATLB script file that calls the MODELICA executable for all combinations of input variables. By use of the DYMOLA-MATLAB interface the output variables are then stored by the MATLAB script in a ".mat"-file. For each output variable a separate file is generated that stores the three vectors of parameter variations

```
p_steps =[p_start : dp_: p_end]
h_steps =[h_start : dh_: h_end] ;
I_steps =[I_start : dI_: I_end] ;
```

and the three-dimensional result matrix containing the results at the nodes defined by the vectors above. The procedure is illustrated in figure 1.

Due to the complexity of the solar field model it is initialized with a fixed set of parameters. The desired operating point for each input parameter combination

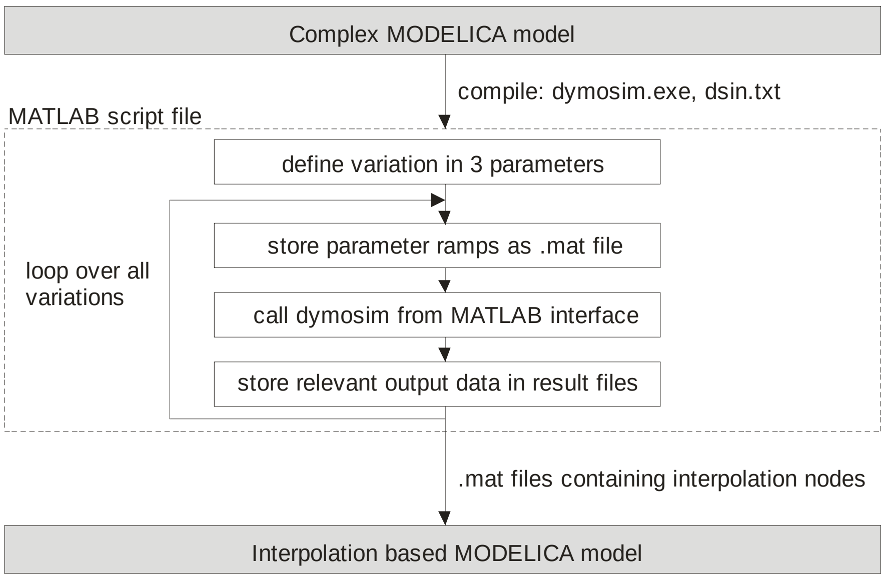

Complex MODELICA model
compile: dymosim.exe, dsin.txt
MATLAB script file
define variation in 3 parameters
loop over all variations
store parameter ramps as .mat file
call dymosim from MATLAB interface
store relevant output data in result files
.mat files containing interpolation nodes
Interpolation based MODELICA model

Figure 1: Procedure for generation of interpolation data

is then reached by a ramp in the three input variables. The final state of the ramp (values of the input variables for the actual combination) is stored by the MATLAB script in a .mat file before the executable is called. The data are then read by the executable to define ramps in the input variables that lead from the fixed initialization state to the desired final state. This approach has the advantage that no problems with the initialization occur during the parameter variations due to the stable initialization state. One separate call of the executable for each parameter variation is chosen, although the ramps might have been defined to generate a number of results points in one simulation run. The advantages for the implementation chosen are:

- only one data point is lost if the simulation does not converge
- high flexibility in the definition of the parameter variations (e.g. no need for equidistant grids) .

The output variable *FIO* is very important for the following interpretation of the interpolated data since it determines if a data point calculated by interpolation is valid. The value is set to false if the solar field can not be operated for the combination of input variables or if the simulation has not converged. In both cases, the data points obtained from the interpolation do not represent a physical state of the solar field.

In order to allow direct access to the interpolation data from the MODELICA 2D-interpolation model *CombiTable2D* the data a stored in the following way. For each value of input variable x_3, e.g. 70 bar, 80 bar, 90 bar, 100 bar, 110 bar, a set of 2D-interpolation data are stored in one separate matrix. In our example, these matrices are named *data1* to *data5*. The matrix contains in the first row the vector of nodes in variable x_2 and in the first column the vector of nodes in variable x_1. The matrix is then filled with the output data at the corresponding nodes:

0	$x_2(1)$	$x_2(2)$	...	$x_2(ih)$
$x_1(1)$	dat(1,1)	dat(1,2)...		dat(1,4)
$x_1(2)$	dat(2,1)	dat(2,2)...		dat(2,4)
...	...	...	...	...
$x_1(iI)$	...	...	...	dat(iI,ih)

All data matrices together are stored in one single .mat-file. This file holds all data necessary for the 3D-interpolation in variables x_1, x_2 and x_3. For each output variable that has to be described by 3D-interpolation a separate file is generated. This allows, in principle, an arbitrary number of output variables. In our example, five output variables are used with the data stored in the files *FIO.mat*, *m_flow.mat*, *p_in.mat*, *P_rec.mat*, *T_out.mat*.

5 3D interpolation model

The three-dimensional table interpolation used in the yearly analyzer is based on the two-dimensional table interpolation model available in the MODELICA standard library. This model is very efficient since the search for the interpolation interval starts at the result found in the last time instant. The two dimensional interpolation model is used to interpolate in the variables x_1 (*DNI*) and x_2 (*h_in*) for a fixed value of variable x_3 (p_out). For each value of the variable x_3 defined in the vector *p_steps* one value u_i (i=1:n) for the output variable is calculated. The final output value is then generated by a 1-D interpolation in the n results u_i. The procedure is illustrated in figure 2. The model that holds the following equations is named *Kennlinie3D* (german word for Characteristic3D). In the following, the code of this model is described. The model contains three inputs

```
Modelica.Blocks.Interfaces.RealInput x1;
Modelica.Blocks.Interfaces.RealInput x2;
Modelica.Blocks.Interfaces.RealInput x3;
```

for variables x_1, x_2 and x_3. In the solar field example these inputs correspond to *h_in*, *DNI*, *p_out*. The result is delivered via output

```
Modelica.Blocks.Interfaces.RealOutput y;
```

A data structure is defined to provide information on the upper and lower limits of x_1 and x_2 as well as the matrix name in the interpolation file that holds the interpolation data.

```
encapsulated record interpolation_source
      Real   x3;
      Real   min_x1;
      Real   max_x1;
      Real   min_x2;
      Real   max_x2;
      String table_name;
end interpolation_source;
```

In the model n instances of this data structure are created as parameters by:

```
parameter interpolation_source[:]
      IP_source;
```

In Dymola, the data can be entered via the graphical user interface which is shown in figure 3. In this example, 2-D-interpolation in x_1 and x_2 data have been generated for five pressure levels from 70 bar up to

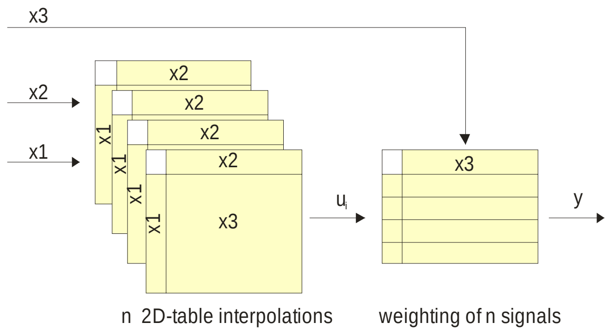


Figure 2: Structure of the 3D interpolation model

110 bar. The interpolation data are found in matrices *data1* to *data5* in the interpolation data file defined by `parameter String SourceFile= "p_in"`.

The variable x_1 (*h_in*) may vary between 500 kJ/kg and 1100 kJ/kg and the variable x_2 (*DNI*) between 0 and 1000 W/m^2. The 2-dimensional interpolation is done in *n* MODELICA interpolation blocks which are instantiated by

```
Modelica.Blocks.Tables.CombiTable2D
     IP_table[n](
         each tableOnFile=true,
         each fileName=SourceFile,
         tableName={IP_source[i].table_name
                    for i in 1:n}
                 );
```

The inputs x_1 and x_2 and connected to the corresponding inputs u_1 and u_2 of the *n* interpolation blocks, taking into account the variable range limitations defined in `IP_source`.

```
for i in 1:n loop
   IP_table[i].u1=
      max(IP_source[i].min_x1,
          min( IP_source[i].max_x1, x1 )
          );

   IP_table[i].u2=
      max(IP_source[i].min_x2,
          min( IP_source[i].max_x2, x2 )
          );
end for;
```

The final result is calculated by weighting the *n* outputs of the 2D-interpolation blocks

```
  y = sum(  IP_table[i].y*weight[i]
            for i in 1:n );
```

The weighting factors are calculated from a linear interpolation in the variable x_3. For example, a value of x_3=82e5 Pa would lead to a vector of weighting factors *weight* =[0 0.8 0.2 0 0]. The Dymola routine *dymTableIpo1* is used for the interpolation. This routine has to be initialized by

```
when initial() then
   Weight_tableID=dymTableInit(
           1.0,
           smoothness,
           "NoName",
           "NoName",
           Weight_matrix,
           0.0);
end when;
```

and called with the command

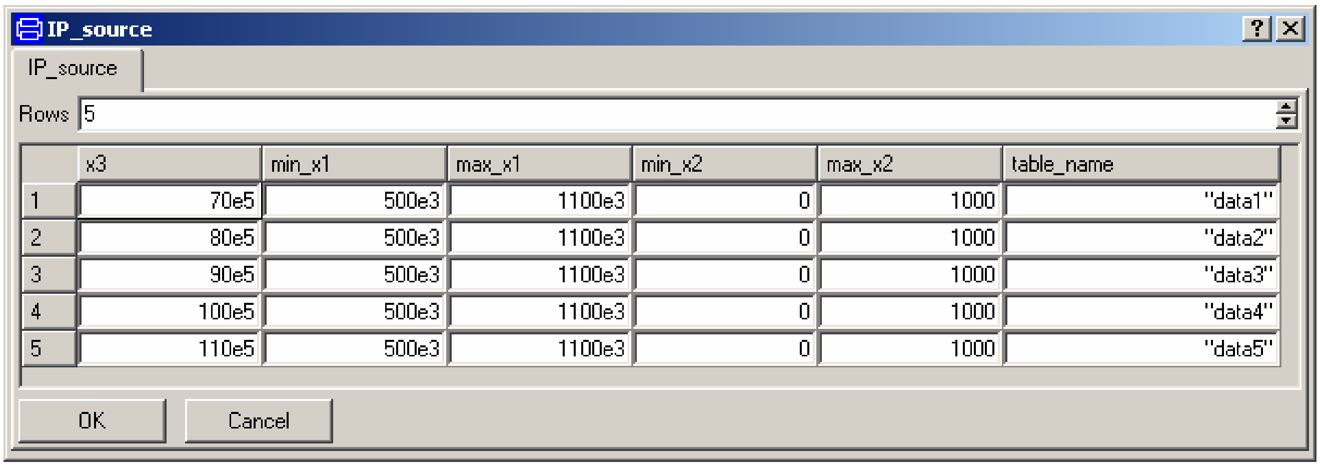
```
for i in 1:n loop
   weight[i] =
      min(1.0,
          max(0.0,dymTableIpo1(
                  Weight_tableID,
                  Weight_columns[i],
                  x3))  );
end for;
```

with the corresponding declarations

```
parameter Real[:,:]   Weight_matrix =
    [IP_source.x3, diagonal(ones(n))];

parameter Integer     Weight_columns[:]=
    2:size(Weight_matrix, 2);

Real      Weight_tableID;
Real[n]  weight;
parameter
   Modelica.Blocks.Types.Smoothness.
   Temp    smoothness =
   Modelica.Blocks.Types.Smoothness.
        LinearSegments;
```



Rows 5

	x3	min_x1	max_x1	min_x2	max_x2	table_name
1	70e5	500e3	1100e3	0	1000	"data1"
2	80e5	500e3	1100e3	0	1000	"data2"
3	90e5	500e3	1100e3	0	1000	"data3"
4	100e5	500e3	1100e3	0	1000	"data4"
5	110e5	500e3	1100e3	0	1000	"data5"

Figure 3: Screenshot of the Dymola graphical user interface for `IP_source` with five pressure levels

6 Solar field model with 3 inputs

The solar field model *SolarField_Characteristic* based on the interpolation is assembled from five 3D-interpolation blocks of type *Kennlinie3D* as shown in figure 4. The three input connectors for *h_in* (red lines), *DNI* (blue lines) and *p_out* (green lines) are connected to the corresponding inputs of the 3D-interpolation blocks. Based on the interpolation data provided in files *FIO.mat*, *m_flow.mat*, *p_in.mat*, *P_rec.mat*, *T_out.mat* the outputs *FIO*, *m_flow*, *p_in*, *P_rec* and *T_out* are calculated. The values are only valid if the indicator *FIO* is 1. In case this value is smaller than 1, a default value, e.g. 70 bar for *p_in*, is used instead of the calculated value.

7 Extension to four dimensions

As mentioned in the beginning of this text the solar field output depends on one more variable namely the ambient temperature. Since the dependence on this variable is nearly linear three nodes in ambient temperature (0 °C, 20 °C, 40 °C) are sufficient for the model. For each of the three temperature levels a separate set of interpolation data is generated. Three instances of the solar field model *SolarField_Characteristic* are created with the outputs linearly weighted with the actual ambient temperature *T_amb*. The weighting is realized by the same approach as in the 3D-interpolation model using the Dymola function *dymTableIpo1*. For reusability a model called *WeightedSignals* is defined. Figure 5 shows a screenshot of the final solar field model with the three *SolarField_Characteristic* models each representing one level of ambient temperature and five *WeightedSignals* models that are responsible for weighting obtained from the three interpolation models.

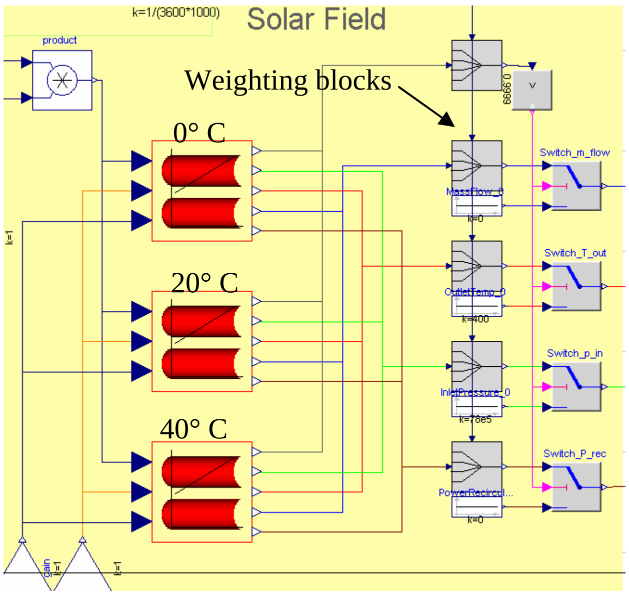


Figure 5: Solar field model with three instances of the *SolarField_Characteristic* model representing three levels of ambient temperatures

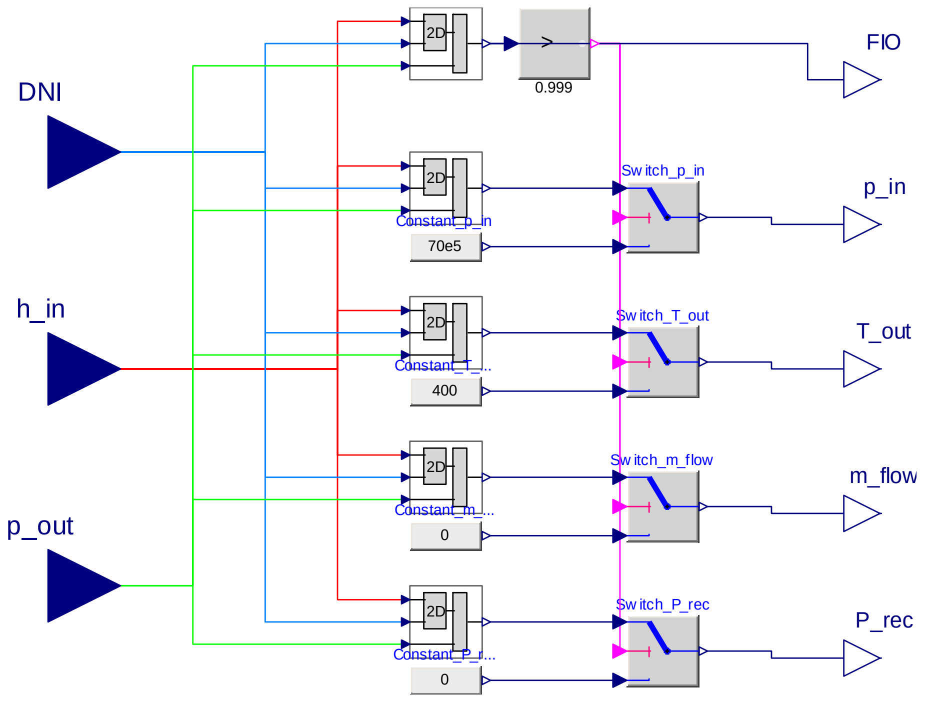


Figure 4: The *SolarField_Characteristic* model composed of five 3D-interpolation blocks of type *Kennlinie3D*

8 Conclusions

A MODELICA model *Kennlinie3D* for table interpolation in three dimensions is developed. The model is based on the MODELICA 2D-interpolation model *CombiTable2D* which gives access to an efficient interpolation routine provided by Dymola. Interpolation to four dimensions is possible with an additional interpolation level supported by the developed model *WeightedSignals*. In order to allow a large number of parameter studies a method is developed that automatically generates the required interpolation data from a complex solar field model. Due to the universal design of the models they can also be used apart from the solar field application.

Acknowledgements

The authors would like to thank the German Ministry for the Environment, Nature Conservation and Nuclear Safety for the financial support given to the ITES project under contract No. 16UM0064.

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

Apendix: Source code of model *WeightedSignals*

```
model WeightedSignals

  Modelica.Blocks.Interfaces.RealInput  x      “actual value of x”;
  Modelica.Blocks.Interfaces.RealInput  u[n]   “values at nodes x_param”;
  Modelica.Blocks.Interfaces.RealOutput y      “interpolation result”;

  parameter Real    x_param[:] "interpolation nodes”
                             // (here [0°C, 20°C, 40°C] )
  parameter Integer n=size(x_param,1) "Dimension of signal vector";
  parameter Modelica.Blocks.Types.Smoothness.Temp
      smoothness=Modelica.Blocks.Types.Smoothness.LinearSegments
      "smoothness of table interpolation”;

  parameter Real[:,:] Weight_matrix     = [x_param, diagonal(ones(n))];
  parameter Integer   Weight_columns[:] =  2:size(Weight_matrix, 2);
  Real                Weight_tableID;
  Real[n]             weight            “weights of the values u[i]”;

equation
  for i in 1:n loop
     weight[i] = dymTableIpo1( Weight_tableID, Weight_columns[i], x);
  end for;
  y = sum(  u[i] * weight[i]    for i in 1:n);

when initial() then
   // Initialize Weighting functionality
   Weight_tableID=dymTableInit(1.0,smoothness,"NoName","NoName",Weight_matrix, 0.0);
end when;

end WeightedSignals;
```

PlanarMultiBody
A Modelica Library for Planar Multi-Body Systems

Mathias Höbinger[1], Martin Otter[2]
[1]Vienna University of Technology, Austria
[2]German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany
mathias.hoebinger@gmx.at, martin.otter@dlr.de

Abstract

A new Modelica library for the modeling and simulation of 2-dimensional mechanical systems has been developed. It is based on the existing Modelica.Mechanics.MultiBody library and implements a number of simplifications and optimizations for 2-dimensional environments, which bring the advantages of a reduced complexity of the modeling process as well as a reduced computational effort. Additionally, new components are present for joints with curve-curve contact (e.g. cam follower joints). The basic approach is, to have a 1:1 mapping of packages, models and functions, if this makes sense, and specialising them to 2 dimensions.

Keywords:
Modelica, planar multi-body, contact mechanics.

1 Introduction

The *PlanarMultiBody* library is a Modelica package providing 2-dimensional mechanical components to model in a convenient way planar mechanical systems. The main design goal of the library was to utilize the fact that in such systems coordinates, directions and rotations can be expressed and computed in a much simpler way than in 3-dimensional systems.

A typical example of this library is a mechanism with 2 kinematic loops as shown in the Figure 1.

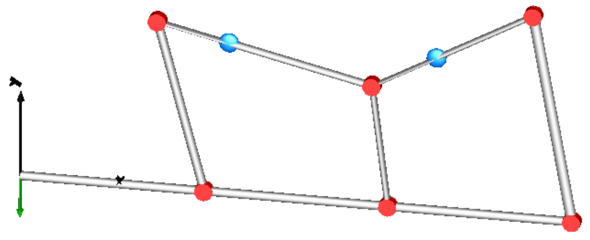
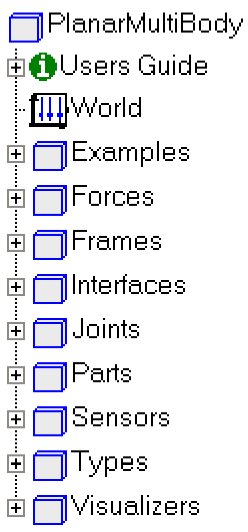


Figure 1: A planar mechanical system containing 2 coupled kinematic loops

The PlananMultiBody library, see screenshot to the right, has the following main features:



- In 2-dimensional systems, the orientation of any object with respect to another one can be described by a single angle. This simplifies the notation for orientation of objects considerably. The use of the "orientation objects" from the Modelica.Mechanics.MultiBody library can be dropped completely, as well as the special handling of the orientation object with Connections.Root (..), Connections.Branch(..) operators to define the connected network of coordinate systems in order to handle over-determined DAEs. The requirements for a Modelica translator to process models of this library are therefore much less as for the 3-dim. Modelica.Mechanics.MultiBody library.
- The visualizer objects used in the MultiBody library for the animation of objects have been altered to achieve two aims: Firstly, all animation objects can be addressed as 2D objects, e.g., the bars used to animate a fixed translation have a length and a width, no height. The *Visualizers.Advanced.Shape* object, as well as the objects used for animating all kinds of arrows, includes input values for length, width and position. Secondly, because the actual animated shapes are still 3D-objects, the height is automatically set to a very low value which gives the animation a "pseudo-planar" look.
- The possibility to model joints based on two curves sliding along each other. In model PlanarMultibody.Joints.CurveCurveJoint, different curve objects can be selected. They all contain functions used to compute three vectors depending on a curve parameter s: the *curvePosition,*, the *curveTangent* and the *curveNormal.* The Planar-

MultiBody.Joints.CurveCurveJoint object includes two instances of arbitrary curve objects, each connected to a frame. This joint constrains the movement of its two frames by requiring proper contact conditions for the two curves. These are computed using the two curve parameters *s1* and *s2*. Additional curves needed by a user can easily be added by just providing the necessary equations of the curve and its normal and tangent vectors.

2 Describing Orientation

The simplified way of describing absolute and relative orientation of objects is the most significant improvement for modeling planar systems compared to model the same system using the 3-dimensional MultiBody library. For notational convenience the word "frame" is used in the sequel as a synonym for "coordinate system". Instead of using three orthogonal unit vectors to define a specific frame we can do that with a single angle φ that describes the rotation of that frame with respect to the global coordinate system around the only possible axis of rotation, the z-axis. To define the position and rotation of a second frame relative to the first one is equally simple: a two-dimensional vector *r_rel* and a relative angle *φ_rel* are everything that is needed. Given the absolute angles *φ_a and φ_b* of two different frames, the relative angle can be computed by simply stating $\varphi\_rel = \varphi\_b - \varphi\_a$.

3 PlanarMultiBody Frame Connector

The "Frame" connector is used to connect planar multibody components together. It is rigidly fixed at an attachment point of a mechanical part. A frame "frame a" is described with respect to the world frame using the

- 2-element vector *r* that is directed from the origin of the world frame to the origin of frame a and is resolved in the world frame and by the
- angle φ between the x-axis of the frame and the x-axis of the world-frame.

It is assumed that a free body diagram is constructed, i.e. that a cut is performed between mechanical parts that shall be connected together at frame a. In the cut plane a resultant cut force $\mathbf{f}_a$ and resultant cut torque τ_a act on frame a. Since in planar multi-body systems there are no advantages to express vectors in local frames, all vectors, and especially $\mathbf{f}_a$, are expressed in the world-frame. The resultant cut torque is a scalar along the z-axis of the world-frame. To summarize, the connector is defined as:

```
connector Frame
  import SI = Modelica.SIunits;
  SI.Position r[2]
     "Absolute position vector";
  SI.Angle phi
     "Angle from x-axis world to frame";
  flow SI.Force  f[2]
     "Constraint force in world frame";
  flow SI.Torque t
     "Constraint torque in world frame";
end Frame;
```

As usual, if velocities or accelerations are needed, they can be obtained by applying the derivative operator *der(...)*. This also holds for the angular velocity which is simply *der(phi)*, where as in the Modelica.Mechanics.MultiBody library the computation of the angular velocity is complicated and is performed with a function.

4 Elementary Components

Using the "Frame" connector and the utility functions in PlanarMultiBody.Frames, it is straightforward to implement the elementary components that are usually available in multi-body programs. The PlanarMultiBody library has about 40 components. The most important ones are shown in Table 1. Exactly like in the Modelica.Mechanics.MultiBody library, equations are only defined on "position" level.

4.1 PlanarMultiBody.World

This model represents a global coordinate system fixed in ground. It is used as inertial system in which the equations of all elements of the PlanarMultiBody library are defined and is the world frame of an animation window in which all elements of the PlanarMultiBody library are visualized. Furthermore, the gravity field of the multi-body model is defined here. Default is a uniform gravity field; a point gravity field can also be selected. The world object is also used to define default settings of animation properties (e.g. the width of the rectangles representing a revolute joint). The world object itself is animated as a coordinate system with 2 axes and labels.

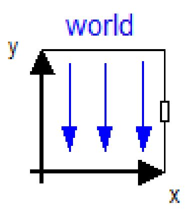
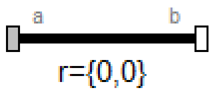
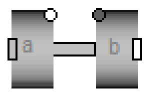
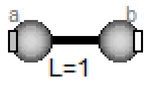
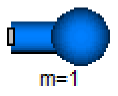
Abbreviations: $\mathbf{r}_a,\varphi_a,\mathbf{f}_a,\tau_a$:= frame_a.r, .phi, .f, .t $\mathbf{r}_b,\varphi_b,\mathbf{f}_b,\tau_b$:= frame_b.r, .phi, .f, .t resolve1(..) := Frames.resolve1(..) grav := world.gravityAcceleration(..)	
World 	$\mathbf{r}_b = \mathbf{0}$ $\varphi_b = 0$
Parts.Fixed Translation 	$\mathbf{r}_b = \mathbf{r}_a + \text{resolve1}(\varphi_a, \mathbf{r}_{rel})$ $\varphi_b = \varphi_a$ $0 = \mathbf{f}_a + \mathbf{f}_b$ $0 = \tau_a + \tau_b + \mathbf{r}_{rel} \times \mathbf{f}_b$
Joints.Revolute 	$\mathbf{r}_b = \mathbf{r}_a$ $\varphi_b = \varphi_a + \varphi_{rel}$ $\mathbf{0} = \mathbf{f}_a + \mathbf{f}_b$ $0 = \tau_a + \tau_b$
Joints.JointRR 	$\mathbf{r}_{rel0} = \mathbf{r}_b - \mathbf{r}_a$ $L*L = \mathbf{r}_{rel0}* \mathbf{r}_{rel0}$ $\mathbf{0} = \mathbf{f}_a + \mathbf{f}_b$ $\mathbf{f}_a = f_{rod} * \mathbf{r}_{rel0} / L$ $0 = \tau_a$ $0 = \tau_b$
Parts.Body 	$w = \text{der}(\varphi_a)$ $z = \text{der}(w)$ $\mathbf{r}_{CM0} = \text{resolve1}(\varphi_a, \mathbf{r}_{CM})$ $\mathbf{r}_{absCM0} = \mathbf{r}_a + \mathbf{r}_{CM0}$ $\mathbf{g} = \text{grav}(\mathbf{r}_{absCM0})$ $\mathbf{v} = \text{der}(\mathbf{r}_{absCM0})$ $\mathbf{a} = \text{der}(\mathbf{v})$ $\mathbf{f}_a = m * (\mathbf{a} - \mathbf{g})$ $I * z = \tau_a - \mathbf{r}_{CM0} \times \mathbf{f}_a$

Table 1: Elementary components of PlanarMultiBody.

4.2 PlanarMultiBody.Parts.FixedTranslation

This component defines a fixed translation of a frame. It is, e.g., used to define frames for several attachment points on a body. The equations state that the position vector of frame_b is defined from the position vector of frame_a and the relative position vector $\mathbf{r}_{rel}$ from frame_a to frame_b ($\mathbf{r}_{rel}$ is defined as parameter "r"). Since frames are translated, the angles in the two frames are set equal. Finally, a force and torque balance of this massless part is present in the Modelica model.

4.3 PlanarMultiBody.Joints.Revolute

In planar systems, the only possible axis of rotation is the z-axis, so this component always defines such a rotation using a vector φ_{rel}. When $\varphi_{rel} = 0$, frame_a and frame_b coincide. Unlike in the Modelica.Mechanics.MultiBody library, the absolute orientation vector of frame_b, frame_b.phi, can easily be obtained by stating

frame_b.phi = frame_a.phi + φ_rel.

As with most other joints, the generalized coordinates (here: *φ_rel* and its derivative *ω_rel*) have the attribute *stateSelect = StateSelect.prefer* in order that they are selected as states if possible. The position vectors of the two frames are identical and there is a force and torque balance present. Instead of implementing an additional model "ActuatedRevolute", a conditional 1-dim. flange connector is present onto which a drive train can be attached driving the revolute joint, e.g, with components from the Modelica.Mechanics.Rotational library. There is a Boolean parameter *drivenFlange* present to activate or deactivate the additional flange.

4.4 PlanarMultiBody.Parts.Body

This component defines the mass and inertia properties of a body. They are defined using the following parameters: *m* for the mass, the position vector *r_CM* from the origin of frame_a to the center of mass (resolved in frame_a) and the inertia value *I*. There is a Boolean parameter *enforceStates* present which defines if the positon vector *r* and orientation angle *φ* of frame_a should be use as states. These variables have the attribute *stateSelect = if enforceStates then StateSelect.always else StateSelect.avoid.* The feature to have potential states both in joints and in bodies makes it easier to model systems with bodies which are connected to the environment without using a joint or freely moving bodies.

4.5 PlanarMultiBody.Joints.JointRR

This component fixes the distance between its two frames to parameter *L*, but does not constrain the orientation angles of any of them. Therefore it can be used as a replacement for two revolute joints connected by a fixed translation. Using this component reduces the order of the nonlinear equation system and helps avoiding problems with non-linear equation systems caused by kinematic loops. The cut force is constrained to act only along the vector between the origins of the two frames. Finally, a force and torque balance is present in this component. There is an additional object called PlanarMultiBody.Joints.JointRRWithMass present which includes a mass fixed relative to the two frames of the joint.

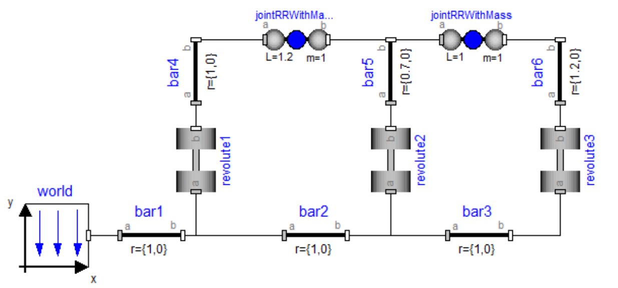


Figure 2: The diagram level of the model animated in Figure 1 using two instances of JointRRWithMass

5 Force Elements

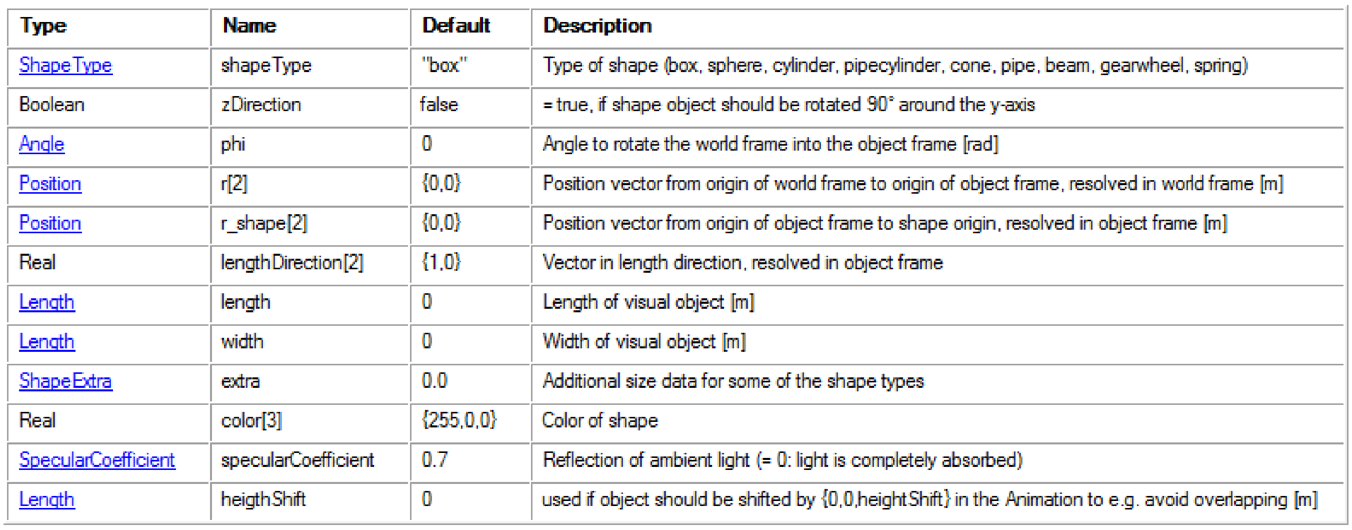
Force elements exert forces and torques between two frames. Because these elements, although they have obviously been altered to fit into the different orientation setup of this new library, are virtually identical in their functionality and structure to the ones in the MultiBody library, we will not discuss them here in great detail. For a more detailed description of the most important force elements, see [1].

6 Animation

The animation environment in Dymola [2] is natively a 3-dimensional one, and all animated objects therefore have to be programmed in that way. However, the Modelica.Mechanics.MultiBody library utilizes a single model to realize virtually of all its animations, MultiBody.Visualizers.Advanced.Shape. The following features were implemented into the PlanarMultiBody animation engine:

- Having a user-interface with purely 2-dimensional animation parameters gives the user the convenience of not having to deal with a z-coordinate that only exists in the animation and has nothing to do with the planar system being modeled.
- To provide users with a maximum of freedom of design, either side of a 3d-object displayed by the "FixedFrame" component of the library can be used as a "pseudo-2d" object. E.g. a cylinder can be used as a circle or a rectangle. For this purpose, a boolean parameter "zDirection" was added to the Shape object which rotates the animated object by 90° around the y-axis.
- To avoid overlapping of objects in the "pseudo-2D" animation, it is possible to shift an object along the z-axis of the animation using the parameter "heigthShift".
- The heigth of all objects is automatically set to a low value which results in the desired "pseudo-2D" look of the animation.

Table 2 shows all the parameters of the PlanarMultiBody.Visualizers.Advanced.Shape object with their default values and a short description of their functionality.



Type	Name	Default	Description
ShapeType	shapeType	"box"	Type of shape (box, sphere, cylinder, pipecylinder, cone, pipe, beam, gearwheel, spring)
Boolean	zDirection	false	= true, if shape object should be rotated 90° around the y-axis
Angle	phi	0	Angle to rotate the world frame into the object frame [rad]
Position	r[2]	{0,0}	Position vector from origin of world frame to origin of object frame, resolved in world frame [m]
Position	r_shape[2]	{0,0}	Position vector from origin of object frame to shape origin, resolved in object frame [m]
Real	lengthDirection[2]	{1,0}	Vector in length direction, resolved in object frame
Length	length	0	Length of visual object [m]
Length	width	0	Width of visual object [m]
ShapeExtra	extra	0.0	Additional size data for some of the shape types
Real	color[3]	{255,0,0}	Color of shape
SpecularCoefficient	specularCoefficient	0.7	Reflection of ambient light (= 0: light is completely absorbed)
Length	heigthShift	0	used if object should be shifted by {0,0,heightShift} in the Animation to e.g. avoid overlapping [m]

Table 2: Parameters of the PlanarMultiBody.Visualizers.Advanced.Shape object

7 Curve-Curve Contact

With Joints.CurveCurveJoint, the PlanarMultiBody library includes a new joint making it possible to simulate two surfaces having to remain in contact with each other. In every instance of this joint, the user can choose two out of a library of curves used to simulate the connected surfaces. Each curve is fixed to one frame of the joint, in the sequel we will use the name curve_1 for the curve object connected to frame_a and curve_2 for the one connected to frame_b. The main idea is to have two variables s_1 and s_2, one for each curve, in the CurveCurveJoint model, which stand for the path parameter of the respective curve, describing the current contact point on the curve with respect to a fixed starting point. Usually “s”is the arc-length along the curve, but this need not to be the case in general. For a given value of their respective curve-variables, curve_1 returns a relative position vector from frame_a to the point of contact as well as the normal and tangent vector at that point on the curve.

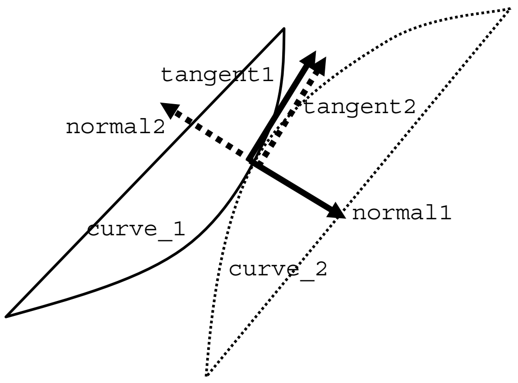
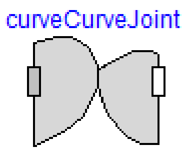


Figure 3: Normal and tangent definition of curve-curve contact

7.1 Joints.CurveCurveJoint



As mentioned above, this model includes two frames as well as two instances of a “curve object”. The possibility of choosing the curves inside the actual instance of the joint is realized by including them as “replaceable” objects:

```
replaceable Joints.Internal.Circle
  curve1(phi=frame_a.phi,r_0=frame_a.r)
    extends
      PlanarMultiBody.Interfaces.BaseCurve
           (phi=frame_a.phi, r_0=frame_a.r)
```

In the equation section of the CurveCurveJoint model, position, normal and tangent variables are connected to the respective variables in the curve objects.

```
r1_rel = curve1.position(s1);
r2_rel = curve2.position(s2);
r1 = Frames.resolve1(frame_a.phi,r1_rel);
r2 = Frames.resolve1(frame_b.phi,r2_rel);
normal1  = Frames.resolve1(frame_a.phi,
                        curve1.normal(s1));
normal2  = Frames.resolve1(frame_b.phi,
                        curve2.normal(s2));
tangent2 = Frames.resolve1(frame_b.phi,
                       curve2.tangent(s2));
```

More importantly, the kinematic constraint equations as well as the force and torque balances of the joint and the curves are defined here:

First, the distance between the contact point on curve_1 and the one on curve_2 is set to zero:

```
{0,} = frame_b.r + r2 - (frame_a.r + r1);
```

Then, additional equations ensure that the contact point is actually an osculation point of the two curves, meaning that their standard normal vectors point in the same direction with different signs:

```
0 = Modelica.Math.atan2(
      normal1*tangent2, -normal1*normal2);
```

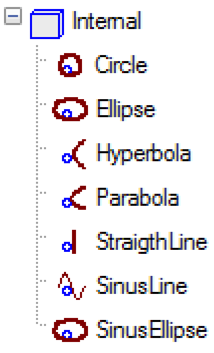
The formulation of this condition is from Hans Olsson [3] and requires some explanation: The contact conditions on the normal could be formulated as “normal1*normal2=0”. However, this equation has a singular Jacobian and therefore every solver would have severe difficulties. The condition could also be formulated as “normal1*tangent2 = 0”, as often suggested in literature. Here, we have the problem that a contact where normal1 and normal2 are directed in the same direction, will also fulfill this equation and therefore it can happen that during simulation suddenly a wrong contact appears. The formulation used in the CurveCurveJoint is basically using the “normal1*tangent2 = 0” formulation, but uses this as the first argument to the “atan2(..)” function. As second argument “-normal1*normal2” is used. The “atan2(..)” function has the property that the signs of the two arguments determine the quadrant of the solution. Especially, only if the second argument is positive, $-\pi/2 <= \text{atan2}(x,y) <= \pi/2$. Therefore, in the solution point “0 = atan2(x,y)”, the second argument “-normal1*normal2” must be positive which means that the two normal’s have to be directed in opposite direction.

Finally, force and torque balances are included:

```
// Force and torque balance of joint
zeros(2) = frame_a.f + frame_b.f;
0 = frame_a.t + frame_b.t +
     Frames.cross(frame_b.r - frame_a.r,
                  frame_b.f)

// Force and torque balance of curve1
f_contact1 = -normal1*f_N;
  zeros(2) = frame_a.f + f_contact1;
  0 = frame_a.t +
      Frames.cross(r1, f_contact1);
```

7.2 Predefined Contact Curves



The package PlanarMultiBody.-Joints.Internal includes the models which are predefined in the CurveCurveJoint object. Additional curves can easily be added by a user. We will use the Ellipse model to explain the functionality of these objects. All curve-definition models extend a model called PlanarMultiBody.Interfaces.BaseCurve which defines the basic input variables *r_0* and *phi* which are the absolute position vector and orientation angle of the frame to which the curve is attached.

The BaseCurve model also establishes the three functions *position*, *normal* and *tangent* and their basic input and output variables. The input variable *s* is the curve parameter; the 2-dimensional output vector is called *r, n* or *t* depending on the function. To enable the different curve-definition models to have different versions of these functions, they are defined as "replaceable encapsulated partial functions" in BaseCurve.

```
replaceable encapsulated partial
  function normal
    input  Real s    "Curve parameter";
    output Real n[2] "Normal to curve";
end normal:
```

Every curve model has its own set of parameters used to adjust the actual curve surface In case of the ellipse there are two of them: *a* and *b*, defining the length of the two ellipse-axis.

Furthermore, there is always at least one parameter defining the path parameter of the animated curve. In case of the ellipse, the final parameter *C* is the approximated circumference of the Ellipse computed from the given parameters *a* and *b*. In the models which define non-closed curves, e.g. "StraightLine", there is an input parameter instead of this final parameter allowing the user to define how long a part of the curve should be animated.

Additionally, there are the usual animation-concerned parameters *animation*, switching the animation of the curve on or off, and *color*, defining the color of the animated curve. Finally, the parameter *ns* defines how many points should be used to interpolate the animated curve and the *SwitchSide* parameter defines on which side of the curve the contact should occur.

The most important part of a curve-definition model are of course the three functions actually defining the shape of the curve: *curvePosition*, *curveNormal* and *curveTangent*. They extend the respective functions in the BaseCurve model by including the necessary additional parameters and adding an "algorithm" section with the statement computing their output variable. Here we present the CurvePosition function from the Ellipse model as an example:

```
model Ellipse "Ellipse contact curve"
  extends
   PlanarMultiBody.Interfaces.BaseCurve(
    redeclare final function position =
             curvePosition(a=a,b=b,C=C),
    redeclare final function normal =
         curveNormal(a=a,b=b,C=C,sw=sw),
    redeclare final function tangent =
             curveTangent(a=a,b=b,C=C));
protected
  function curvePosition
    extends PlanarMultiBody.Interfaces.
                     BaseCurve.position;
    input Modelica.SIunits.Length a
          "Length of a-axis of ellipse";
    input Modelica.SIunits.Length b
          "Length of b-axis of ellipse";
    input Modelica.SIunits.Length C
           "Approximated circumference";
  algorithm
    r := { a*sin(s*2*pi/C),
          -b*cos(s*2*pi/C)};
  end curvePosition;
 ...
end Ellipse;
```

Finally, the model includes an algorithm computing the points used to animate the curve in its current position defined through the curve parameter. This is done by filling three coordinate vectors with length *ns*. These vectors are actually realized as *ns**2 matrices, the second columns being filled with slightly shifted values to ensure better visibility of the animated curve. The animation is performed with Dymola's built-in support for parameterized surfaces.

```
  final parameter Real s_min=0
                  "Minimum value of s";
  final parameter Real s_max=C
                   "Maximum value of s";
algorithm
  for i in 1:ns loop
    s := s_min + (i - 1)*
               (s_max - s_min)/(ns - 1);
    r := Frames.resolve1(phi,
                     position(s));
    x[i,1] := r_0[1] + r[1];
    x[i,2] := r_0[1] + r[1] + 0.01;
    y[i,1] := r_0[2] + r[2];
    y[i,2] := r_0[2] + r[2] + 0.01;
    z[i,1] := 0;
    z[i,2] := 0.01;
  end for;
```

7.3 Examples

Package PlanarMultiBody.Examples.CurveCurve-Joint includes a number of examples demonstrating the use of this new joint. The most obvious example is probably the classic Cam-Follower setup. In this model, an elliptic object driven by gravity acting upon a body attached to it turns on a revolute joint fixed to the ground. It is connected to an object with a straight surface being attached to a prismatic joint and forced into movement by the ellipsoid (see model schematic und animation in next Figure 4).

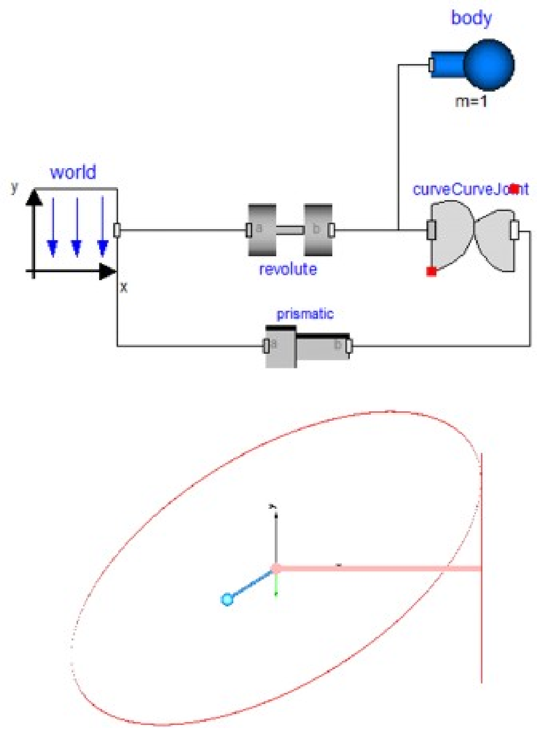
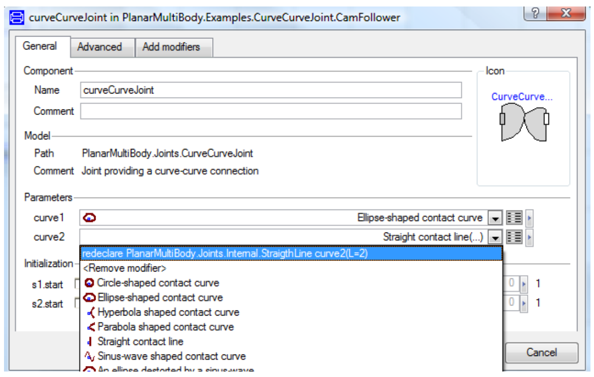


Figure 4: Model and animation of CamFollower

It it realized by connecting frame_a of a CurveCurveJoint to the world frame through a revolute joint and doing the same with frame_b using a prismatic joint. Then the appropriate curves have to be selected by double clicking on the joint and selecting them from a dropdown menu (see next Figure 5).


curveCurveJoint in PlanarMultiBody.Examples.CurveCurveJoint.CamFollower

Component
Name: curveCurveJoint
Comment:

Model
Path: PlanarMultiBody.Joints.CurveCurveJoint
Comment: Joint providing a curve-curve connection

Parameters
curve1: Ellipse-shaped contact curve
curve2: Straight contact line(...)

redeclare PlanarMultiBody.Joints.Internal.StraigthLine curve2(L=2)

Initialization
s1.start: 0 1
s2.start: 0 1
General
Advanced
Add modifiers
Icon
CurveCurve...
<Remove modifier>
Circle-shaped contact curve
Ellipse-shaped contact curve
Hyperbola shaped contact curve
Parabola shaped contact curve
Straight contact line
Sinus-wave shaped contact curve
An ellipse destorted by a sinus-wave
Cancel

Figure 5: Selecting a curve in the CurveCurveJoint menu

Finally, a body is attached to frame_a of the joint and the start value of the ellipses curve parameter is set to an appropriate value to ensure that the system is not in an idle position at time 0.

Another example from this package demonstrates the effect of the *switchSide* parameter, see Figure 6. Two CurveCurveJoint objects are present, both describing the contact between two circles. In the upper circle-circle contact, switchSide = **true**, whereas in the lower circle-circle contact, the default switchSide = **false** is used. The effect can be seen in Figure 6.

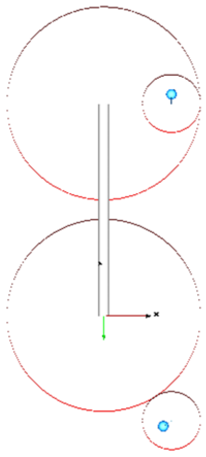


Figure 6: Example CurveCurveJointSwitchSides demonstrating the switchSide parameter

The third example, see Figure 7, shows the possibility of more complex curves by using an ellipse distorted by a sinus wave. This curve has amplitude and frequency of the wave as additional parameters. Here, a very small circle attached to a small body runs along the distorted ellipse. It is connected to the world frame using a prismatic joint.

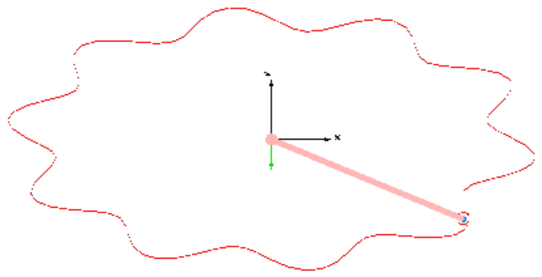


Figure 7: Example SinusEllipse demonstrating more complicated curve-curve contacts

8 Conclusions

The PlanarMultiBody library is a mechanical library to model planar mechanical systems. The main advantage is its simplicity and that no special symbolic manipulation features of the Modelica simulation environment is needed, contrary to the Modelica.Mechanics.MultiBody library that describes 3-dim. mechanical systems. Therefore, the PlanarMultiBody library is well suited for teaching, but also for a quite large class of technical problems that are 2-dim. in nature. Besides standard joints, the PlanarMultiBody library allows the definition of curve-curve contacts, especially to describe cam-follower types of contact. The non-standard formulation [3] of the contact condition with the atan2(..) function has proven to result in reliable solutions of the occurring non-linear algebraic equation systems.

It is planned to include this library as free package in the Modelica Standard Library after an evaluation phase. Currently, there is also an Interpolation package under development. Once available, it is planned that the curve descriptions in the curve-curve contact description can be optionally described by splines of this package.

References

[1] Otter M., Elmqvist H., Mattson S.E. **The new Modelica Multibody Library**. Proc. of the 3rd International Modelica Conference, pp. 311-330, 2003. http://www.modelica.org/Conference2003/papers/h37_Otter_multibody.pdf

[2] Dynasim. **Dymola Users Guide**, Version 6.0, http://www.dynasim.se.

[3] Olsson H.: **Formulation of contact conditions**. Personal communication to M. Otter, Sept. 2007.

Implementation of Hybrid Electric Vehicles using the VehicleInterfaces and the SmartElectricDrives Libraries

Dragan Simic Thomas Bäuml
Arsenal Research
Giefinggasse 2, 1210 Vienna, Austria

Abstract

In this paper different configurations of hybrid electric vehicles summarized in the *SmartHybridElectricVehicles* library were examined and simulated. The presented simulation models and results were created and achieved with *Modelica* using *Dymola*. The models represent different kinds of electric and hybrid electric vehicle configurations. Furthermore, different strategies for operating the hybrid electric vehicles energy sources are provided. The parameters needed for parameterization of the vehicle models were, in case of the electric vehicle, taken from real measurements on the vehicle and vehicle components. For all other models parameters were assumed due to a lack of measurement data. In the library three *Modelica* packages specifically designed for modeling systems including mechanical components, electrical components and control components have been used. These are the *SmartElectricDrives* library, the *VehicleInterfaces* library and the *PowerTrain* library. Due to the object oriented architecture of these libraries all necessary components needed for the implementation and simulation of electric and hybrid electric vehicle configurations are provided and can be reused. Hence, the efficiency optimization of such configurations gets eased by these libraries.

Keywords: simulation, modeling, hybrid electric vehicles, optimization, fuel consumption, operating strategy

1 Introduction

In this contribution a simulation library, the *SmartHybridElectricVehicles* (SHEV) library, will be presented. This library is developed by *arsenal research* with focus on automotive applications, such as electric and hybrid electric vehicles (HEV). The SHEV library is written in *Modelica* language [1] and simulated using the *Dymola* simulation environment. The library is implemented on the basis of the *VehicleInterfaces* (VI) library [2]. Therefore compatibility with all other libraries based on the VI library is ensured. For simulations of the electrical components the *SmartElectricDrives* (SED) library [3] is used. The *StateGraph* library, included in the *Modelica Standard Library* (MSL), has been chosen for modeling the operating strategies of the included vehicles. All mechanical components, such as the power train including transmissions, differentials, axles, etc. are provided by the *PowerTrain* library.

2 Electric Vehicle

An electric vehicle using the above mentioned libraries was modeled as depicted in figure 1. This configuration consists of a front axle modeled in the `driveline` model, a transmission (`trans.`) with one gear and an electric machine (`MG2`). Attention is paid to the energy consumption during a simulated drive cycle. Therefore the quasi stationary model of an electrical excited DC machine with integrated converter and control system, including voltage and current limitation as well as flux weakening from the SED library is used here. For powering the vehicle, an energy source (`battery`) is modeled using a simple idealized battery model included in the SED. This battery model consists of a constant capacitor and a constant internal resistor only. All mechanical components, such as `brakes`, `chassis` and `driveline` are taken from the *PowerTrain* library. They are provided there as ready to use models. For controlling the vehicle velocity (acceleration pedal and brake pedal position) a virtual `driver` model taken from the *PowerTrain* library was adapted. In the controller model (`control.`), different operating strategies are implemented.

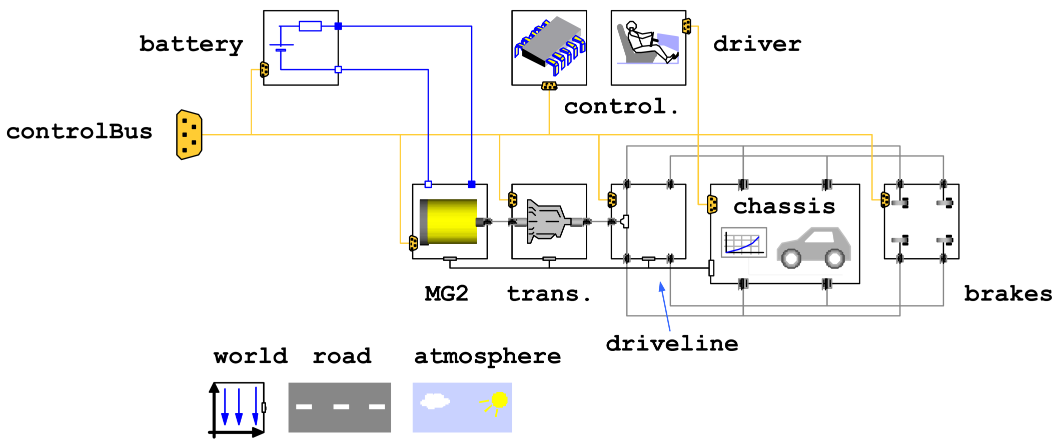


Figure 1: The *Modelica* simulation model of the electric vehicle

2.1 Operating strategies

Three different operating strategies are implemented in the controller of the electric vehicle. These strategies are modeled using the *StateGraph* library of the MSL. All operating strategies control the reference torque of the electric machine. In the first case reference torque of the electric machine is limited between maximum machine torque and zero. The reference torque is restricted to be positive only. In this first operating strategy, only the drive mode of the electric machine is active, no recuperation occurs.

The second operating strategy includes the basic functionality of the first operating strategy with an additional recuperation mode. When the virtual driver actuates the brake pedal, the electric machine is driven in generator mode and the battery is recharged. The reference torque of the electric machine is directly proportional to the brake pedal position. Additionally, vehicle deceleration occurs by mechanical braking.

The third operating strategy is split into two braking mode levels. During the first stage, vehicle deceleration occurs by electrical braking and recuperation only. The battery is charged. If the demanded reference braking torque exceeds the electric machines maximum torque, additional mechanical braking occurs.

In the last two strategies electrical braking and hence electrical recuperation only occurs if the battery state of charge decreases beneath a certain limit. By reaching the upper set limit, electrical braking is switched off to prevent overloading and damaging the battery.

The model of the electric vehicle in figure 1 was simulated with all three operating modes in the *New European Drive Cycle* (NEDC). The state of charge (SOC) of the battery was compared and is shown in figure 2. $\texttt{mode}_1$ represents the first operating strategy without recuperation, $\texttt{mode}_2$ the second operating strategy with proportional recuperation and mechanical braking and $\texttt{mode}_3$ the third implemented operating strategy. Due to a high recuperation ratio $\texttt{mode}_3$ is the strategy with the lowest energy consumption and the highest recuperation potential, respectively. Mainly electrical braking occurs and therefore the battery is recharged more than in any other implemented strategy.

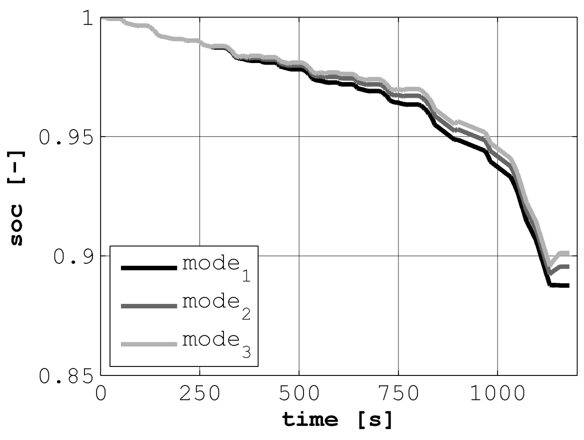


Figure 2: Simulated state of charge of the battery during different operating strategies

3 Series Vehicle

The series hybrid electric vehicle depicted in figure 3 is modeled based on the electric vehicle model. It contains an additional internal combustion engine (ICE),

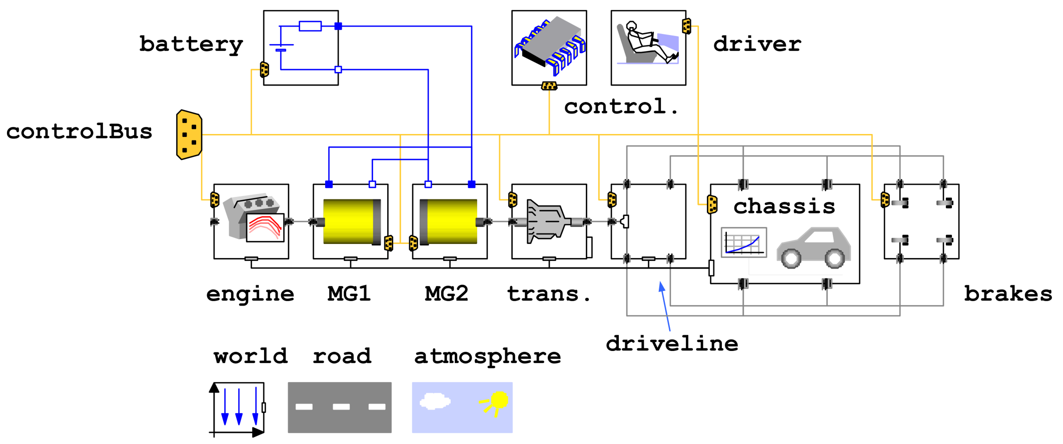


Figure 3: The *Modelica* simulation model of the series hybrid electric vehicle

`engine`, and an electric machine acting as generator, `MG1`. The generator is driven by the ICE and is used to charge the battery. The operating maps and the fuel consumption of the ICE are taken from a *Toyota Prius*, according to [4]. The basic drive modes for the `MG2` are taken from the electric vehicle. Additionally, different operating strategies for the ICE and the MG1 have been implemented. With the disabled recuperation mode of the MG2 and a disabled generator MG1, the behaviour of the series vehicle is the same as the electric vehicle. For operating the MG1 a shift of the ICE operating point is implemented. It is dependent on the demanded electrical power and, hence, the required torque and speed of the generator. The input value for this operating strategies are the measured motor power and the current generator power, respectively. During a change of the demanded generator power the strategy calculates the most efficient operating point of the ICE regarding fuel consumption. In figure 4 the shifting between two `operating points` with different demanded generator power is depicted. τ_{max} is the maximum torque and τ_{min} is the drag torque of the ICE. These two operating points of the ICE are those with the highest efficiency and the lowest fuel consumption, respectively. The operating strategy is modeled in the controller (`control.`) block and based on different control algorithms that will not be described here in detail. The control is independent of size and type of the electric machine as well as of the size of the ICE, which means, that any kind of ICE or machine can be included in the model. Currently the user can choose between two engines and various transient and quasi stationary electric machines in different power classes.

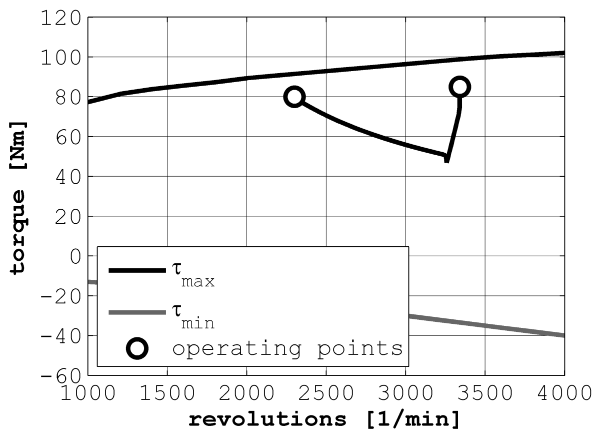


Figure 4: Operating point shift of the internal combustion engine

4 Parallel Vehicle

The parallel HEV, figure 5, contains an ICE, `engine`, and an electric machine acting as starter/generator, `MG1` with two shaft ends. This electric machine is used for starting the ICE, for boosting during driving mode and for recharging the battery. The electric machine, `MG1`, is coupled on one side with the ICE by a mechanical clutch, `C1`, and on the other side with the transmission by a mechanical clutch, `C2`. The mechnical clutch, `C2`, is embedded in the transmission model (`trans. + C2`). Using this kind of power train configuration, it is possible to switch between more driving modes. Potential driving modes are driving with the electric machine only, driving with engine and electric machine (ICE and boosting electric machine), start/stop operation of the ICE, load point shifting of

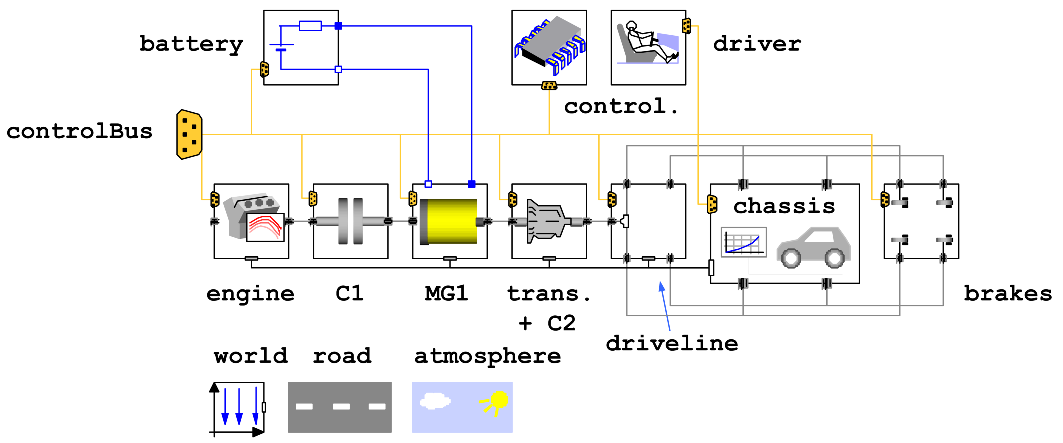


Figure 5: The *Modelica* simulation model of the parallel hybrid electric vehicle

the ICE and recuperation during vehicle deceleration.
Exemplarily, two operating strategies for the ICE and the starter/generator are simulated and shown here. The first operating strategy demonstrates the basic operating strategy of a conventional vehicle, only driven by the ICE without recuperation or start/stop operation. The second operating strategy manages the start/stop driving operation of the ICE and the starter/generator. The comparison of the fuel consumption of the ICE is depicted in figure 6, where $\Sigma_{conventional}$ is the fuel consumtion of conventional driving and $\Sigma_{start/stop}$ is the fuel consumption during start/stop operation, respectively. Both vehicle models are simulated in an NEDC operating cycle. The SOC of the battery is balanced in both models at start and end of the simulation, figure 7. One can see, that the state of charge of the conventional vehicle remains unchanged, because no electrical driving or boosting occurs. By contrast the SOC during start/stop operation shows slight changes. During standstill the engine is switched off. By activating the acceleration pedal, the engine is started by the electric machine. While accelerating the engine, the SOC decreases until the engine has reached idle speed. Then the electric machine switches to recuperation mode and the battery is recharged to the upper set limit. Due to a fuel saving during standstill, the vehicle with start/stop operating mode shows a slightly lower fuel consumption as the conventional vehicle.

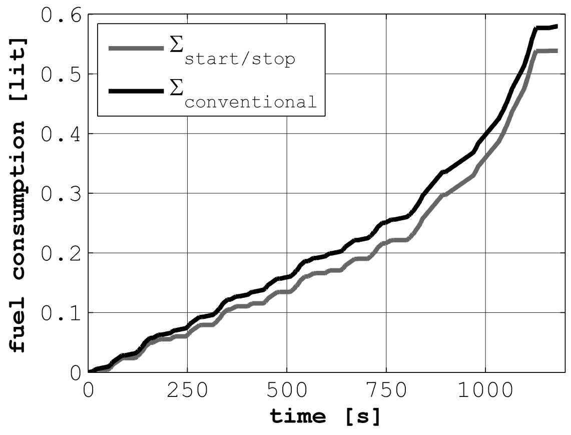


Figure 6: Comparison of the fuel consumption of conventional and parallel hybrid electric vehicle

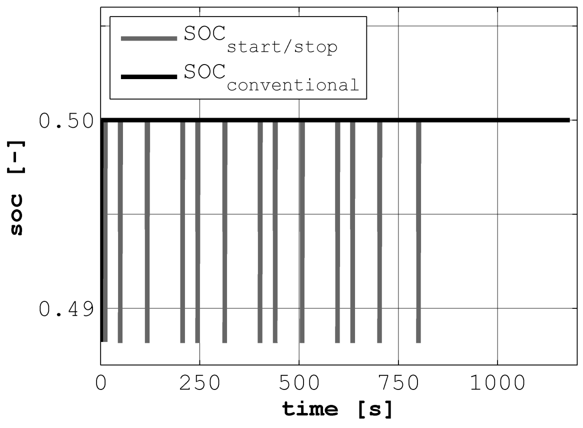


Figure 7: Comparison of the battery SOC of conventional and parallel hybrid electric vehicle

5 Electric Vehicle Validation

For validation of the HEV models and the SHEV library the electric vehicle was used in a first step, be-

description	*value*	*unit*
vehicle mass	1625	kg
front area	2.653	m²
wheel radius	0.285	m
inertia of electric machine	0.2	kgm²
final gear ratio	7.35	-
aerodynamic resistance coefficient	0.407	-
rolling resistance coefficient	0.0144	-

Table 1: Parameters of the chassis model and driving resistances derived from measurements

cause measurements on and electric vehicle could be accomplished easily. A *Citroën Belingo electrique* vehicle was chosen for validation, according to [5]. After determination of the component parameters, all single components and the entire electric vehicle model were parameterized. Afterwards simulation results were gathered and compared with measurement results of the real vehicle.

5.1 Parameterization

Every component of the electric vehicle model needs a set of parameters which have to be determined prior to the simulation. They have been derived from numerous measurements on all mechanical and electrical components and data sheets. The data sheet for the electric machine is taken from a *Peugeot Partner Electric* vehicle which has the same as the *Citroën Berlingo Electrique*, according to [6]. For the parameterization of the chassis model and the driving resistances, freewheeling curves of the electric vehicle were determined. Out of these measurements parameters listed in table 1 were calculated and used for the simulation.

For a detailed battery simulation a dynamic battery model was developed at arsenal research, whereas for the simulation and validation of the entire electric vehicle power consumption the more simplified idealized model was used. The parameterization of both battery models, linearized and dynamic, is based on measurements on the real vehicle battery using a standardized charging/discharging test cycle. Throughout this investigation it was possible to determine the parameters of the battery.

The electric machine as described in the data sheet according to [6] was parameterized with the values listed table 2.

description	*value*	*unit*
nominal armature voltage	162	V
nominal armature current	110	A
nominal excitation current	12.5	A
nominal rotor speed	1340	rev/min
maximum rotor speed	6500	rev/min
warm armature resistance	0.069	Ω
armature circuit inductance	0.00169	H
warm excitation resistance	9.47	Ω
excitation circuit inductance	0.0947	H

Table 2: Parameters of the electric machine according to the Peugeot Partner data sheet

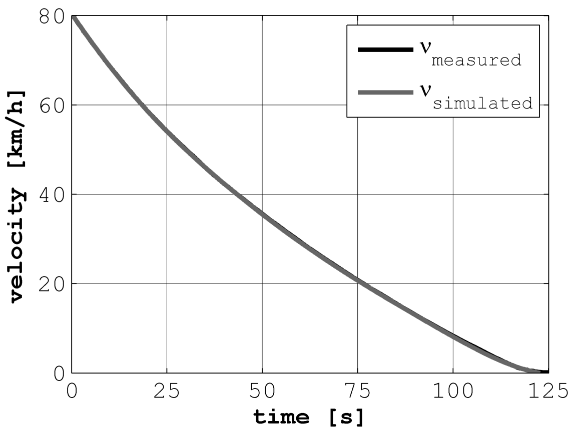


Figure 8: Comparison of the measured and simulated freewheeling curve of the electric vehicle

5.2 Model Validation

The validation of the electric vehicle model was executed first on component level and then regarding the complete vehicle. All mechanical and geometrical parameters, the electrical parameters of the electric machine and the battery as well as the overall power consumption of the entire electric vehicle were determined. The vehicles driving resistances such as aerodynamic and rolling resistances have been calculated based on the measured freewheeling curve. For validating the electric vehicle resistance model the simulated freewheeling curve is compared with the measured one in figure 8. The very small difference between the real measured and the simulated freewheeling curve allows the assumption, that the driving resistances have been chosen in an accurate way.

For validation of the vehicles power train, the electrical excited DC machine, the DCDC converter and the battery model are validated. For modeling the electric machine a torque controlled quasi stationary

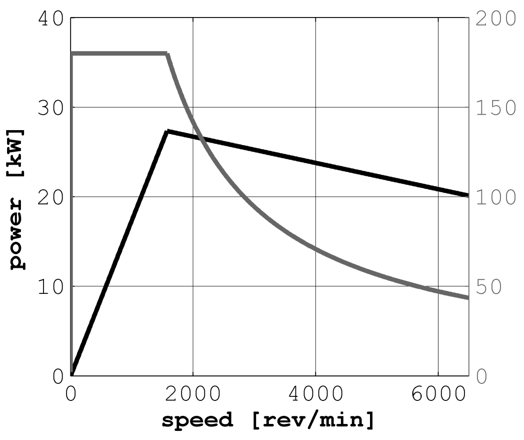


Figure 9: Power and torque curves of electric machine

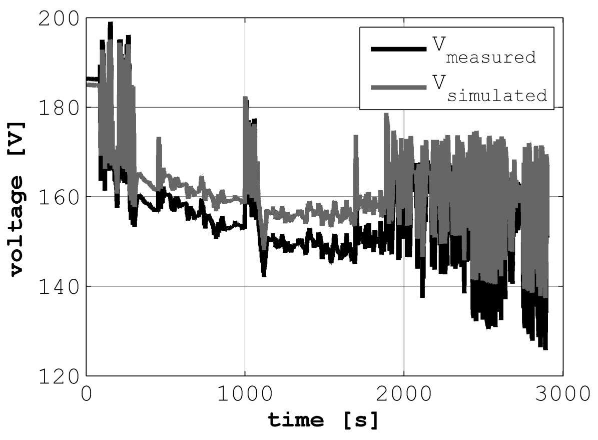


Figure 10: Mesured and simulated battery voltage

model, taken from the SED library, was used. The electric machine is driven by a reference torque and the simulation covers the entire admissible electric machine speed range. The maximum feasible inner electric torque and the mechanical output power in dependence on the electric machine speed is depicted in figure 9. This parameterization is based on the electric machine manufacturers data sheet and shows good congruence with the measured values.

Using measurement results of the voltage, current and temperature gathered during road test procedures, the complex battery model was parameterized. The measurement results were recorded during a ride through the city of Vienna, Austria. For the battery model and the entire electric vehicle validation the measured curents were used as reference signals. The measured, $V_{measured}$, and simulated, $V_{simulated}$, battery voltages are depicted in figure 10. The deviation of the voltages is assigned to the fact that some cells of the real battery were slightly damaged. Though, the overall voltage error of less then 5% is still in an acceptable bandwidth and shows the applicability of the used models.

6 Conclusions

The presented vehicle simulations allow the determination of the energy and fuel consumption as well as the identification of the economic savings potential by integrating alternative vehicle drive train concepts. Using the developed SHEV library different HEV concepts and operating strategies can be analyzed and tested very quickly . Based on the developed vehicle models different potential concepts have been identified and analyzed under different application scenarios. A significant acceleration of the development process of HEV drive train concepts and technologies can be achieved and effort can be reduced. The achievable improvements of a HEV concept highly depend on the specific driving cycle and the boundary conditions, e.g. driving time without recharging possibilities, recharging time during standstill periods, recharging during recuperation, recharging during load point shifting of the ICE operating point, etc. Therefore, these boundary conditions should be defined prior to the simulations to assure simulation results that can match the real system behaviour in a satisfying way. Furthermore, already small changes in the control strategy can have big influence on the overall energy consumption. Also these steps of development can be simulated by means of this library in a rather easy way.

References

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

[2] M. Dempsey, H. Elmqvis, 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] J.V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, and F. Pirker, “The SmartElectric-Drives Library – powerful models for fast simulations of electric drives”, *5th International Modelica Conference 2006, Vienna, Austria*, 2006.

[4] National Renewable Energy Labaratory (NREL), "ADVISOR documentation, ADVISOR data file fc prius jpn", *www.ctts.nrel.gov*, 2002.

[5] M. Noll, H. Giuliani, D. Simic, V. Conte, H. Lacher, and P. Gollob, "Simulation and optimisation of a full electric hybrid vehicle", *International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition, 22nd, EVS23, Anaheim, USA*, 2008.

[6] Peugeot, "Partner Electric Motor, Beschreibung Technischer Daten, Organummer 7725, Ref: 1238-D-04/98", *Partner Electric Motor*, 1998.

Definitions, Acronyms and Abbreviations

SHEV	*SmartHybridElectricVehicles*
HEV	hybrid electric vehicle
VI	*VehicleInterfaces*
PT	*PowerTrain*
SED	*SmartElectricDrives*
MSL	*ModelicaStandardLibrary*
NEDC	*New European Drive Cycle*
SOC	state of charge
ICE	internal combustion engine

Modeling of CO2 Reduction Impacts on Energy Prices with Modelica

Philip Machanick[1], Ariel Liebman[1], Peter Fritzson[1,2]
[1]School ITEE, University of Queensland, Australia
[2]PELAB, Department of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
aliebman@itee.uq.edu.au, philip.machanick@gmail.com, petfr@ida.liu.se

Abstract

There is growing evidence that anthropogenic carbon dioxide (CO_2) emissions as a by-product of the combustion of fossil fuels for energy use is raising the earth's temperatures and potentially leading to irreversible climate change. Additionally the growth in global emissions is likely to rise at an increasing rate due economic growth, especially in developing countries. Leading climate change mitigation strategies require a global CO_2 emission permit trading regime which is postulated to facilitate the lowest cost emission reduction options and technologies. However, given the technologies are still maturing the economic considerations appear to dictate slow initial reductions which will then grow at an increasing rate as technologies such as wind, solar and carbon capture and storage mature. These economic considerations however may be in conflict with longer-term optimization of costs and benefits, which may be better addressed by earlier intervention. In this paper we present a Modelica model designed to allow exploration of the tradeoffs between least cost emission cuts and early stabilization of atmospheric carbon dioxide.

1 Introduction

The energy and climate systems are now intimately bound through human activity. The evidence that anthropogenic carbon dioxide (CO_2) emissions as a by-product of the combustion of fossil fuels for energy use is raising the earth's temperatures and potentially leading to irreversible climate change [6]. Additionally the growth in global emissions is forecast to rise rapidly due to economic growth, especially in developing countries. In order to minimize the impacts of rising emissions on global temperatures and potentially catastrophic events such as multi-metre sea level rises deep cuts are required early [5,16].

The leading climate change mitigation strategies require a global CO_2 emission permit trading regime which is postulated to facilitate the lowest cost emission reduction options and technologies. However, given the technologies are still maturing the economic considerations appear to dictate slow initial reductions which will then grow at an increasing rate as technologies such as wind, solar and carbon capture and storage mature.

A significant question in the politics of climate change has been the trade-off between the costs of mitigation versus the costs of doing nothing. What is missing is a model quantifying the costs and benefits of the rate of of mitigation, taking into account that early strategies may be less efficient than later ones, yet have more value for mitigation if it is accepted that early mitigation is better than late mitigation, since effects accumulate.

The leading climate change mitigation strategies require a global CO_2 emission permit trading regime which is postulated to facilitate the lowest cost emission reduction options and technologies. This kind of scheme has its origin in earlier approaches to emissions reduction, such as the US Acid Rain Program, initiated by the Clean Air Act of 1990 [17], with the underlying theory of artificial markets being created to correct for market failures dating back to the late 1960s [18].

Given that the technologies are still maturing, the economic considerations appear to dictate slow initial reductions which will then grow at an increasing rate as technologies such as wind, solar and carbon capture and storage mature – hence the need not only to create an artificial market, but to explore how to use price as an instrument to drive change at the appropriate rate.

In this paper we present a Modelica model which explores the tradeoffs between least cost emission cuts and early stabilization of atmospheric carbon dioxide.

1.1 Model assumptions

1.2 The climate system

The climate model allows for either linear or exponential growth in emissions and in atmospheric carbon dioxide; current trends look linear but exponential growth may occur in the worst case if growth in energy use tracks population growth. As a first approximation, although there are indications that environmental sinks may saturate [7], we assume a fixed ratio of natural CO_2 sinks (plants, land, ocean) to emissions. This assumption is reasonable if abatement measures are effective (changes in the ocean in particular can be rapid [8]), i.e., this is a conservative assumption for the benefits of early abatement.

Our climate model assumes the following parameters:

- We assume all variation in greenhouse cases, at a first approximation, is in CO_2 (reasonable since methane outputs have stabilized since 1990, and CO_2 output is the largest single anthropogenic contributor to greenhouse gases [10]) and therefore work with gigatonnes CO_2-equivalent ($GtCO_2$-eq)
- We base our scenarios on the IPCC's, which vary total emissions increases from 2000 to 2030 from 9.7 $GtCO_2$-eq to 36.7 $GtCO_2$-eq off a baseline of 39.8 GtCO2-eq, prior to mitigation [11]
- Total sinks including oceans and land-based consumers of CO_2: 50% of anthropogenic CO_2 production (30% oceans, 20% land) [9]

Our starting point is the scenarios defined by the Intergovernmental Panel on Climate Change (IPCC) [12]. These scenarios are intended to illustrate a range of possibilities, without attempting to predict the likelihood of any one outcome [13]. Any of these scenarios could equally well be modeled and for completeness all should be modeled. However, for purposes of illustrating the use of Modelica, we focus here on using only one base scenario, and vary mitigation strategy assuming a given trend in energy demand. Specifically, we choose the A1C scenario, because that represents high growth with maximal convergence of developing economies with developed economies. This scenario combination is relevant because of the debate as to whether mitigation implies forcing unremitting poverty on developing countries [14,15].

1.3 Structure of Paper

The remainder of this paper is structured as follows. In Section 2, we develop a model, based on plausible parameters, In Section 3, we present examine outputs of the model, and discuss future applications. Finally, Section 4 concludes with an overall discussion of findings and proposals for future work.

2 The Model

2.1 Methodology and assumptions

- We assume that the system is continuous since all physical process are continuous and the abatement and economic changes happen slowly
- Assume that the influence of abatement paths impacts only the cost of abatement represented by the carbon price. We don't model the feedback in the other direction
- Assume that 50% of emissions are absorbed environmentally

2.2 The economics of abatement

We develop a simple model based on the technology assessments of McKinsey and Co.'s climate change mitigation team in Sweden [19,20]. This model includes a cost curve for marginal abatement integrated with a mean reverting model for global energy prices.

- Assume that costs reduce over time as learning occurs
  - o constant learning rates for efficiency of energy production and use
- There are two ways to reduce emissions:
  - o efficiency-based which reduces total energy produce to meet same "virtual demand"
  - o increase proportion of zero-CO_2 energy
- Underlying energy price remains constant and is increased only through carbon pricing (likely to be incorrect as supply fails to keep up with demand, e.g., as appears to be happening at time of writing with oil).

2.3 Model design

The continuous assumption allows use to use ordinary coupled differential equations (ODEs).

Data from IPCC converted to rates of emission change and energy production/efficiency change and are incorporated as growth parameters in ODEs.

The most significant equations are:

1) $U'(t) = E(t) \times U(t) + L$
2) $P_E'(t) = P_{E\text{-}MRR} \times (P_{LT} + P_C \times C_{BI} - P_E)$
3) $P_C'(t) = P_{C\text{-}MRR} \times (P_A - P_C)$

Equation (1) allows us to express energy use *U* as an exponential component *E* and a linear component *L*. *U* represents virtual energy as explained above: it is the trend in energy demand, not taking into account that actual energy use may be less owing to efficiency gains. In our examples in this paper, we hold *E* to zero.

Equation (2) captures the variation in energy price (P_E) in terms of the energy price mean reversion rate ($P_{E\text{-}MRR}$) which captures the tendency for price spikes to smooth out, long term energy price (P_{LT}), the modeled carbon price (P_C), the carbon intensity at the start of the modeled time (C_B).

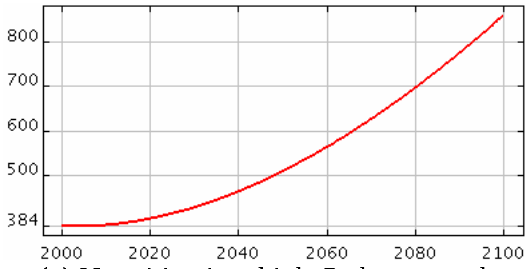
Equation (3) models the trend in carbon price in terms of the carbon price mean reversion rate ($P_{C\text{-}MRR}$) and abatement cost (P_A).

This is a closed form model for the interaction between energy costs under a carbon pricing regime and the concentration of carbon dioxide in the atmosphere.

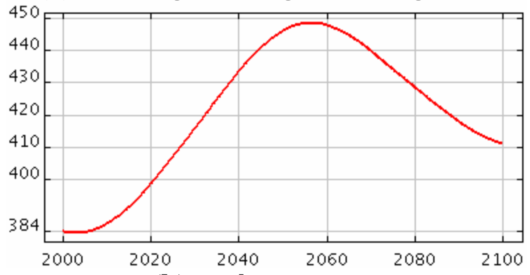
These equations can be expressed in Modelica as follows:

```
der(energyUse) = // (1)
       energyGrowthExp * energyUse +
       energyGrowthLinear;
der(energyPrice) =  // (2)
       energyPriceMRR *
       (longTermEnergyPrice +
       carbonPrice * baseCarbonIntensity -
       energyPrice);
der(carbonPrice) =  // (3)
       carbonPriceMRR * (abatementCost -
       carbonPrice);
```

This model is provided as a starting point, so the parameters should be taken as examples. Given that the IPCC has deliberately not provided probabilities for their scenarios [12], in the same spirit we do not claim that our specific examples are predictions, but rather case studies on which predictions can be built, once it becomes clearer which scenarios are most likely.



(a) No mitigation, high Carbon growth



(b) With mitigation

Figure 1. *CO_2 concentration*

3 Results

We have run some variations on parameters through the model, to illustrate how scenarios can be explored.

The A1C scenario explored here in its worst case with no mitigation results in rapid growth in carbon emissions, resulting in atmospheric CO_2 of the order of 800 parts per million (ppm), as illustrated in Figure 1(a). In this scenario, most energy by 2100 is carbon-based, as we have assumed zero mitigation: no increase in efficiency, no increase in non-emitting energy sources. With mitigation CO_2, peaks at around 450ppm (Figure 1(b) illustrates the early mitigation strategy; the late mitigation strategy is similar with a slightly higher, later peak).

Our mitigation strategy is based on reducing emissions to those of the B1T IPCC scenario. The early mitigation and late mitigation strategies are based on assuming the same cumulative reduction in emissions, but reversing the order, with faster change earlier in the more aggressive scenario.

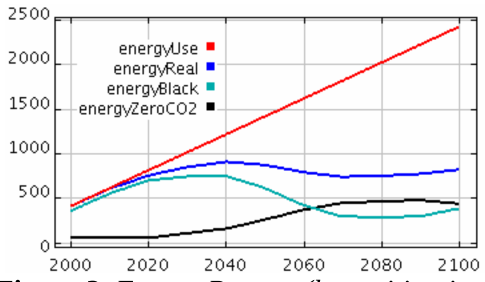
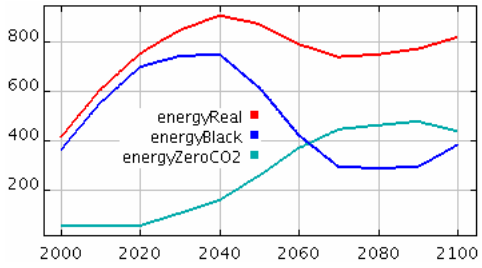
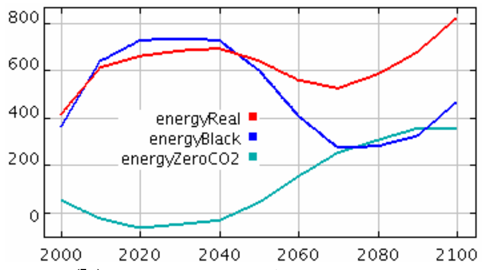


Figure 2. *Energy Pattern (late mitigation)*



(a) Less aggressive strategy



(b) More aggressive strategy.

Figure 3. *"Real" Energy Pattern*

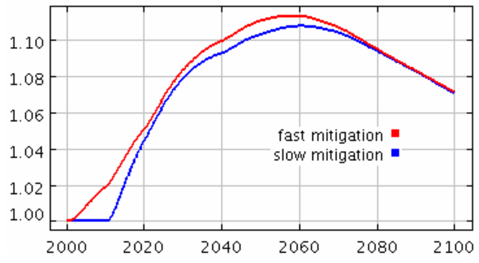


Figure 4. *Energy cost relative to no mitigation*

Figure 2 illustrates the change in energy pattern with our late mitigation (less aggressive) strategy. In this scenario, an abatement strategy has already started in 2000, and increases up to 2060, when new measures start to ease off. In the meantime efficiency measures increase up to 2050. In graphs, *energyUse* means "virtual" energy demand (energy demand not taking into account reductions caused by efficiency), *energyReal* is actual energy demand, allowing for efficiency measures, *energyBlack* is energy resulting in carbon emissions, and *energyZeroCO2* is emission-free energy.

Figure 3 contrasts the less aggressive (a) and more aggressive (b) strategies, this time leaving out the "virtual" energy line, since it is the same in all cases. Required non-emitting energy goes below zero in (b) because we are more than meeting the emission target in early years without adding more zero-emission energy, by aggressive efficiency measures. This is a flaw in the model, since we should not force abatement costs to be higher for more mitigation than is actually needed.

When we compare costs, the two mitigation strategies come out approximately equal – in the end. As illustrated in Figure 4 (cost scaled to no mitigation = 1), the fast mitigation strategy results in higher energy costs in the interim. However, the following limitations in the model favour the late mitigation strategy and therefore make it appear the better strategy in terms of cost:

- The constant learning rate assumption biases the simulation towards lower costs for late mitigation, as new technologies are more efficient, later
    - in practice, an aggressive mitigation strategy is likely to increase the learning rate e.g. if carbon taxes are passed through to low emission R&D
- Extra costs of late mitigation to the environment are not factored in, especially if environmental sequestration becomes less efficient as CO_2 levels rise
- Extra costs of early decommissioning of polluting plant would be higher in a late mitigation strategy, as a higher fraction of such plant would be built later in the strategy

We should however note that even where the faster mitigation strategy is more expensive, the gap is not large (at most 2%), owing to the fact that efficiency strategies are included in the mix.

4 Conclusions

This model provides a starting point for evaluating abatement paths for bringing CO_2 levels into line with requirements for stabilizing climate change.

We have modeled a limited range of scenarios to illustrate the techniques. Once it becomes clearer which scenarios are more probable, it will be a simple matter to rerun the model with different parameters.

In our future work we will investigate a wider range of scenarios, and fine-tune the model for a better fit to the real world, for example, changes in environmental sequestration as CO_2 levels rise. We will also fine-tune economic assumptions, to allow for a range of policy options such as more aggressive support for R&D for low-emissions technologies, and carbon taxes.

5 Acknowledgements

This work was supported by ARC Centre for Complex Systems Univ. Queensland, by Vinnova in the Safe and Secure Modeling and Simulation project, and by the Swedish Science Council (VR). Philip Machanick has been hosted by the School of ITEE, University of Queensland as a Visiting Fellow.

References

[1] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, David Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society* (SIMS2005), Trondheim, Norway, October 13-14, 2005. http://www.ida.liu.se/projects/OpenModelica

[2] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1,* 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.

[3] The Modelica Association. The Modelica Language Specification Version 3.0, Sept 2007. http://www.modelica.org.

[4] Peter Fritzson et al. The OpenModelica Users Guide, July 2007. http://www.ida.liu.se/projects/OpenModelica.

[5] JE Hansen. Scientific reticence and sea level rise, *Environ. Res. Lett.*, vol. 2 no. 2 April-June 2007

[6] GC Hegerl and FW Zwiers and P Braconnot and NP Gillett and Y Luo and JA Marengo Orsini and N Nicholls and JE Penner and PA Stott, Chapter 9: Understanding and Attributing Climate Change. In *Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change* (ed. S Solomon and D Qin and M Manning and Z Chen and M Marquis and KB Avery and M Tignor and HL Miller, pages 663-745, Cambridge University Press 2007

[7] Peter M. Cox, Richard A. Betts, Chris D. Jones, Steven A. Spall and Ian J. Totterdel. Acceleration of global warming due to carbon-cycle feedbacks in a coupled climate model, *Nature* vol. 408, 9 November 2000, pp 184-187

[8] John E. Dore, Roger Lukas, Daniel W. Sadler and David M. Karl. Climate-driven changes to the atmospheric CO_2 sink in the subtropical North Pacific Ocean, *Nature* vol. 424, 14 August 2003 pp 754-757

[9] Richard A. Feely, Christopher L. Sabine, Kitack Lee, Will Berelson, Joanie Kleypas, Victoria J. Fabry and Frank J. Millero. Impact of Anthropogenic CO_2 on the $CaCO_3$ System in the Oceans, *Science* 16 July 2004: Vol. 305. no. 5682, pp. 362 – 366

[10] IPCC, 2007: Summary for Policymakers. In: *Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change* [Solomon, S., D. Qin, M. Manning, Z. Chen, M. Marquis, K.B. Averyt, M.Tignor and H.L. Miller (eds.)]. Cambridge University Press, Cambridge, United Kingdomand New York, NY, USA.

[11] IPCC, 2007: Summary for Policymakers. In: *Climate Change 2007: Mitigation. Contribution of Working Group III to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change* [B. Metz, O.R. Davidson, P.R. Bosch, R. Dave, L.A. Meyer (eds)], Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.

[12] Nebojsa Nakicenovic, Joseph Alcamo, Gerald Davis, Bert de Vries, Joergen Fenhann, Stuart Gaffin, Kenneth Gregory, Arnulf Grübler, Tae Yong Jung, Tom Kram, Emilio Lebre La Rovere, Laurie Michaelis, Shunsuke Mori, Tsuneyuki Morita, William Pepper, Hugh Pitcher, Lynn Price, Keywan Riahi, Alexander Roehrl, Hans-Holger Rogner, Alexei Sankovski, Michael Schlesinger, Priyadarshi Shukla, Steven Smith, Robert Swart, Sascha van Rooijen, Nadejda Victor, Zhou Dadi. *IPCC Special Report on Emissions Scenarios (SRES),: Special Report on Emissions Scenarios, Working Group III, Intergovernmental Panel on Climate Change (IPCC),* Cambridge University Press, Cambridge, 2000. http://www.grida.no/climate/ipcc/emission/index.htm

[13] Nebojsa Nakicenovic, Arnulf Grübler, Stuard Gaffin, Tae Tong Jung, Tom Kram, Tsuneyuki Morita, Hugh Pitcher, Keywan Riahi, Michael Schlesinger, P. R. Shukla, Detlef van Vuuren, Ged Davis, Laurie Michaelis, Rob Swart and Nadja Victor. IPCC SRES Revisited: A Response, *Energy & Environment*, vol. 14, nos. 2 & 3, 2003, pp 187-214.

[14] Olive Heffernan. A push for political will, *Nature Reports Climate Change*, vol. 6 Nov 2007 p 79. http://www.nature.com/climate/2007/0711/pdf/climate.2007.60.pdf

[15] *Up in smoke? Threats from, and responses to, the impact of global warming on human development,* New Economics Foundation, London, 2004, http://www.neweconomics.org/gen/uploads/igeebque0l3nvy455whn42vs19102004202736.pdf

[16] J. Hansen, Mki. Sato, P. Kharecha, G. Russell, D.W. Lea, and M. Siddall. Climate change and

trace gases. *Phil. Trans. Royal. Soc. A*, vol. 365, 2007, pp 1925-1954, doi:10.1098/rsta.2007.2052

[17] Clean Air Act, US Government, 1990

[18] W.D. Montgomery, Markets in Licenses and Efficient Pollution Control Programs, Journal of Economic Theory 5 (Dec 1972):395-418

[19] Per-Anders Enkvist, Tomas Nauclér and Jerker Rosander. A cost curve for greenhouse gas reduction, *The McKinsey Quarterly*, no. 1 2007, pp 35-45

[20] Diana Farrell, Scott S. Nyquist and Matthew C. Rogers. *Curbing the growth of global energy demand*, The McKinsey Quarterly Web exclusive, 12 pp, July 2007

Appendix – The Complete Model

```
class CarbonWorldXIIa

  parameter Integer scenario=1 "1 for faster
early abatement, to 2 for slow early
abatement or 3 for 0 abatement";

  parameter Real gamma = 0.006725 "Correction
factor which can be used to account for
concentration dependent sequestration such as
sea and bio-";

  parameter Real absorptionFactor = 0.5;

  Real emission(start = baseEmission);
  Real carbConc(start=384)
    "Carbon Concentration";
  Real abatementCO2(start =
startAbatementCO2);
  Real abatementEfficiencyCO2 (start=0);
  Real abatementCO2Imputed;
  Real energyZeroCO2 (start = 0 );
  Real energyEfficiency (start = 0 );
  Real energyReal (start = 0);
  Real abatementCost;
  Real energyPrice(start =
   longTermEnergyPrice);
  Real carbonPrice(start =
    longTermCarbonPrice);
  Real energyUse (start = baseEnergyUse);
  Real energyBlack;
  Real totalCarbonIntensity;
  Real totalCarbonIntensity100;
  Real efficiencyValue(start =
    startEfficiencyValue);
  parameter Integer abateCO2 = 1,
    efficiencyEnergy = 2, abateEffciency = 3;
  Real abatementStepsCO2(start =
    plans[1, scenario, abateCO2]);
  Real  efficiencyStepsEnergy (start =
    plans[1, scenario, efficiencyEnergy]),
  Real abatementStepsEffciency(start =
    plans[1, scenario, abateEffciency]);

    parameter Real plans [:,:,:] = {
    {{0.22, 0.69, 0.2},{0, 0, 0.76},{0, 0,
0}},
    {{0.22, 0.69, 0.2},{0, 0, 0.76},{0, 0,
0}},
    {{0.57, 5.33, 0.56},{0.72, 15.23, 1.11},
     {0, 0, 0}},
    {{1.07, 10.64, 0.58},{1.01, 17.76, 0.86},
     {0, 0, 0}},
    {{1.16, 14.02, 0.65},{0.99, 19.04, 0.83},
     {0, 0, 0}},
    {{1.69, 23.47, 0.72},{1.52, 25.11, 0.78},
     {0, 0, 0}},
    {{1.84, 28.18, 0.76},{1.84, 28.18, 0.76},
     {0, 0, 0}},
    {{1.52, 25.11, 0.78},{1.69, 23.47, 0.72},
     {0, 0, 0}},
    {{0.99, 19.04, 0.83},{1.16, 14.02,
0.65},{0, 0, 0}},
    {{1.01, 17.76, 0.86},{1.07, 10.64,
0.58},{0, 0, 0}},
    {{0.72, 15.23, 1.11},{0.57, 5.33,
0.56},{0, 0, 0}},
    {{0, 0, 0.76},{0.22, 0.69, 0.2},{0, 0,
0}}
    };

  parameter Real abatementCatchupRate=1 "From
final abatementPlan to end of sim";

parameter Real energyGrowthExp=0.0,
energyGrowthLinear=20.0/(GJ_MWh/energyConvFac
tor);
  parameter Real tonnesToPPM =0.127365 "from
H:-aliebman-My Research-Energy-Climate
Change-Emissions trading-AL - Carbon Trading
Research-Modelica Models-
CalibrationData.xls";
  parameter Real carbonToCO2 = 3.664
"Conversion between mass Carbon and Carbon
Dioxide";

  parameter Real startEfficiencyValue= 31.06
"150 $/tCO2e";
  parameter Real startAbatementCO2=5 "tCO2e";
  parameter Real learningRate=0.02;
  parameter Real GJ_MWh=3.6,
energyConvFactor=GJ_MWh "GJ_MWh or 1.0";
  parameter Real baseEmission=40 "40 GtCO2e
from energy sector - McKinsey", baseEnergyUse
= 411*energyConvFactor/GJ_MWh "IPCC Special
Report on Emission Scenario (SRES) 2000 -
linear fit and interpolation between 1990-
2050 ";
  parameter Real baseCarbonIntensity =
baseEmission /baseEnergyUse "0.7
/energyConvFactor - tonnes/MWh converted to
tonnes/GJ";
  parameter Real carbonPassThrough = 1;
  parameter Real longTermEnergyPrice = 80
/energyConvFactor; //"$100/MWh long term
energy price" // Will need to be a dynamic
quantity later
  parameter Real longTermCarbonPrice = 0.0;
// "$20/tCO2 long term abatement /carbon
cost" // Need to check this actually makes
sense!
  parameter Real energyPriceMRR = 1.0 "Energy
price mean reversion rate";
  parameter Real carbonPriceMRR = 1.0 "Carbon
Price mean reversion rate" ;
  Real relEnergyPrice (start = 1);
  Real energyCostTrend (start = 1);
  Real scaledEnergyPrice (start=0);
  Integer which (start = 2); // used which =
1 to initialize abatements

  function nextStep
    input Real data[:,:,:];
    input Integer i,j,k;
    output Real step;
  algorithm
    step := data[i,j,k];
  end nextStep;

equation
  energyCostTrend = relEnergyPrice *
energyUse / baseEnergyUse;
  // useful to compare strategies on cost
  relEnergyPrice = energyPrice /
longTermEnergyPrice;
  // useful to compare energy cost across
strategies that vary total use
  scaledEnergyPrice = relEnergyPrice *
energyReal / energyUse;
  abatementCost =
efficiencyValue*(sqrt(abatementCO2/startAbate
mentCO2) - 1);
  der(efficiencyValue) = -
learningRate*efficiencyValue " -
longTernmEnergyPrice *
someKindOfCarbonIntensity)";

  when sample(0, 10) then //StartTime
    which = if pre(which) < size(plans,1)
then
      pre(which) + 1 else pre(which);
  end when;

  abatementStepsCO2 = nextStep (plans, which,
scenario, abateCO2);
  efficiencyStepsEnergy = nextStep(plans,
which, scenario, efficiencyEnergy);
  abatementStepsEffciency = nextStep(plans,
which, scenario, abateEffciency);

  der(abatementCO2) =  abatementStepsCO2; //
This is a carbon dioxide quantity

  der(energyEfficiency) =
efficiencyStepsEnergy*energyConvFactor/GJ_MWh
; // This is an energy quantity

  der(abatementEfficiencyCO2) =
    abatementStepsEffciency; // This is a
carbon dioxide quantity

  energyZeroCO2=(abatementCO2-
abatementEfficiencyCO2)/baseCarbonIntensity;
  energyBlack = energyUse - energyEfficiency-
energyZeroCO2;
  emission = energyBlack*baseCarbonIntensity;

abatementCO2Imputed=energyZeroCO2*baseCarbonI
ntensity;

  totalCarbonIntensity = emission/energyUse;
  der(carbConc) =
tonnesToPPM*(emission*absorptionFactor)-
gamma*carbConc;

  der(energyUse) =
energyGrowthExp*energyUse+energyGrowthLinear;
  der(energyPrice) = energyPriceMRR*(
longTermEnergyPrice +
carbonPrice*carbonPassThrough*
baseCarbonIntensity - energyPrice);
  der(carbonPrice) =
  carbonPriceMRR*(abatementCost -
carbonPrice);
  totalCarbonIntensity100=
    100*totalCarbonIntensity;
  energyReal = energyBlack + energyZeroCO2;
end CarbonWorldXIIa;
```

Modelling of an adsorption chiller with Modelica

Matthias Schicktanz
Fraunhofer Institute Solar Energy Systems
Heidenhofstr. 2, 79110 Freiburg
matthias.schicktanz@ise.fraunhofer.de

Abstract

This paper describes the model of an adsorption chiller. The model follows a component modeling approach based on the Modelica Media and Modelica Fluid Library. New models describe the phenomenon of condensing, evaporation and adsorption. A new library has been created to describe the physical properties of adsorption materials. First simulations were performed and are compared to measured data of an existing machine. The simulated curves show good accordance to measured data.

Keywords: thermally driven chiller, adsorption chiller

1 Introduction

Facing a globally increasing cold demand to cover the need of comfort in hot areas and at the same time facing the problem of global warming, the market for thermally driven chillers is increasing. Thermally driven chillers produce cold but are powered by heat instead of mechanical work (electricity). Depending on the application, heat sources with temperatures above 70°C such as solar heat, waste heat or heat of a cogeneration unit can be used.

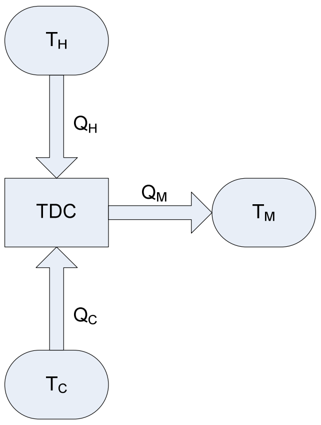


Fig. 1: Thermally driven chiller (TDCs) pump heat from a low temperature heat source at T_C to a middle temperature heat source at T_M and are powered by heat at a temperature level T_H ($T_C<T_M<T_H$).

Fig. 1 shows the working principle of a thermally driven chiller. It pumps heat from a low temperature heat source at T_C to a middle temperature heat source at T_M powered by heat at a temperature level T_H ($T_C<T_M<T_H$).

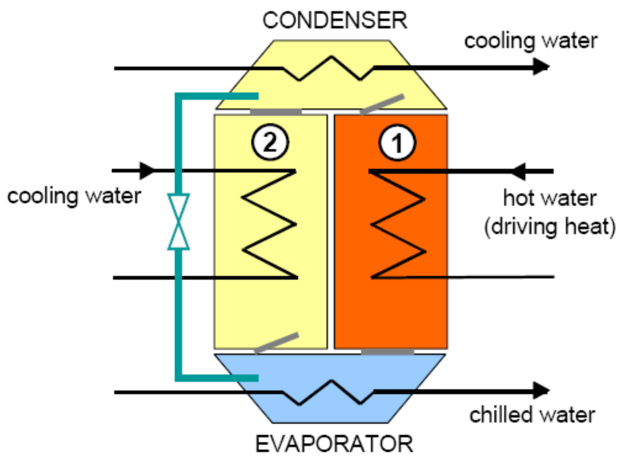


Fig. 2: Scheme of an adsorption chiller with two adsorbers. Cited from [3].

Fig 2 shows the technical implementation of an adsorption chiller. It shows four vacuum vessels for the four main components evaporator, condenser and two adsorbers. Each component contains a heat exchanger that is connected to one of the three heat reservoirs T_C, T_M and T_H as mentioned in Fig 1. The water in the loops is called chilled water, cooling water and hot water, respectively. The components are separated by four flaps that control the vapor flow in the machine. An expansion valve connects the condenser to the evaporator.

At a low pressure and low temperature level refrigerant (here: water) evaporates in the evaporator and passes the flap to the left adsorber (2). Thereby it takes up heat from the chilled water. The left adsorber adsorbs the water vapor at the surface of the adsorbent coating (here: silica gel). The energy released during this exothermal process is passed to the cooling water.

In the meantime the second adsorber (1) at the right side gets desorbed powered by energy of the hot water loop. This occurs at a higher pressure level by heating the adsorbent. The released water vapor passes the flap to the condenser where it condenses and releases energy to the cooling loop. The condensate afterwards passes an expansion valve before reaching the evaporator.

2 Structure of the adsorption chiller model

Fig. 3 shows the Modelica representation of the process described above. The four main components condenser, evaporator and two adsorbers are separated by four flaps. All models are connected via the fluid port of the Modelica Fluid Library [1]. A causal connector represents the expansion valve between condenser and evaporator.

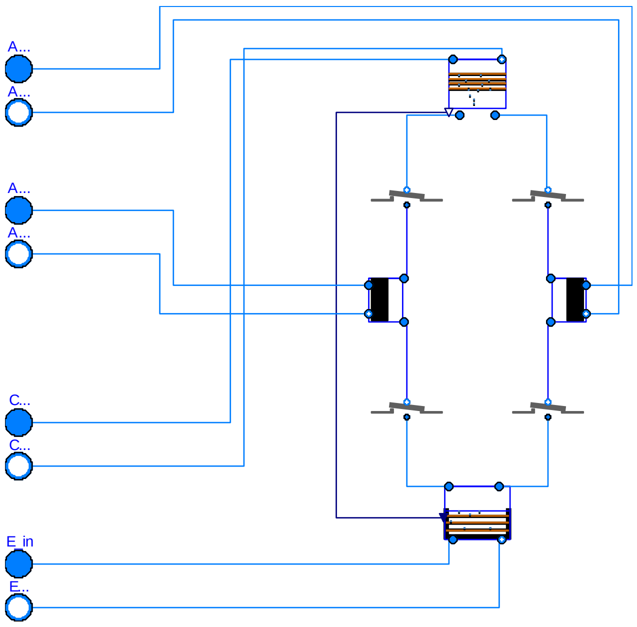


Fig. 3: Modelica representation of the adsorption chiller main components.

2.1 Functional Component Models

All main components used in the model have a similar design. Fig. 4 shows the graphical representation of the condenser.

The model mainly uses components from the Modelica Fluid Library. The ports at the top lead to the hydraulic connections (here cold water). The golden box in the middle represents a finned heat exchanger which is described below. As a first approximation, the heat transfer coefficient describing the condensation of water vapor at the heat exchangers surface is assumed to be constant. Therefore, a constant thermal conductor taken from the Modelica Standard Library connects the heat exchanger to a condensing model. In the condensing model simple heat and mass conservation equations are taken into account. Within this model, no mass and energy storage takes place. All condensate is released to the water outlet connector which is a causal output connector and was especially designed for this purpose. Opposed to the standard Modelica Fluid connector it only transmits flow variables (m_flow, H_flow) but no state variables (p, h), since the later change during the expansion process in the expansion valve.

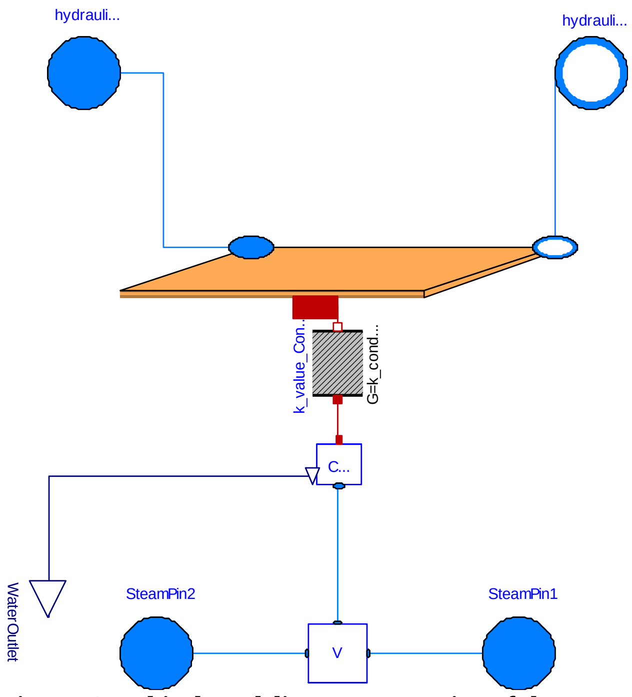


Fig. 4: Graphical Modelica representation of the condenser.

The models for the evaporator and the adsorber have the same structure as the condenser but the condensing model is replaced by an evaporation model and an adsorption model, respectively.

The evaporation model in the evaporator contains basically the same heat and mass conservation equations as the condensing model, but additionally heat and mass is stored to describe the refrigerant pool that covers the heat exchanger. Moreover, the connector for the condensate is defined as input as a counterpart to the condenser.

The adsorption model contains fundamental heat and mass conservation equations with internal storage to describe the adsorption process. The load $x = f(p,T)$ describes the amount of refrigerant that is adsorbed by the adsorbent

$$x = \frac{m_{refrigerant}}{m_{adsorbent}}$$

In equilibrium the load only depends on temperature and pressure at the adsorbent surface. The specific adsorption enthalpy h_{ad} as well as the equilibrium relation $x = f(p,T)$ are defined in the adsorption material package described below. However, the speed of adsorption is described as a simple linear relation between driving pressure and mass flow

$$\dot{m} = \beta(p_{sat} - p)$$

where p_{sat} is the saturation pressure for the refrigerant in the adsorbent, p is the vapor pressure in the vessel and $\dot{m}$ is the mass flow of refrigerant into or out of the adsorbent. β is an effective diffusion coefficient that describes the kinetics and so for is a fit parameter [2].

2.2 Finned Heat Exchanger Model

Fig. 5 shows a graphical representation of a finned heat exchanger model. It is a simple model consisting of different heat capacities for fins and tubes and a constant heat transfer coefficient model that represents the heat transfer from the hydraulic medium in the pipe to the pipe's wall. The pipe model from the Modelica Fluid Library is applied.

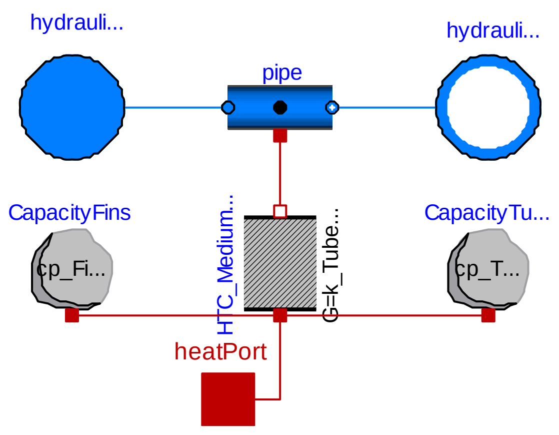


Fig. 5: Modelica representation of a simple finned heat exchanger.

2.3 Adsorption chiller piping model

Fig. 6 shows a graphical representation of the adsorption chiller piping. The purpose of the piping is to distribute the flow of the three loops for hot water, cooling water and chilled water to the four main components.

The connections to the hot water, cooling water and chilled water loops are shown on the left side. The connections to the four vessels from Fig. 3 are on the right side. The single valve in the lower right controls the distribution of the cooling water between condenser and cooled adsorber. From a hydraulic point of view both vessels are arranged in parallel.

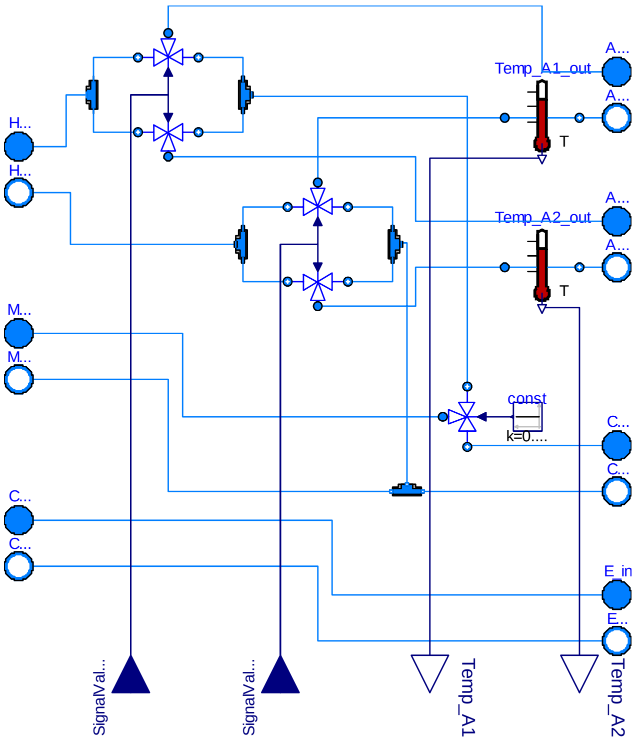


Fig. 6: Modelica representation of the piping of the adsorption chiller.

The two three-way-valves in the upper left of Fig. 6 control the forward flow of hot water and cooling water. Either the one or the other adsorber is connected to the hot water loop and cold water loop, respectively. Similarly, the two three-way-valves in the middle control the reverse flow. An external controller connected via the control connector at the bottom controls the valves. In order to improve the efficiency of the chiller the reverse flow valves are switched according to the temperatures in the reverse flow of the adsorbers. The warmer outlet flow is connected to the hot water and the colder outlet flow to the cooling water. Therefore, temperatures at the adsorber outlets are delivered to the controller. Switching of the valves in reverse flow occurs at a later time than switching of the valves in forward flow.

3 Adsorption Material Properties

At Fraunhofer ISE different adsorption materials are measure and characterized. The material package in the adsorber model is therefore defined as replaceable and simulation can be performed with different materials.

3.1 Description of the adsorption physics

According to Dubinin's theory [4] the physical equilibrium between temperature, pressure and load in the adsorber can be described by knowing only one function

$$W = f(A)$$

in which $[W] = {}^{m^3}\!/\!_{kg}$ is the adsorption volume onto the adsorbent surface that describes how much vapor can be adsorbed. Therefore, it is proportional to the load

$$x = \rho \cdot W$$

where ρ is the density of the refrigerant in the liquid adsorbed state. The adsorption potential $[A] = {}^{J}\!/\!_{kg}$ describes the conditions of pressure and temperature charactering adsorption process and is defined as

$$A = R \cdot T \cdot \ln \frac{p_{refrigerant}}{p}$$

with specific gas constant R, saturation pressure of the pure refrigerant $p_{refrigerant}$ and saturation pressure p of the refrigerant in the adsorbed state. Also the adsorption enthalpy h_{ad} is derived from the characteristic material equation:

$$h_{ad} = h_v + A - T \cdot \alpha \left(\frac{\partial A}{\partial \ln W} \right)_T$$

Here h_v is the specific evaporation enthalpy of the refrigerant and α is the linear thermal expansion coefficient of refrigerant in the adsorbed state.

3.2 Implementation of adsorption data in Modelica

The implementation of the adsorption material properties in Modelica follows Dubinin's theory to describe all parameters with the $W = f(A)$ relation. In practice, for the specific adsorption enthalpy also the derivative ${}^{\partial W}\!/\!_{\partial A}$ is needed since Modelica cannot perform this transformation. Therefore, three functions are needed to describe the properties of a material:

The first function describes the relation between W and A (which may contain piecewise-defined functions), the second function gives the according derivative and the third function contains the needed coefficients. The first two functions therefore are extended by the coefficient function.

Each material package is then extended by a partial base class package. In this partial package all physical adsorption properties as described above are calculated.

So far, all implemented adsorption materials work with water as refrigerant but in principle it is possible to extend the package to the physical properties of other adsorption pairs like methanol/activated carbon.

4 Preliminary results and discussion

Simulations with real measured data as input have been performed in order to compare the model with a real machine. As working water according to the IAPWS-formulation from the Modelica Media Library and pair silica gel from the adsorption materials package are used. Measurement data come from the SorTech SKA PT 402 chiller. Data for temperature and mass flow at the inlets of the adsorption machine are given as inputs for the simulation from the measurement. Weight of adsorbent and heat capacities are given as parameters. Moreover, switching times for forward valves in the adsorption chiller piping are set manually, whereas reverse flow valves are switched by the controller as described above.

Fig. 7 compares measured data with the simulated results. It shows the measured temperatures at the inlet and outlet of the hot water, cooling water and chilled water loops versus time. Moreover, simulated results at the outlets are shown.

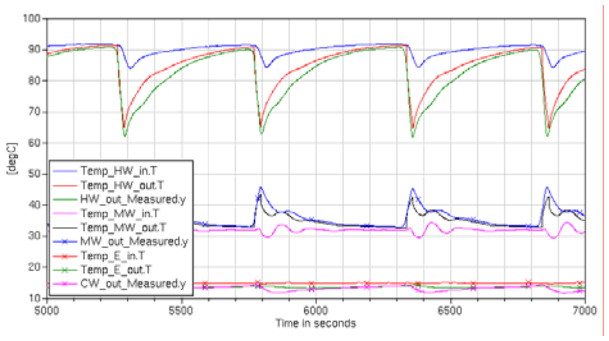


Fig. 7: Comparison between measured and simulated data. Temp_HW_in.T, Temp_MW_in.T and Temp_E_in.T are input data for the inlets of the adsorption chiller. Temp_HW_out.T, Temp_MW_out.T and Temp_E_out.T are simulated temperature values at the chiller's outlets. HW_out_Measured.y, MW_out_Measured.y and CW_out_Measured.y are measured outlet temperatures.

A half cycle needs approximately about 500s. After this time adsorption or desorption, respectively, stops and the valves in the piping model are switched to change operation mode. Therefore, a complete adsorption/desorption cycle needs about 1000s. The peaks especially in the hot and cooling water loop are caused by this switching process.

The simulated curves show good accordance to the measured data. After the switching process the simulated output temperature from the hot water loop (Temp_HW_out.T) and measured data (HW_out_measured.y) start from a similar temperature and converge against the same final desorption temperature.

The same is true for the simulated and measured values (Temp_MW_out.T) and (MW_out_measured.y), respectively, in the cooling water loop.

In both loops the simulated temperature differences after the switching process are smaller than the measured values. This might indicate that the switching in the reverse flow valves of the SKA PT 402 happens to soon.

Inlet temperatures actually were supposed to be constant temperatures but the test bench for the chiller was not able to handle the high power requirement which resulted in oscillating inlet temperatures. For example the middle temperature level Temp_MW_in.T shows a double overshoot in the time interval 5800s-5900s. The model can handle this fluctuation. At the outlet Temp_MW_out.T and MW_out_measured.y both show a reaction to the fluctuation. But since the model does not include the length of the pipes between adsorber and thermometer the simulated reaction happens before the real measured events.

5 Conclusion

A simulation model for an adsorption chiller on a component approach has been implemented in Modelica. Even though it consists of simple equations for heat and mass transfer it already shows good accordance to measured data. It demonstrates the principle functions of the adsorption chiller and shows reaction to dynamic changes.

Moreover a package for different adsorption materials has been designed according to Dubinin's Theory.

New measurements will also contain pressure data in the adsorption machine, with this data it will be possible to calibrate the free parameters in the model which are currently only first approximations.

Acknowledgment

I would like to thank SorTech AG for appropriating measured data.

References

[1] Casella F., Otter M., Proelss K., Richter C., Tummescheit H., The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. Modelica Conference 2006

[2] Núñez, T. Charakterisiserung und Bewertung von Adsorbentien für Wärmetransformationsanwendungen. Freiburg im Breisgau, Germany: PhD thesis, Department for physics, Albert-Ludwigs-Universität, 2001.

[3] Henning H-M, Solar cooling. Presentation ISES Beijing, China 2007.

[4] Dubinin M. Physical Adsorption of Gases and Vapors in Micropores, volume 9, pages 1-70. Academic Press, 1975.

An External Model Interface for Modelica

Torsten Blochwitz Gerd Kurzbach Thomas Neidhold
ITI GmbH, Webergasse 1, 01067 Dresden, Germany

Abstract

The paper describes the integration of non-Modelica submodels to a complete Modelica model. We show, that the Modelica standard interfaces to external code (external function and external object) are not suited to integrate the behavior of non-trivial models. The necessary enhancements of the external object interface are worked out and the usage is demonstrated.

Keywords: External Function, External Object, C-Interface

1 Introduction

With ITI-SIM and SimulationX [1] the company ITI develops and distributes software for system simulation since 1991. SimulationX provides full support for Modelica since release 3.0. The steadily growing acceptance of these programs is based on a modern user interface, which enables engineers an easy access to modeling, simulation and optimization techniques by using efficient calculation methods associated with a wide range of libraries and tools. A large contribution to this success is the availability of interfaces to other CAE tools like MATLAB/Simulink, MSC.ADAMS or SIMPACK. In addition to various forms of co-simulation the C code based exchange of models between different tools is also supported. This enables the user to cooperate across team boundaries independent of the finally used simulation tools. The encapsulation of the model functionality, which will be achieved by the compilation of the code generated from the original model, also allows an effective protection against unwanted insight into the parameters and behavior. With the description of this interface, as well as our proposal for its integration into the Modelica language we want to make available the described advantages to the whole Modelica community.

2 Motivation

There are different motivations to integrate non-Modelica submodels into Modelica models:

1. Sometimes a component is modeled using a specialized simulator for a specific physical domain (e.g., SIMPACK for complex multi body systems or GT-POWER [2] for combustion engines). For system simulation within a Modelica simulator the component should be integrated into a Modelica model. Often the model functionality of the special simulator can be exported as C-code.
2. A supplier has developed a model of a component in Modelica. He wants to supply this model to the OEM but wants to protect his know how, contained in the physical model. The safest way to do that is to provide the model in binary form as a compiled library with a well defined interface.

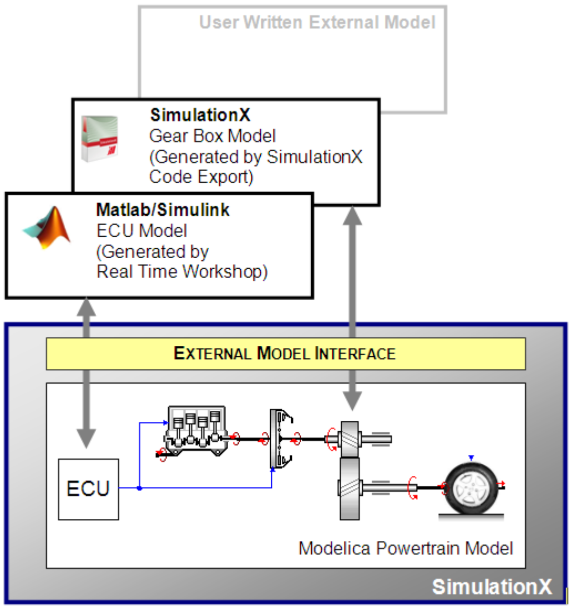

User Written External Model
SimulationX
Gear Box Model
(Generated by SimulationX Code Export)
Matlab/Simulink
ECU Model
(Generated by Real Time Workshop)
EXTERNAL MODEL INTERFACE
ECU
Modelica Powertrain Model
SimulationX

Figure 1: Modelica model with embedded external components

Modelica currently supports the following interfaces to external functionality [3]:

- external function interface
- external object interface

According to the Modelica Language Specification results of external functions may only depend from their arguments, i.e., the functions have no internal memory. Complex models do have a memory.

External objects as an improvement of external functions provide a memory context which is reported between the function calls.

Beside the more or less complex function for the right hand side of an ODE or DAE, external models may contain discrete states, state- or time-dependent events, or delay buffers. To integrate those into the simulation, information about the objects have to be exchanged between the external model and the simulation environment. The Modelica external object interface does not provide the functionality to exchange this information. It must be extended to an "External Model Interface." The following chapter describes the requirements to the external model interface resulting from the features of complex models.

The inclusion of controller code (e.g. ECU code generated by the Real Time Workshop from The MathWorks) is not subject of this article. Such components must be called with a constant sample rate during the simulation. This can be done utilizing the existing Modelica interfaces (external function or external object). No extensions are necessary.

3 Requirements for External Model Interface

3.1 Requirements Resulting from Model Features

At first we consider external models, which are represented by ordinary differential equations (ODE). The equations may contain discontinuities.

Such systems are represented by following equations:

$$\dot{x} = f(x,u,p,z,r,t) \qquad (1)$$

$$y = g(x,u,p,z,r,t) \qquad (2)$$

$$z = h_1(x,u,p,r,s,t) \qquad (3)$$

$$r = h_2(x,u,p,z,t) \qquad (4)$$

with:

x	Continuous states
u	Inputs
y	Outputs
p	Parameters
z	Discrete states
r	Root functions
s	Sample variables
t	Time.

Equation (1) represents the right hand side (RHS) of the ODE. Equation (2) represents the calculation of the outputs. Both calculations should be separated in different functions to enable an optimum arrangement of the external model in the calculation sequence of the enclosing model.

The other equations deal with event handling and discontinuities.

Events:

Two kinds of events must be handled: time events and state-dependent events. Time events are produced by timers or the Modelica **`sample`** keyword. They are signaled from the solver to the model by setting corresponding sample variable s.

State events are signaled from the model to the solver by zero crossings of the root functions r. Discrete variables z can change its values only at events.

According to our experience a reliable event handling is crucial for a robust and fast calculation.

Reinitialization of States:

At event instants state values may be reinitialized by the external model. The solver should be informed about such an operation.

Additional Model Information:

External models of specific domains may provide further information which eases a robust and fast solution. Examples are minimum and maximum permissible values for states (e.g. absolute temperatures and pressures have to be positive).

Other models could provide the Jacobian matrix directly.

It depends on the simulator, if this data is used.

Special Features:

Some special features demand actions on valid model data. For example, the buffers of delay blocks must be updated with valid data once after a successful time step. For that reason, the external model must be called once after successful steps with valid data and must be informed about that.

Other model features may require the allocation and freeing of memory or data is to be read from files once. For that reason special functions must be called once at the beginning and the end of the simulation run.

3.2 Requirements Resulting from the Integration into the Enclosing Model

For integration of the external model into the enclosing Modelica model the external model calls must be correctly positioned in the calculation sequence.

If the outputs of the external model depend only from states, the arrangement is simple: the external model must be called before one of the outputs is needed.

If the external model has direct feed through (outputs depend directly from inputs) the situation is more complex. The external model must be called before the outputs are needed and after the inputs are calculated. If the enclosing model defines dependencies of the inputs from the outputs of the external model, we have algebraic loops. The simulator must treat them in an appropriate manner.

For this reason it is essential for the external model to provide the information, which output depends on which input(s). If the model creator is not able to offer this structural information, the worst case (each output depends from each input) has to be assumed.

3.3 Technical Requirements

The external model interface for Modelica should be similar to the Simulink S-function interface from The MathWorks [4]. This interface is quite well adopted and widely used.

The realization of the data transfer should be simulator-specific. The external model accesses the data via functions or macros. These functions or macros are provided by the target simulator.

The external model interface should be usable by non-Modelica simulators too. These simulators should be able to use and/or to create models using the interface.

We will assume that at least the interface part of external models is written in C. How the external model is linked to the simulator is tool specific and depends on the capabilities of the operating system.

4 The External Model Interface

The external model interface can be seen from the following three perspectives:

- Specification of the functions and data provided by the external model.
- Specification of the calling sequence by the solver.
- Specification of the interface to Modelica.

These three views to the external model interface are shown in Figure 2.

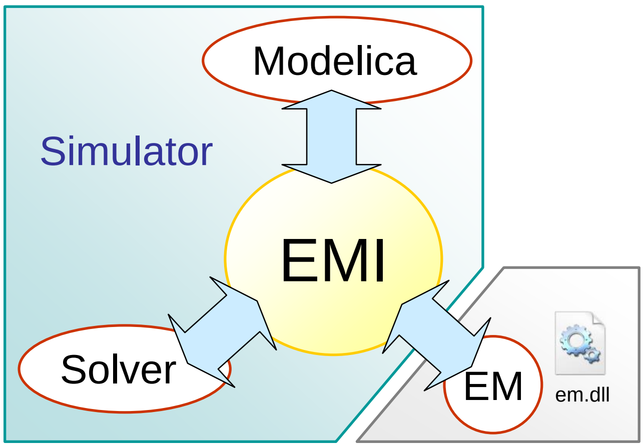


Figure 2: Three views to the external model interface

On the other hand the interface provides a set of utility functions which can be called from external model code.

According to the requirements we get the following data flow between the components (Figure 3).

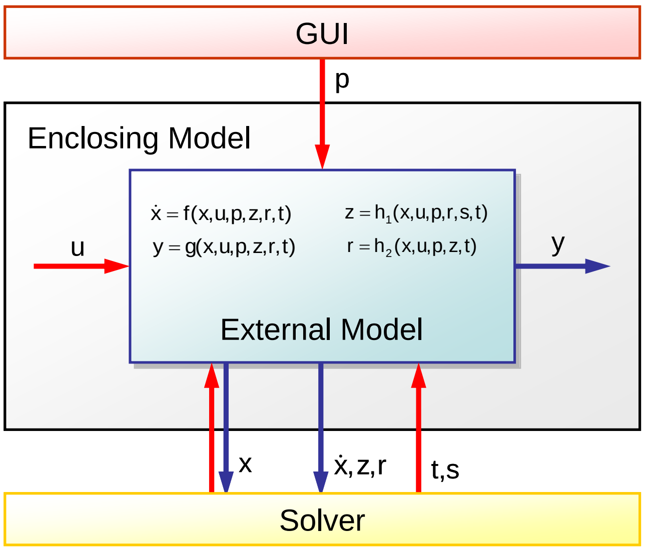


Figure 3: Data flow

The details, i.e. which data is to be provided by which function, are part of the complete specification, which will be published by the authors.

4.1 External Model View

The data transfer is realized via the external model context, the structure `emc`. The external model must implement the following functions:

`void emiInitializeSizes(emc *C)`

- Defines the dimensions of the model.
- Transfers additional information (input – output dependencies)
- Is called multiple times before the calculation.

`void emiStart(emc *C)`

- Is called once at the beginning of the simulation run.
- Can be used, e.g., to allocate memory.

`void emiInitializeSampleTimes(emc *C)`

- Transfers constant sample times.
- Is called once at the beginning of the simulation run.

`void emiInitializeConditions(emc *C)`

- Sets the initial conditions for continuous and discrete states.
- Is called once at the beginning of the simulation run.

`void emiTerminate(emc *C)`

- Is called once after the simulation run.
- Allocated memory can be freed here.

The following functions are called *multiple* times during one calculation step:

`void emiDerivatives(emc *C)`

- Computes the RHS of the ODE **(1),** and **(3)** during event iteration.

`void emiOutputs(emc *C)`

- Computes the outputs **(2)**.

`void emiZeroCrossings(emc *C)`

- Computes the root functions **(4)**.

The next function is called *once* after a successful calculation step:

`void emiUpdate(emc *C);`

- Called after a successful calculation step with valid data.

It is not allowed to access the data in the external model context `emc` directly. Instead, a set of function or macros is to be used, e.g.:

`emcSetNumContStates(emc *C, int_T n)`

- Sets the number of continous states.

`emcGetContStates(emc *C)`

- Returns a pointer to the state array.

`emcSetSolverNeedsReset(emc *C)`

- Informs the solver about a reinitialization of states.

The implementation of the `emc` and the access functions are target tool specific and must be provided by the simulator manufacturer.

4.2 Solver View

Figure 4 shows a simplified flow chart of the solution process for a Modelica model. It demonstrates which functions are called at each stage.

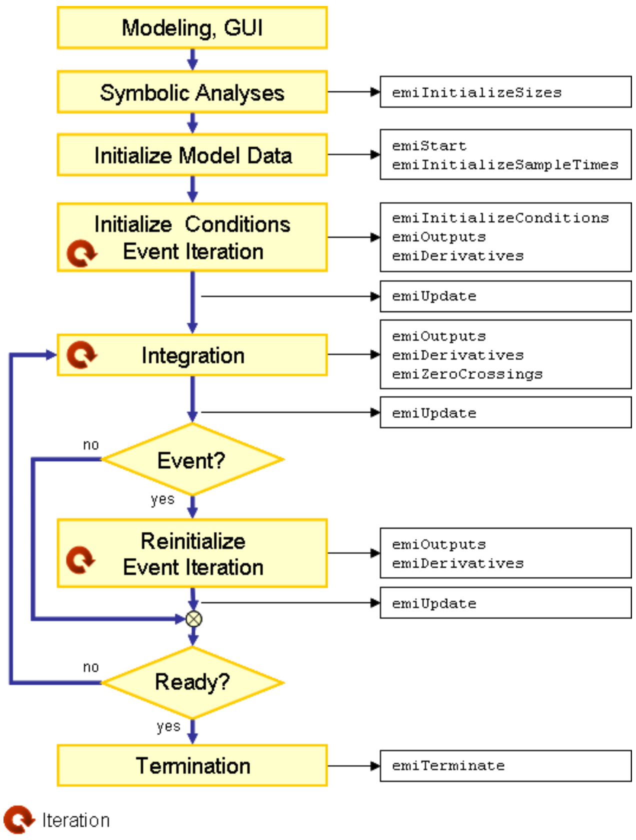


Figure 4: Solution process flow chart

If the integrator works iteratively, the functions `emiOutputs` and `emiDerivatives` may b called several times at the same time be instant with temporary data. These functions are to be implemented as reentrant and must not store any data.

For these purposes `emiUpdate` is called with valid data once after a successful time step.

The method for robust handling of discrete variables during event iteration is an open issue at the moment. There are several possibilities, which should be discussed with other simulator vendors.

4.3 Modelica View

This section describes the enhancements of the external object call interface to the external model interface. The information to be exchanged between the external model and the Modelica simulator are of two types:

- Data for the model (parameters, inputs, outputs). These are exchanged via usual function arguments and appear inside the Modelica model.
- Data for the solver (states, derivatives, residuals, discrete states, root functions…). These are handled implicitly by the simulator using the external model context.

We suggest the new Modelica built in type "external model" as an extension of the external object interface. The implicit declaration of the type could be:

```
class ExternalModelInterface
  extends ExternalObject;
    function constructor
      input String emName;
      output ExternalModelInterface emi;
      external "C" emi=initEM(emName);
    end constructor;
    function destructor
      input ExternalModelInterface emi;
    external "C" terminateEM(emi);
  end destructor;
end ExternalModelInterface;
```

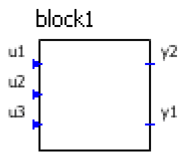
The calculation function is declared implicitly as follows:

```
function calcEM
  input ExternalModelInterface emi;
  input Real u[nu];           //inputs
  input Parameter Real p[np]; //parameters
  output Real y[ny];          //outputs
  external "C" y=calcEM(emi, u, p);
end calcEM;
```

Differing from the external object interface, the functions `initEM`, `terminateEM` and the calculation function `calcEM` do not correspond one to one to the functions of the external model. During the symbolic analyses of the model these functions have to be mapped to the appropriate function calls of the external model.

The dimensions (`nu`, `np`, `ny`) and the dependencies of the outputs from the inputs must be known during the symbolic analyses. This information should be provided by the external model. To get this information, the external model must be called already during the analyses. This is another difference to the external object interface.

The usage of the external model interface in a Modelica model is:



```
model Block "Block with External Model"
  input SignalBlocks.InputPin u1;
  input SignalBlocks.InputPin u2;
  input SignalBlocks.InputPin u3;
  output SignalBlocks.OutputPin y1;
  output SignalBlocks.OutputPin y2;
  ExternalModelInterface emi=
    ExternalModelInterface("c:\test.dll");
  equation
  {y1,y2}=calcEMI(emi,{u1,u2,u3},{1,2,3});
end Block;
```

As denoted before, the Modelica model handles only the inputs, outputs, and parameters of the external model. The other information is exchanged implicitly between the solver and the external model. If the user wants to access such internal data for debugging purposes, special functions could be provided. Access to the states could be given by:

```
function getEMStates
  input ExternalModelInterface emi;
  output Real x[nx];      //states
  external "C" x=getEMStates(emi);
end getEMStates;
```

5 Application Scenarios

5.1 Hand-Written External Models

External models can be developed by any programmer. The complete API with all necessary data structures and functions is described in a programmer's manual. Normally it should be the exception to implement an external model completely by hand. Instead, the adaption and integration of existing source code according to the external model interface requirements will be the typical task. This way is practicable for single solutions and non-commercial applications. The necessary work can be simplified by using precast templates.

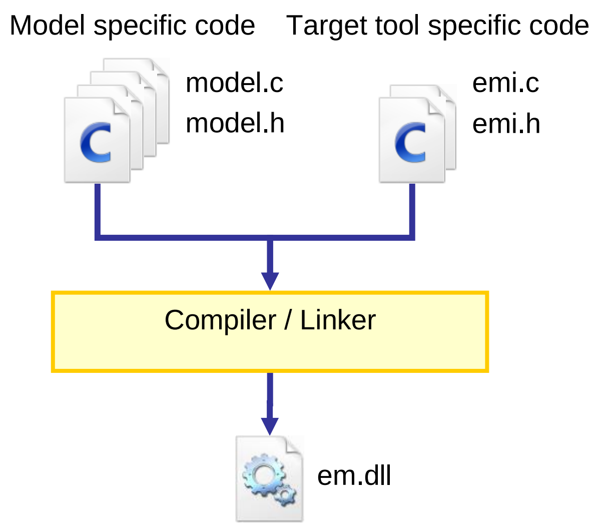


Figure 5: Work flow for hand-written external models

5.2 Tool-Generated External Models

For commercial CAE tools the automatic generation of external models is feasible. The Code Export Wizard integrated in SimulationX is already able to generate source code for various target platforms.

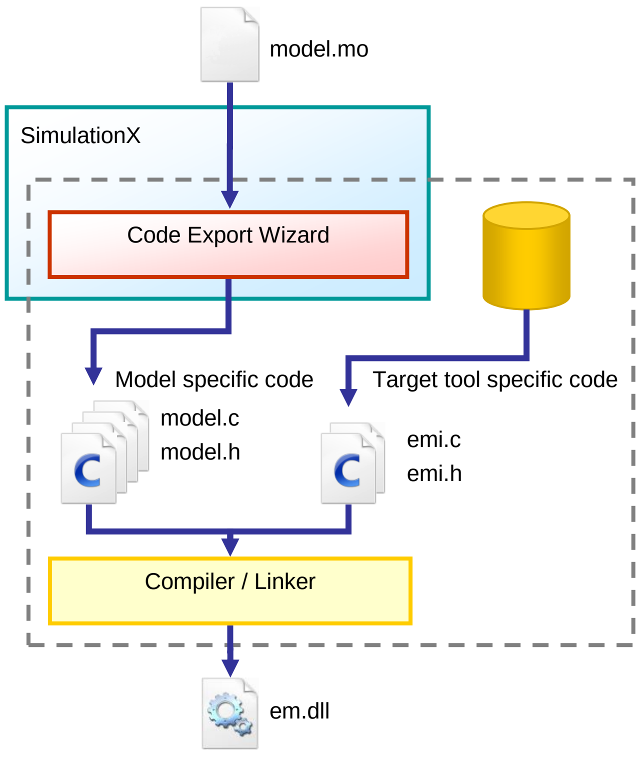


Figure 6: Work flow for tool-generated external models

Among S-functions for MATLAB/Simulink and UFORCE-routines for SIMPACK [5] also real time targets like ProSys-RT from Cosateq [6] are supported. For the automatic generation of EMI-conform model code a new target project type was added to the SimulationX Code Export Wizard. The wizard assists the user in the selection of inputs, outputs, and parameters. If a supported compiler is installed, SimulationX is able to build the External Model DLL immediately. The resulting model library does not need any additional runtime modules and can be distributed without limitations.

6 Conclusions and Outlook

We have shown how the interface to an external model in SimulationX is structured.

In one of the next Modelica Design meetings, we will make a proposal for the new predefined partial class `ExternalModel` which represents the model context inside the Modelica language.

The external model interface will be open for other software vendors. The interface itself does not contain Modelica specific parts. In this way external model components could be created and used by non-Modelica simulators too.

The authors explicitly invite interested colleagues for discussions about the interface proposal. A detailed specification is available on requested.

7 References

[1] http://www.simulationx.com

[2] http://www.gtisoft.com

[3] Modelica Association: Modelica A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version3.0, September 5th 2007.

[4] The MathWorks: Writing S-Functions (Manual), 2002.

[5] http://www.simpack.com

[6] http://www.cosateq.de

Two Steady State CHP Models with Modelica : Mirafiori overall Model and Multi-configuration Biomass Model

Baligh EL HEFNI Benoît BRIDE Bruno PECHINE

baligh.el-hefni@edf.fr benoit.bride@edf.fr bruno.pechine@edf.fr

EDF R&D
6 Quai Watier
F-78401 CHATOU CEDEX
FRANCE

Keywords : *Power Plant, CHP Models, Biomass Model Steady State Modelling, Combined Cycle, Inverse Problems*

Abstract

Steady state 0D/1D models are useful to check, validate and improve through simulation the energy performances of existing heat and/or power plants. They are also used to find the best design that meets required economical criteria.
A library of fully static 0D thermal-hydraulics component models was built. It contains the models of a grid furnace, gas combustion chamber, electrical boiler, steam boiler, multifunctional heater, waterwall gas/water steam exchangers, tubular air heater, steam turbine, condenser, aero-condenser, pump, drum, valves, pipes, gas turbine, compressor, kettle boiler, mixer and splitter etc...
This library now enables us to build models of any CHP plant. A 0D steady state model of the MiraFiori heat and power plant was built in order to check, validate and improve the energy performances of the plant. A multi configurations steady state model of a combined heat and power biomass plant was built, the plant satisfies the steam demand during all the year and produces electricity with its remaining energy.
Models were built by connecting the component models in a technological way, so that its topology reflects the process flow diagram of the plant.

A preliminary calibration of the Mirafiori model was made based on measurement data obtained from on-site sensors and using inverse calculations. The best steam cycle configuration for the Biomass CHP plant was chosen computing various normal conditions points. The models were 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. They converge very quickly, provided that the iteration variables are properly fed in by the user (approx. 5% of the total number of variables).

1 Introduction

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

Steady state 0D/1D models are useful to check, validate and improve through simulation the energy performances of existing heat and/or power plants. They are also used to find the best design that meets required economical criteria.
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.

For present and future models, we are using MODELICA modelling tool. New blocks and models are being developed with Modelica and standard guidelines have been adopted for power plants modelling. It is now used at EDF-R&D as well as in Engineering Departments.
Modelica models are used by EDF to improve its knowledge about existing or future types of power plants, check the design performances and understand important transients situations.

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

Two Steady State CHP models with Modelica - Mirafiori overall model and Multi-configuration Biomass model - were built in 2007.
The modelling and simulation were carried out with the commercial tool Dymola, as it is the most advanced Modelica based tool up to now.

2 Modelling practices at EDF

Modelling and simulation play a key role in the design phase and performance optimization of complex energy processes. At EDF, modelling and simulation of the plant was originally carried out with LEDA. LEDA is a tool developed and maintained by EDF for the modelling and simulation of normal or incidental operation in nuclear and fossil-fuel power plants. LEDA models are used by researchers and engineers in order to improve their knowledge of existing or future types of power plants, to check the design performances and to understand important transient situations.

EDF traditionally used steady state models in order to check precisely the performances and the design given by manufacturers. EDF used dynamic models to check automation and operating procedures and to optimise design for a specific operation.

In order to improve the performance of its simulation tools while reducing their cost, EDF R&D made the decision to replace LEDA with Modelica and the commercial tool Dymola.

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.
- Innovative technologies.

3 EDF Modelica Library

3.1 Component models

A library of fully static 0D thermal-hydraulics component models was built. It contains the models of a grid furnace, gas combustion chamber, boiler, electrical boiler, steam boiler, multifunctional heater, waterwall gas/water steam exchangers, tubular air heater, steam turbine, condenser, aero-condenser, pump, drum, valves, pipes, gas turbine, compressor, kettle boiler, mixer and splitter etc...

The model equations take into account the non-linear and the state-of-the-art physical behaviour of each important phenomenon.

3.2 The thermodynamic properties

Properties of flue gases

The thermo-physical properties of the flue gases (for the exchangers, gas turbines, compressors, gas combustions chambers, …..) were computed using Fortran subroutines called MONOMELD.

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.

4 The Mirafiori model

Steady state model of the MiraFiori heat and power plant was built in order to check, validate and improve the energy performances of the plant. The model contains six units (systems) of production :

- HP water/steam cycles with 3 gas boiler,
- IP water/steam cycles with 4 gas boiler,
- 2 combined cycles,
- 2 GT.

As it has already been mentioned, MiraFiori is a fully static 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 7 in the appendix). It is composed of 420 elementary models, generating 9560 variables and 1950 non-trivial equations.

The model is composed of : 7 gas boilers,14 exchangers, 10 steam turbine stages, 15 pumps, 28 pressure drops, 4 gas turbines, 4 compressors, 4 kettles boilers, 4 gas combustions chambers, several mixers, several collectors and several boundary conditions.

It is very important to provide an efficient way to handle the iteration variables, as the task of setting them properly is time consuming. It is by no way automatic, since it requires a good expertise of the problem to be solved (the number of iteration variables represent roughly 5% of the total number of variables).

4.1 Model calibration

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

A preliminary calibration of the model was made based on 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 main computed performance parameters are :

- the ellipse law coefficients of the turbines,
- the isentropic efficiencies of the turbines,
- the pressure drop correction coefficients of the exchangers and of the pipes between pieces of equipment.
- the compression ratio of the GTs.

Etc.

The main computed outputs are :

- fuel mass flow rate of gas boilers,
- Air mass flow rate of gas boilers,
- thermal power of exchangers,
- temperatures and pressures in places where no sensor are installed.

Etc.

4.2 Simulation results

After calibration, the model allowed us to make what-if simulation and provide to the plant operators :

- The performances of the equipments (for example boiler performances),
- The global efficiencies of the water/steam cycles,
- The gains or extra costs associated with the varying operating conditions of the unit (condenser pressure, exhaust temperature, excess air, fouling coefficients…),
- The best operating point with respect to the various operating conditions of the unit.

4.3 Sensitivity analysis

Then, the model allowed us to make a sensitivity analysis of the effect of air mass flow rate (excess air), ambient air temperature (combustive), temperature of the exhaust flue gases and the condenser vacuum on the thermo-hydraulic behaviour of the power plant and the efficiencies of boilers.

Figure 1 shows the evolution of the efficiencies of the boilers as a function of the ambient air temperature calculated through Dymola, the variation of the efficiencies of the boilers is +/-1% compared to the nominal value.

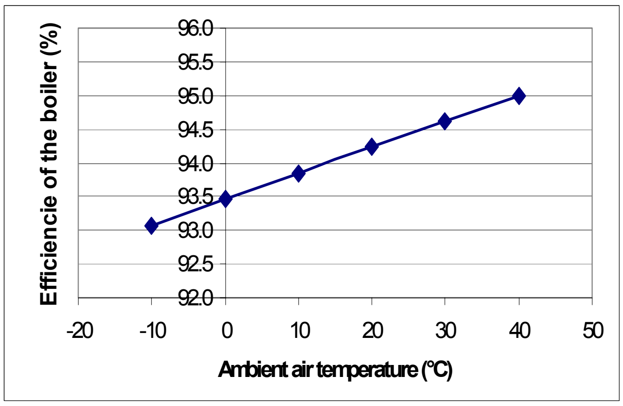


Figure 1 - Efficiencies of the boilers as a function of the ambient air temperature

Figure 2 shows the efficiencies of the boilers as a function of the excess air, the boilers efficiencies vary from 94,4% down to 92% when the excess of air passes from 10% (nominal value) to 90% (maximum value recorded on the operating data).

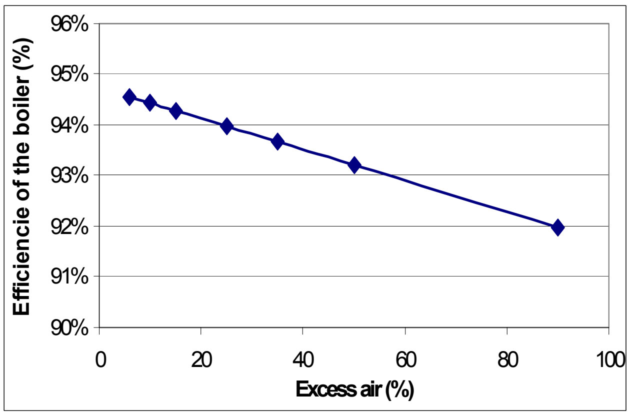


Figure 2 - Efficiencies of the boilers as a function of excess air

Figure 3 shows the efficiencies of the boilers as a function of temperature of the exhaust flue gases, the boilers efficiencies decreases by 2% when the temperature of the exhaust flue gases passes from 110 °C (nominal value) to 150 °C (maximum value recorded on the operating data).

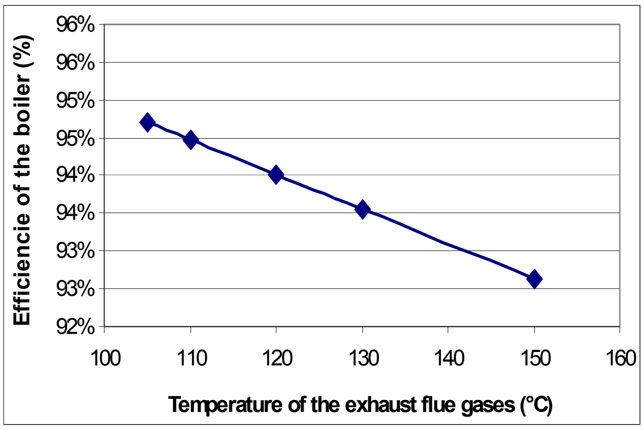


Figure 3 - Efficiencies of the boilers as a function of the temperature of the exhaust flue gases

Figure 4 shows the evolution of the power of gas turbines of the combined cycles as a function of ambient air temperature. The nominal value of power of gas turbines is 80.5 MW for ambient air temperature at 20 °C.

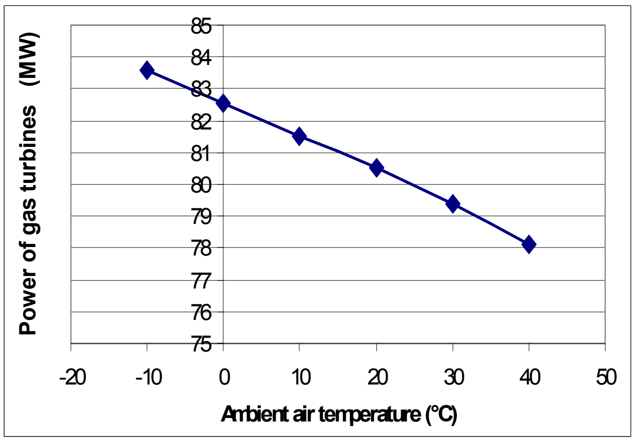


Figure 4 - Power of gas turbines of the combined cycles as a function of the ambient air temperature

Figure 5 shows the evolution of the steam turbine power of the combined cycles as a function of the condenser vacuum, the loss of the steam turbine power is about 7,5 MW, between a condenser vacuum of 50 mbar and a vacuum of 250 mbar.

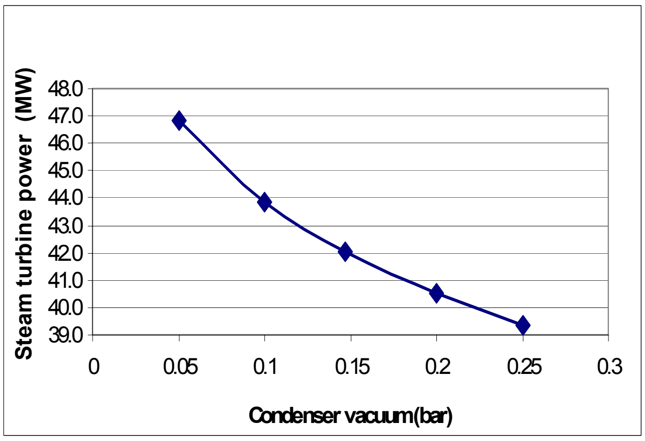


Figure 5 - Evolution of the steam turbine power of GT as a function of the condenser vacuum

4.4 Correction curves

The correction curves used to forecast the behavior of the pieces of equipment. These correction curves represent a simplified physical model of the plant, which is fed into a mathematical model used to compute on a six-week period the cheapest operating scenario which meets environmental and technical requirements.

The different correction curves create with the model are:

- **Gas boiler :** (Boiler Power / Fuel Power) ,
- **Steam turbine :** (Mechanical Power / Boiler Power) ,
- **Gas turbine :** (Mechanical Power / Fuel Power) ,
- **Combined cycle :** (Total mechanical Power / Fuel Power).

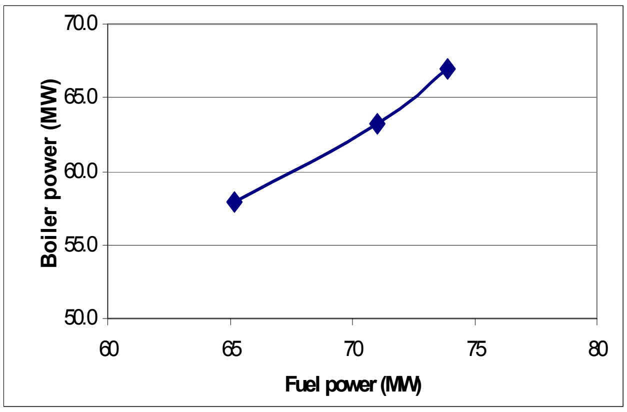


Figure 6 - Example of correction curve : Evolution of the boiler power as a function of the fuel power

5 Biomass CHP steady state model

Recent developments of environmental concerns drove states to promote renewable energies and energy efficient solutions. Some invitation to tender often are proposed so as to create new biomass CHP plants at the best operating cost.

5.1 Need

Companies answering to these invitations to tender for biomass CHP plants shall be able to choose the best configurations for the plants in order to reach the following criteria:

- The yearly average efficiency (steam + electricity) is greater than 50%;
- The plant is able to satisfy the steam demand of the customer (usually an industry) at all time;
- The yearly biomass consumption is fixed;
- The return on investment time is as low as possible.

Usual studies for this type of issue only give an efficiency at nominal point for one or two plant configuration. Models are able to provide various configurations and what-if studies in order to broaden the range of efficiency calculations and help the company to choose the best investment.

One of these companies asked us to assist them by creating and using a MODELICA Biomass CHP plant.

5.2 Building the model

This model uses the same library as the Mirafiori one. It is a fully static model. It also needs to use the same physical properties as Mirafiori.

The full model is built by connecting the component models in a technological way, so that its initial topology reflects the functional schema of the more complex plant (see Figure 7C in the appendix).

In order to be able to answer to many different situations, we created some variables in some of the component model enabling to switch itself on or off.

This multi configurations steady state model of a combined heat and power biomass plant contains 96 elementary models, generating 2162 variables and 460 non-trivial equations.

5.3 Multi – configuration calculations at normal operating condition

First the model is able to give figures at nominal point for various situations.

The same model can simulate 16 different plant configurations:

- w/wo air heater
- w/wo reheaters
- w/wo water heating
- w/wo condenser

NB: any fuel can be set into the grid furnace, but its physical equations are ideal for solid fuels (coal, waste, biomass etc.).

The plant works with a fixed biomass flow rate, it satisfies the steam demand during all the year and produces electricity with its remaining energy.

We make an inverse calculation (such as the calibration phase for Mirafiori model) with DYMOLA setting the nominal parameter to their expected value in the plant projects.

The results given by the model are :

- The efficiency at nominal point (steady state calibration),
- The electric power produced

These results at nominal point are a first step to choose the best configuration regarding the investment cost of each type of plant.

5.4 What-if steam demand varies?

Of course, the results given at nominal point are not consistent to know precisely the average performance on a one-year operation.

Consequently, we use what-if ability of DYMOLA/MODELICA model in order to realize the following computations :

- What-if simulation varying any parameter: e.g. steam flow rate,
- Economic study on a one-year typical steam demand (what-if quasi-static simulation).

The forecast of steam demand is defined as a load curve with 365 values of flow rate (one per day). It is based on measurements made by the customer on a past year considered as normal. The variation of the steam flow rate makes the global efficiency vary and changes the electric power produced.

Hence the best yearly average figures (global efficiency, electric power) are given by the model.

It gives a much better forecast of the incomes that will be generated by the plant.

5.5 Creation of a tool for non-modeller

The executable file of the model has been integrated in an easy-to-use Excel sheet for non-modelers, and it was given to our customer.

With this tool, one who is not used to models can make calculations on any plant configuration and launch what-if calculation varying steam demand.

5.6 Trigeneration issues

An absorption chiller model is being created in the static library. This could represent one-stage or two stage Water/LiBr systems on hot water or hot flue gas.

This will give us the ability to model trigeneration systems in order to compute performance figures for existing and projected plants and to simulate various behavior.

The optimal point, harder to find for a trigeneration than for a CHP, will easily be found with a DYMOLA/MODELICA model.

Conclusion

Two Steady State CHP Models were built with Modelica to evaluate the capacity of Modelica based tools to perform steady state direct and inverse computations for the sizing of power plants.

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] El Hefni B., Bouskela D. Modelling of a water/steam cycle of the combined cycle power plant “Rio Bravo 2” with Modelica “Modelica 2006 conference“

[2] Souyri A., Bouskela D. Pressurized Water Reactor Modelling with Modelica “Modelica 2006 conference“

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

Appendix

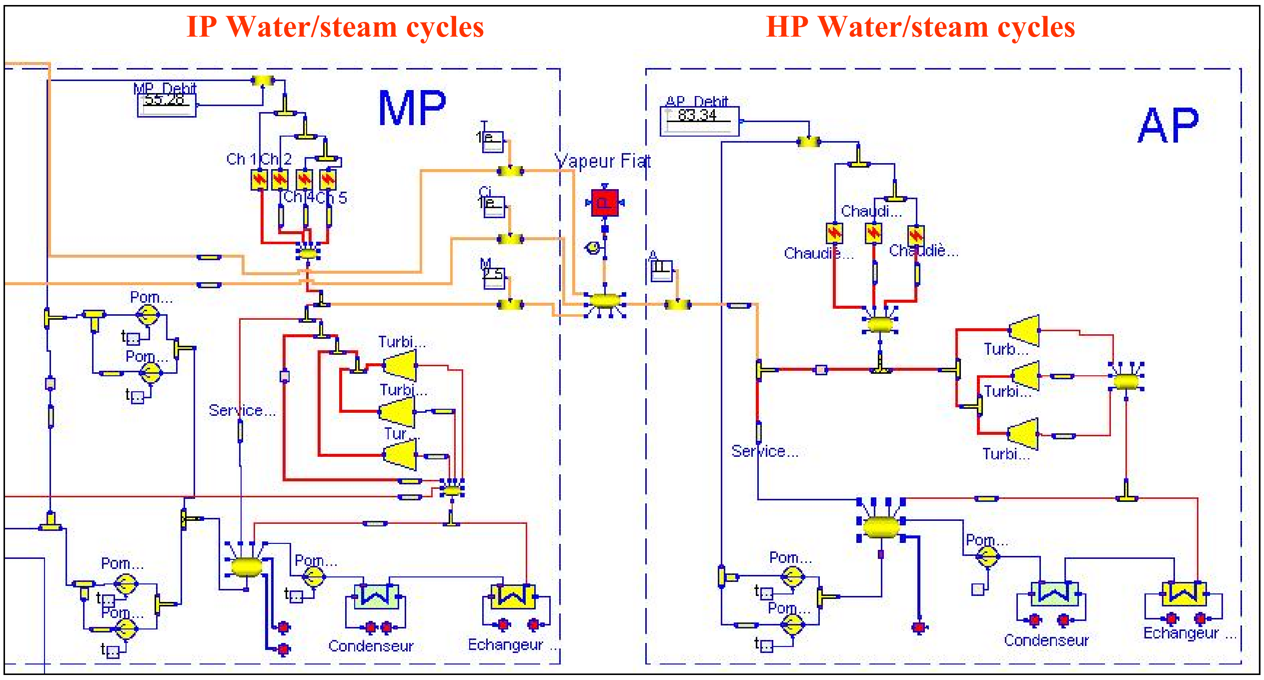


Figure 7A - Parts of the Dymola model of "Mira-Fiori

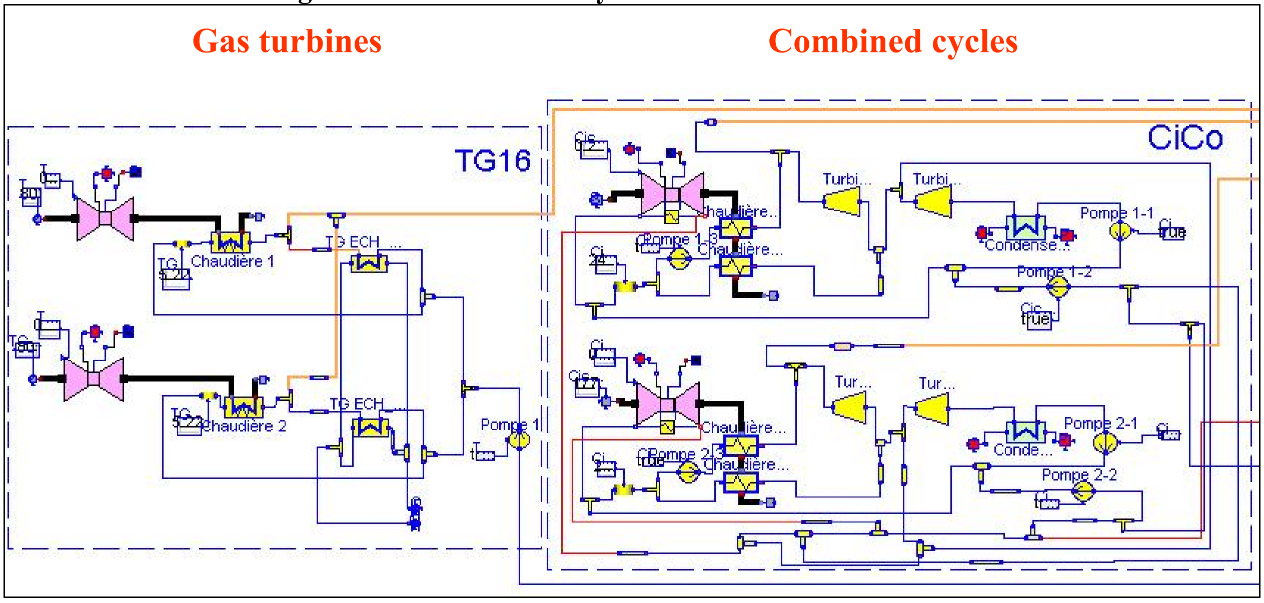


Figure 7B - Parts of the Dymola model of "Mira-Fiori

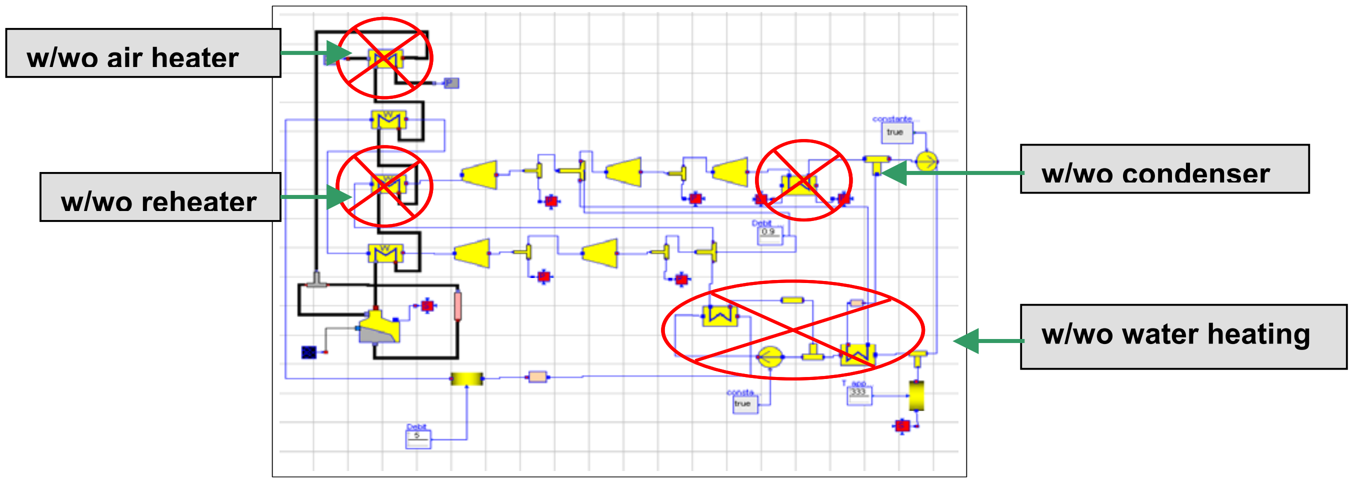


Figure 7C Dymola steady state model of a biomass CHP plant

Efficient Analysis of Harmonic Losses in PWM Voltage Source Induction Machine Drives with Modelica

Johannes V. Gragger Anton Haumer Christian Kral Franz Pirker
Arsenal Research
Giefinggasse 2, 1210 Vienna, Austria

Abstract

This paper presents an approach to calculate the copper and core losses caused by harmonics of the PWM of a voltage source inverter. For the analysis some models of the *Smart Electric Drives* (SED) library, and additionally, a *Modelica* library for modeling AC circuits by means of electric time phasors, are used. With the proposed analysis the influence of space phasor PWM signals on the machine efficiency is investigated. A *Modelica* model of a speed controlled induction machine drive working at different load points and different switching frequencies is presented. The results of the simulation are compared and discussed.

Keywords: induction machine, inverter, speed controlled drive, efficiency, copper losses, core losses, space phasor PWM, SED library

1 Introduction

In most variable speed drives pulse width modulation (PWM) voltage source inverters are used. Usually machine design tools only consider the fundamental harmonic of the stator voltage when calculating the losses. The major aim of the presented work is to investigate the negative impact of PWM switching on additional losses in the machine windings and the iron cores. These additional losses are caused by harmonics of the voltage and the current due to the PWM. The harmonic losses of the induction machine are modeled using the AC library, which is based on the stationary analysis with complex time phasors [1].

A number of algorithms for PWM voltage generation are available. Some well known techniques are unipolar voltage switching and bipolar voltage switching [2], harmonic elimination [3] and space vector PWM [4]. In fact there are many more techniques in which the basic principles of the ones mentioned are used with some modifications. Different PWM algorithms cause different voltage harmonics. These voltage harmonics give rise to current harmonics due to the machine impedance. The voltage harmonics cause additional core losses whereas the current harmonics cause additional losses in the stator and rotor winding of the machine. Moreover, the frequency of the carrier signal has a significant influence on the voltage and current spectra and consequently increases the losses arising in the machine. It is widely accepted that PWM switching has a negative impact on the efficiency of the drive and some efforts had been undergone to calculate the amount of losses caused by PWM switching.

In [5, 6, 7, 8, 9] finite element analysis (FEA) techniques are implemented, which require high computational expenses for calculating the additional losses. In an FEA model the ohmic heat losses due to the PWM switching are inherently covered. The additional core losses are computed by a frequency and flux dependent model, which is evaluated locally throughout the machine volume.

Alternatively, the harmonic losses can be assessed keeping the processing efforts low by defining specific loss factors [10]. In this case it is crucial to keep the energy balance between the electric terminals and the shaft of the machine consistent.

In this paper the energy balance is implemented straight forward by defining an equivalent circuit [11, 12, 13, 14]. The presented work is based solely on analytical equations using data from conventional induction machine calculation programs without FEA. An equivalent circuit with elements taking deep bar effects and the influence of the stator voltage and the stator frequency on iron losses into account is used to calculate the harmonic losses with the principle of superposition. The proposed models are designed such way that it takes only little effort to replace the PWM algorithm by an alternative one and to change machine

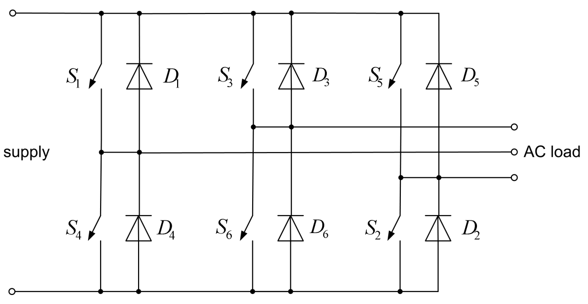


Figure 1: Three phase full bridge.

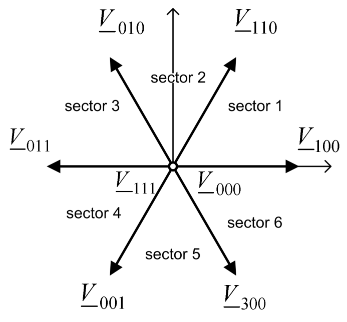


Figure 2: Possible voltage space vectors of a three phase space vector PWM.

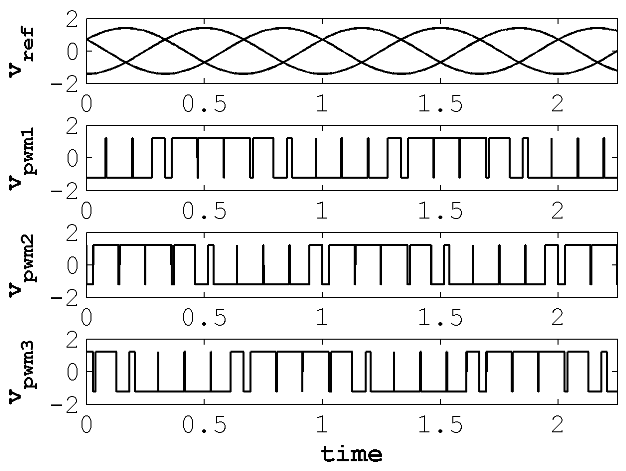


Figure 3: Reference signals and resulting PWM signals, v_{PWM}, of the investigated space vector PWM algorithm.

data in order to benchmark different variable speed drive setups.

2 PWM voltage generation

The PWM waveform depends on the control unit and the converter topology. In this work one of the most commonly used PWM waveforms is analyzed, the space vector PWM. Space vector PWM can be implemented if a three phase converter of the topology shown in fig. 1 is used. The states of the six switches (S_1 to S_6) must be chosen such way that the switches of one leg of the converter switch complementary. Neglecting dead times, it must hold that whenever one switch of a leg is ON the other one must be OFF. By no means both can be ON at the same time.

There are eight possible combinations for the switch commands, which result in seven elementary output voltage space vectors as shown in fig. 2. By using PWM for switching between these seven elementary space vectors any space vector position can be realized. The output phase voltages of a space phasor PWM controlled three phase full bridge are shown in fig. 3. Using the models of ideal switching converters and the respective PWM control blocks from the SED library [15, 16], it is possible to compare the ”quality” of PWM signals with different switching frequencies (and with different switching algorithms). In fig. 4 the spectra of space vector PWM with two different frequency ratios are shown where v_{ref} is the phase voltage amplitude of the reference space vector rotating with constant angular speed and magnitude and f_{ref} is the frequency of the phase voltage. The frequency per unit (p.u.) is $\frac{f}{f_{ref}}$ and the voltage p.u. is $\frac{v}{v_{ref}}$. It appears that space vector PWM with high switching frequency, f_{Switch}, causes considerably lower harmonics with low order numbers than space vector PWM with low switching frequency. If the spectrum of the PWM voltage signal and the frequency dependent impedances of the machine are known the harmonic copper losses and the harmonic core losses can be calculated. It can be shown that high switching frequencies help decreasing the iron losses and the copper losses in voltage source inverter drives.

3 Model of the copper losses

The copper losses in an induction machine can be determined by using the well known single phase equivalent circuit [17]. Figure 5 shows the *Modelica* model of the investigated induction machine. This equivalent circuit represents the machine behavior in steady state operation. The connectors used in the equivalent circuit model contain complex current time phasors as flow variables and complex voltage time phasors as potential variables. Furthermore, the reference frame of the time phasors is defined by a reference angle φ in the connectors. For the calculation of the copper losses the deep bar effects of the rotor stray inductance and

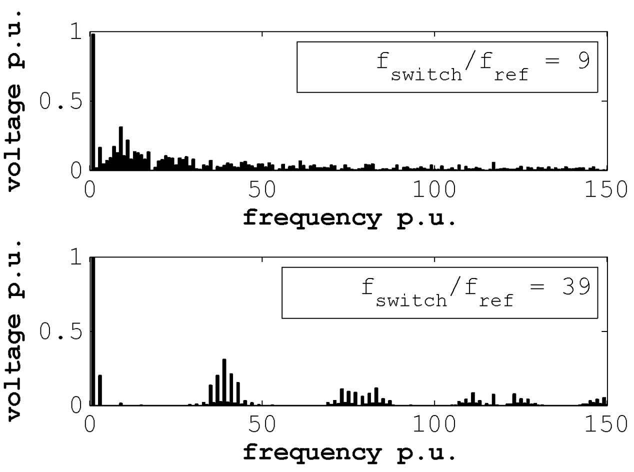


Figure 4: Spectra of space vector PWM voltages with low and with high switching frequency.

the rotor resistance are considered. Skin effects in the stator resistance and stator inductance are neglected because they can be mostly avoided through wires with small radial length in the stator winding [18]. Saturation effects are neglected as well.

The slip with respect to a certain harmonic depends on the orders of this harmonic. It can be shown that the rotational directions of the spatial harmonic waves of the stator field are dependent on the order number [19]. Therefore the slip related to the different voltage harmonics,

$$s_\nu = \frac{\omega_\nu - \omega_m}{\omega_\nu} \tag{1}$$

where ω_m is the shaft speed of an equivalent two pole machine and the angular velocities of the harmonic waves of the stator quantities,

$$\omega_\nu = \omega_1 \cdot \nu. \tag{2}$$

Using (1) and (2) the slip can be written as

$$s_\nu = 1 - \frac{1 - s_1}{\nu}. \tag{3}$$

In a symmetric induction machine with m phases fed by PWM voltages the order numbers of the harmonics of the stator field [20] are

$$\nu = 2 \cdot m \cdot k + 1, \tag{4}$$

where

$$k = \{0, \pm 1, \pm 2, \pm 3, ...\}. \tag{5}$$

It is well known that

$$R'_{r\nu} = \frac{R_{r\nu}}{s_\nu} = R_{r\nu} + R_{mech\nu} \tag{6}$$

with

$$R_{mech\nu} = R_{r\nu}\frac{1 - s_\nu}{s_\nu}. \tag{7}$$

For each harmonic order, the power dissipated by $R_{r\nu}$ represents the copper losses in the rotor and the power dissipated by $R_{mech\nu}$ represents the mechanical power of the machine distributed to the shaft (without considering stray load losses) [17].

According to [21] the deep bar effects in rotor bars with rectangular profile can be considered by a resistance factor

$$K_{R\nu} = \xi_\nu \cdot \frac{\sinh(2\xi_\nu) + \sin(2\xi_\nu)}{\cosh(2\xi_\nu) - \cos(2\xi_\nu)} \tag{8}$$

and an inductance factor

$$K_{I\nu} = \frac{3}{2\xi_\nu} \cdot \frac{\sinh(2\xi_\nu) - \sin(2\xi_\nu)}{\cosh(2\xi_\nu) - \cos(2\xi_\nu)}, \tag{9}$$

with

$$\xi_\nu = h \cdot \sqrt{\frac{\mu_0 \cdot 2\pi f_\nu}{2\rho} \cdot \frac{b}{b_s}}. \tag{10}$$

The subsidiary quantity ξ_ν is a function of the bar height, h, the frequency of the voltage harmonic, f_ν, the specific resistance of the rotor bars, ρ, the width of the rotor bar, b, and the width of the rotor slot, b_s. Hence, the rotor resistance,

$$R'_{r\nu} = K_{R\nu} \cdot R'_{r,var} + R'_{r,const}. \tag{11}$$

In (11) the constant resistance, $R'_{r,const}$, represents the end rings and the parts of the rotor bars that are not embedded in the slots whereas the variable resistance, $K_{R\nu} \cdot R'_{r,var}$, represents the parts of the rotor bars that are embedded in the slot.

The rotor stray inductance is modeled the same way:

$$L'_{r\sigma\nu} = K_{I\nu} \cdot L'_{r\sigma,var} + L'_{r\sigma,const} \tag{12}$$

By calculating the stator current, $\underline{I}_{s\nu}$, and the rotor current, $\underline{I}_{r\nu}$, of the single phase equivalent circuit for all harmonics, the stator and rotor copper loss increase due to the harmonics can be expressed by

$$p_{Cu,s\Sigma\nu} = \frac{\Sigma(|\underline{I}_{s\nu}|^2)}{|\underline{I}_{s1}|^2} - 1 \tag{13}$$

and

$$p_{Cu,r\Sigma\nu} = \frac{\Sigma(R_{r\nu} \cdot |\underline{I}'_{r\nu}|^2)}{R_{r1} \cdot |\underline{I}'_{r1}|^2} - 1 \tag{14}$$

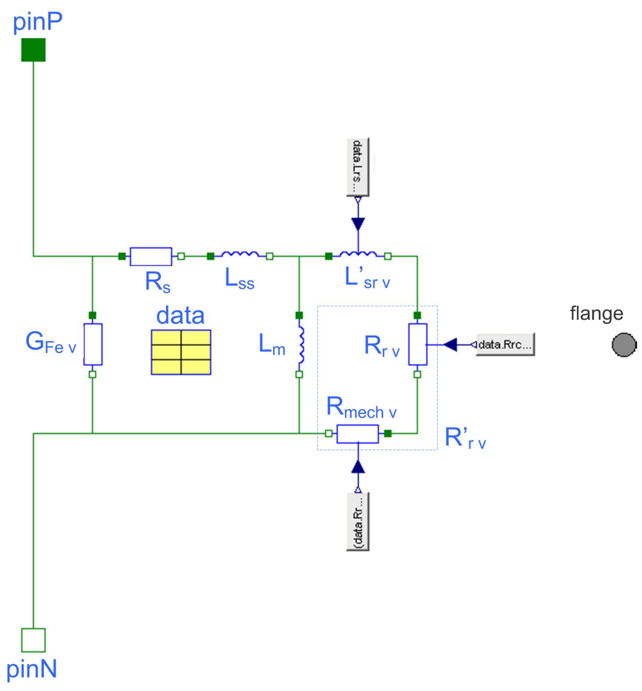


Figure 5: Equivalent circuit of an induction machine implemented with *Modelica*.

Consequently, the total harmonic copper losses are

$$P_{Cu\Sigma\nu} = P_{Cu,s1} \cdot p_{Cu,s\Sigma\nu} + P_{Cu,r1} \cdot p_{Cu,r\Sigma\nu} \qquad (15)$$

where $P_{Cu,s1}$ and $P_{Cu,r1}$ are the stator and rotor copper losses with respect to the fundamental wave.

4 Model of the core losses

The core losses, P_{Fe}, in an induction machine can be divided into two parts: the hysteresis losses and the eddy current losses [22, 23, 24, 25]. Hysteresis losses, $P_{Fe,h}$, and eddy current losses, $P_{Fe,e}$, can both be expressed as functions of the magnetic flux linkage, ψ, and the stator frequency, f:

$$P_{Fe,h} = F_h\{\psi^2, f\} \qquad (16)$$

$$P_{Fe,e} = F_e\{\psi^2, f^2\}. \qquad (17)$$

Considering that the voltage is directly proportional to the flux linkage and the frequency according to

$$V_\nu = \psi_\nu \cdot \omega_\nu \qquad (18)$$

the hysteresis losses and eddy current losses caused by the harmonics of the stator voltage can be calculated per unit to

$$p_{Fe,h\nu} = [(\frac{f_1 \cdot V_{s\nu}}{f_\nu \cdot V_{s1}})^2 \cdot \frac{f_\nu}{f_1}] = \frac{f_1 \cdot V_{s\nu}^2}{f_\nu \cdot V_{s1}^2} \qquad (19)$$

$$p_{Fe,e\nu} = [(\frac{f_1 \cdot V_{s\nu}}{f_\nu \cdot V_{s1}})^2 \cdot (\frac{f_\nu}{f_1})^2] = \frac{V_{s\nu}^2}{V_{s1}^2}. \qquad (20)$$

In the equivalent circuit shown in fig. 5 the hysteresis and the eddy current losses are both considered in one conductor $G_{Fe\nu}$. Using the hysteresis losses, $P_{Fe,h1}$, and the eddy current losses, $P_{Fe,e1}$, of the fundamental together with (19) and (20),

$$G_{Fe\nu} = \frac{P_{Fe,h1}}{3 \cdot V_{s\nu}^2} \cdot p_{Fe,h\nu} + \frac{P_{Fe,e1}}{3 \cdot V_{s\nu}^2} \cdot p_{Fe,e\nu}. \qquad (21)$$

Hence, the total harmonic core losses

$$P_{Fe\Sigma\nu} = 3 \cdot \sum (G_{Fe\nu} \cdot V_{s\nu}^2) \qquad (22)$$

which can also be written as

$$\begin{aligned} P_{Fe\Sigma\nu} &= P_{Fe,h1} \cdot \{[\sum p_{Fe,h\nu}] - 1\} + \\ &+ P_{Fe,e1} \cdot \{[\sum p_{Fe,e\nu}] - 1\}. \end{aligned} \qquad (23)$$

5 Simulation setup

In mining, chemical, waste water, gas or oil industries there are high-power medium-voltage variable speed drives used that work with IGCT or IGBT converters. Such IGCT converters have switching frequency ranges around 1 kHz and IGBT converters work with frequencies up to 10 kHz [26].

In this paper a high-power medium-voltage water pump drive is simulated. The specifications of the investigated induction machine are shown in table 1. The spectra of the voltage waveforms generated by space vector PWM are calculated using the SED library and the *Modelica* Standard library in a Dymola simulation environment. In fig. 6 the model calculating the harmonic components from the inverter voltage is shown. The model contains three ideal reference voltage signals, a block generating the PWM switching commands, and a model representing a three phase full bridge with integrated DC-link voltage source as well as a Fourier analysis block. In the Fourier analysis block the Fourier coefficients, a_k and b_k, get calculated according to the Euler-Fourier formulas [27] with two integrators and a sine and a cosine signal source. The spectral components are computed by converting the Cartesian coordinates, a_k and b_k, to polar coordinates, d_k and φ_k. From the generated spectral components the harmonic losses are processed through a

Table 1: Parameters of the high-power medium-voltage induction machine.

Induction Machine		
Pole Pairs		2
Nominal Power	[kW]	1600
Nominal Frequency	[Hz]	50
Nominal Voltage	[V]	6000
Nominal PF		0.873
Nominal Slip	[%]	0.25

Fourier synthesis in the model shown in fig. 7. The encapsulated induction machine model applied for loss calculation is depicted in fig. 5. It is fed with an array of stator voltage time phasors.

In the induction machine model the inner torque components, $T_{inner\nu}$, with respect to a harmonic, ν, are computed by

$$T_{inner\nu} = p \cdot \frac{P_{input\nu} - (P_{Cu,s\nu} + P_{Fe\nu})}{(2 \cdot \pi \cdot f_\nu)}, \tag{24}$$

where p is the number of pole pairs, $P_{input\nu}$ are the electrical input power components of the machine, $P_{Cu,s\nu}$ are the stator copper loss components, $P_{Fe\nu}$ are the iron loss components, and f_ν are the harmonic stator frequencies. The shaft speed of the induction machine is controlled by an integral action controller such way that the fundamental component of the mechanical power

$$P_{airgap1} = T_{inner1} \cdot (2 \cdot \pi \cdot f_1), \tag{25}$$

matches the reference power, P_{ref}. Since the friction losses, P_{fr}, are not considered in the equivalent circuit, the reference power with respect to the shaft of the machine model is determined by

$$P_{ref} = P_{m,ref} + P_{fr}, \tag{26}$$

where $P_{m,ref}$ is the desired mechanical power of the induction machine. If the actual power, P_{real}, at the shaft of the machine model matches P_{ref}, the desired operation point is reached.

The harmonic losses of the inverter drive are computed for two different operation points of the machine. In case A the machine is operated at nominal supply frequency, $f_{Nominal}$, and nominal mechanical power, $P_{Nominal}$. In case B the machine is simulated at $\frac{f_{Nominal}}{3}$ and $\frac{P_{Nominal}}{27}$. The reason for assessing case B is to investigate the influence of the harmonic losses specifically in variable speed drives that are connected

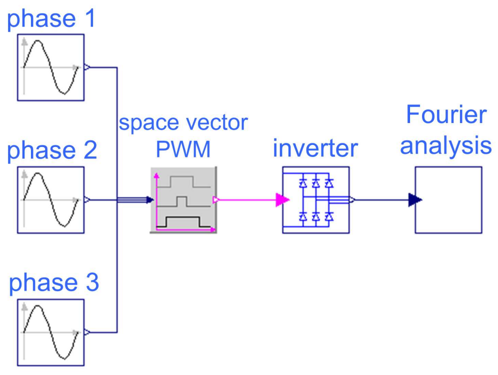


Figure 6: Model used for the PWM signal analysis.

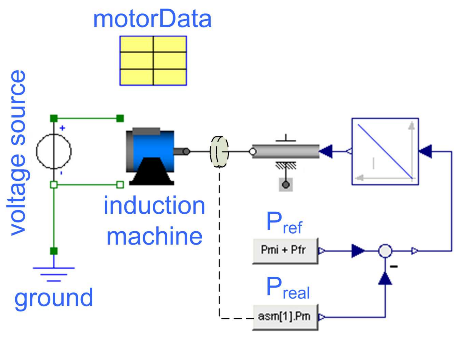


Figure 7: Model of the PWM voltage source induction machine drive.

with mechanical loads such as pumps or fans. Many of these loads have an approximately quadratic speed dependent load torque characteristic.

Besides the variation of the operation point also the converter switching frequency is varied. The harmonic spectra of the PWM voltages with switching frequencies of 1950 Hz, 1050 Hz and 450 Hz are calculated and fed to the machine model.

6 Simulation Results

In table 2 the simulation results of the machine fed with space vector PWM are presented. The investigations show that increasing the PWM switching frequency decreases the total harmonic core losses. The total harmonic core losses at a switching frequency of 450 Hz are about 60% higher than at a switching frequency of 1950 Hz. It can also be seen that the total harmonic copper losses rise much more than the total harmonic core losses. In case A, for instance, the total harmonic copper losses become more than ten times higher if the switching frequency gets decreased from

Table 2: Modelica simulation results of the high-power medium-voltage induction machine drive.

		Case A: $f = f_{Nominal}$ $P = P_{Nominal}$	Case B: $f = \frac{f_{Nominal}}{3}$ $P = \frac{P_{Nominal}}{27}$
Switching Frequency = 1950 Hz			
Machine Output Power (Fundamental)	[W]	1600000.00	59259.30
Shaft Speed	[rpm]	1496.34	499.62
Stator Current (Fundamental)	[A]	179.85	49.19
Power Factor (Fundamental)		0.87	0.38
Friction Losses	[W]	5566.00	1996.00
Core Losses (Fundamental)	[W]	10706.50	3039.31
Stator Copper Losses (Fundamental)	[W]	8453.70	630.36
Rotor Copper Losses (Fundamental)	[W]	3928.22	47.19
Core Losses (Harmonics)	[W]	582.59	377.65
Stator Copper Losses (Harmonics)	[W]	20.84	5.46
Rotor Copper Losses (Harmonics)	[W]	436.41	152.45
Efficiency (Fundamental)	**[%]**	**98.24**	**91.21**
Efficiency considering Harmonics	**[%]**	**98.18**	**90.46**
Switching Frequency = 1050 Hz			
Machine Output Power (Fundamental)	[W]	1600000.00	59259.30
Shaft Speed	[rpm]	1496.32	499.62
Stator Current (Fundamental)	[A]	180.33	49.18
Power Factor (Fundamental)		0.87	0.38
Friction Losses	[W]	5566.00	1996.00
Core Losses (Fundamental)	[W]	10651.60	3037.93
Stator Copper Losses (Fundamental)	[W]	8499.40	630.16
Rotor Copper Losses (Fundamental)	[W]	3952.34	47.21
Core Losses (Harmonics)	[W]	678.37	561.57
Stator Copper Losses (Harmonics)	[W]	73.31	21.81
Rotor Copper Losses (Harmonics)	[W]	1115.07	463.48
Efficiency (Fundamental)	**[%]**	**98.24**	**91.21**
Efficiency considering Harmonics	**[%]**	**98.13**	**89.76**
Switching Frequency = 450 Hz			
Machine Output Power (Fundamental)	[W]	1600000.00	59259.30
Shaft Speed	[rpm]	1496.17	499.61
Stator Current (Fundamental)	[A]	183.42	49.14
Power Factor (Fundamental)		0.87	0.38
Friction Losses	[W]	5566.00	1996.00
Core Losses (Fundamental)	[W]	10312.80	3029.27
Stator Copper Losses (Fundamental)	[W]	8796.23	628.91
Rotor Copper Losses (Fundamental)	[W]	4108.64	47.35
Core Losses (Harmonics)	[W]	917.97	763.86
Stator Copper Losses (Harmonics)	[W]	461.05	125.53
Rotor Copper Losses (Harmonics)	[W]	4429.64	1780.97
Efficiency (Fundamental)	**[%]**	**98.23**	**91.22**
Efficiency considering Harmonics	**[%]**	**97.88**	**87.62**

1950 Hz to 450 Hz. The results also show that the harmonic stator copper losses only make up for a small share of the entire losses caused by the PWM harmonics. The biggest parts of the PWM harmonic losses are the harmonic rotor copper losses, especially when the drive is operated at low switching frequencies. The overall machine efficiency without stray load losses is also presented in table 2.

It appears that the consideration of harmonic losses only causes an efficiency decrease from 98.24 % to 98.18 % in case A with $f_{switch} = 1950$ Hz. The efficiency decreases from 98.23 % to 97.88 % for $f_{switch} = 450$ Hz. For case B the impact of the harmonics is much higher on the efficiency. At $f_{switch} = 1950$ Hz the efficiency decreases from 91.21 % to 90.46 %. The largest impact on the efficiency is due to a switching frequency $f_{switch} = 450$ Hz.

From this comparison one can conclude that when designing a machine for variable speed drives the PWM harmonic losses should be taken into account, especially if PWM frequencies below 1 kHz are used. Furthermore, the overall efficiency values show that as long as the machine is operated close to the nominal operation point (case A) with switching frequencies above 1 kHz the PWM harmonic losses can be neglected.

7 Conclusions

An analytical approach to calculate the copper and core losses caused by the harmonics of PWM voltages in variable speed induction machine drives is presented. The derived equations are implemented in *Modelica* language applying the AC library for modeling electric circuits by means of time phasors. By using the proposed models the PWM harmonic losses of a high-power medium-voltage induction machine with 1600 kW are calculated. Furthermore, the influence of reduced load and changes in the switching frequency are investigated. The results show that if the switching frequency is low and the machine is likely to be operated at low load points the PWM harmonic losses can decrease the overall efficiency of the machine considerably. Still, as long as the switching frequencies of the PWM are above 1 kHz and the load point does not vary significantly from the nominal load point the PWM harmonic losses can be neglected.

References

[1] A. Haumer, C. Kral, J. V. Gragger, and H. Kapeller, “Quasi-stationary modeling and simulation of electrical circuits using complex phasors”, *International Modelica Conference, 6th, Bielefeld, Germany*, 2008.

[2] Mohan and Robbins, *Power Electronics*, J. Wiley Verlag, New York, 2 edition, 1989.

[3] H. S. Patel and R. G. Hoft, “Generalized techniques of harmonic elimination and voltage control in thyristor inverters: Part I-harmonic elimination”, *IEEE Transactions on Industry Applications*, vol. IA-9, Issue 3, pp. 310–317, 1973.

[4] H.W. van der Broeck, H. C. Skudelny, and G.V. Stanke, “Analysis and realization of a pulsewidth modulator based on voltage space vectors”, *IEEE Transactions on Industry Applications*, vol. 24, No. 1, pp. 142–150, 1988.

[5] T.C. Green, C.A. Hernandez-Aramburo, and A.C. Smith, “Losses in grid and inverter supplied induction machine drives”, *IEE Proceedings - Electric Power Applications*, vol. 150, no. 6, pp. 712–724, 11 2003.

[6] Y. Wu, R.A. McMahon, Y. Zhan, and A.M. Knight, “Impact of PWM schemes on induction motor losses”, *41st IAS Annual Meeting, IEEE Industry Applications Conference*, vol. 2, pp. 813–818, 8-12 Oct. 2006.

[7] C.A. Hernandez-Aramburo, T.C. Green, and S. Smith, “Assessment of power losses of an inverter-driven induction machine with its experimental validation”, *IEEE Transactions on Industry Applications*, vol. 39, Issue 4, pp. 994–1004, 2003.

[8] S. Mukherjee, G.E. Adams, and R.G. Hoft, “FEM analysis of inverter-induction motor rotor conduction losses”, *IEEE Transactions on Energy Conversion*, vol. 4, no. 4, pp. 671–680, Dec. 1989.

[9] A.M. Knight, P.D. Malliband, C.Y. Leong, and R.A. McMahon, “Power losses in small inverter-fed induction motors”, *IEEE International Conference on Electric Machines and Drives*, pp. 601–607, 15-18 May 2005.

[10] A. Boglietti, G. Griva, M. Pastorelli, F. Profumo, and T. Adam, “Different PWM modulation techniques indexes performance evaluation”, *IEEE International Symposium on Industrial Electronics, ISIE’93 - Budapest.*, pp. 193–199, 1993.

[11] V. Kinnares, S. Potivejkul, and B. Sawetsakulanond, “Modified harmonic loss model in PWM fed induction machines”, *IEEE Asia-Pacific Conference on Circuits and Systems, IEEE APCCAS*, pp. 535–538, 24-27 Nov. 1998.

[12] Isao Takahashi and Hiroshi Mochikawa, “A new control of PWM inverter waveform for minimum loss operation of an induction motor drive”, *IEEE Transactions on Industry Applications*, vol. 21, no. 3, pp. 580–587, May 1985.

[13] H.W. van der Broeck and H. Skudelny, “Analytical analysis of the harmonic effects of a pwm ac drive”, *IEEE Transactions on Power Electronics*, vol. 3,2, pp. 216–223, 1988.

[14] E. N. Hildebrand and H. Roehrdanz, “Losses in three-phase induction machines fed by PWM converter”, *IEEE Transactions on Energy Conversion*, vol. 16, no. 3, pp. 228–233, Sept. 2001.

[15] H. Giuliani, D. Simic, J. V. Gragger, C. Kral, and F. Pirker, “Optimization of a four wheel drive hybrid vehicle by means of the SmartElectricDrives and the SmartPowerTrains library”, *The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition, EVS22*, 2006.

[16] J.V. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, and F. Pirker, “The SmartElectricDrives Library – powerful models for fast simulations of electric drives”, *5th International Modelica Conference 2006, Vienna, Austria*, 2006.

[17] R. Fischer, *Elektrische Maschinen*, Carl Hanser, München, 9 edition, 1995.

[18] H. Sequenz, *Die Wicklungen elektrischer Maschinen*, vol. 3, Springer Verlag, Wien, 1954.

[19] G. Müller, *Elektrische Maschinen - Theorie rotierender elektrischer Maschinen*, VEB Verlag Technik, Berlin, 2 edition, 1967.

[20] H. Kleinrath, *Stromrichtergespeiste Drehfeldmaschinen*, Springer Verlag, Wien, 1980.

[21] W. Schuisky, *Berechnung elektrischer Maschinen*, Springer Verlag, Wien, 1960.

[22] Th. Bödenfeld and H. Sequenz, *Elektrische Maschinen*, Springer Verlag, Wien, 7 edition, 1965.

[23] G. Müller, *Elektrische Maschinen - Grundlagen, Aufbau und Wirkungsweise*, VEB Verlag Technik, Berlin, 4 edition, 1977.

[24] V. Del Toro, *Electric Machines and Power Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[25] I. Boldea and A. Nasar, *The Induction Machine Handbook*, CRC Press, London, 2002.

[26] B. Bose, *Power Electronics and Motor Drives*, Elsevier, 2006.

[27] Bronstein and Semendjajew, *Taschenbuch der Mathematik*, B.G. Teubner Verlag, Leipzig, 19 edition, 1979.

Monte Carlo Simulation with Modelica

Joachim Haase Susann Wolf Christoph Clauß
Fraunhofer-Institute for Integrated Circuits, Design Automation Division
Zeunerstraße 38, 01069 Dresden, Germany
{Joachim.Haase, Susann.Wolf, Christoph.Clauss}@eas.iis.fraunhofer.de

Abstract

Monte Carlo simulation allows to obtain statistical information derived from estimates of the random variability of component parameters. The paper demonstrates how to describe the random characteristic of parameters in a tool-independent manner in Modelica. Using the multi-run facilities of a simulation engine statistical analysis can be carried out without any code intervention concerning the tool. The approach is based on the SAE 2748 standard. Solutions of implementation problems with respect to Modelica are discussed. This paper is based on results, which were developed in the Fraunhofer collaborative project "Computer Aided Robust Design (CAROD)".

Keywords: Statistical analysis, SAE 2748, Monte Carlo simulation

1 Introduction

It is more and more required within industrial applications to consider the influence of the variability of design parameters on the behaviour of systems. For instance yield and reliability often depend on the statistical characteristics of such parameters [1].

Monte Carlo methods are widely used to analyze the effects of parameter tolerances. In a Monte Carlo simulation, a mathematical model of a system is repeatedly evaluated. Each run uses different values of design parameters. The selection of the parameter values is made randomly with respect to given distribution functions. Monte Carlo simulation is very time consuming. A lot of simulation runs are required to investigate the behavior of a system subject to the statistical distribution of parameters. Nevertheless, Monte Carlo simulation is very favored in various application areas where an analytical relation between design and system parameters is difficult to find. For example mixed-signal electrical systems consisting of analog and digital components often belong to this class of systems.

The objective of this paper is to make a proposal how to handle the description of random parameters in Modelica in a tool-independent way. Furthermore a way is presented how to carry out a Monte Carlo simulation within an existing simulation engine. It is only required that the simulator supports multiple runs of a simulation task.

The approach is close to the standard J 2748 prepared by the Electronic Design Automation Standards Committee of the Society of Automotive Engineers (SAE) that describes random parameter handling in a VHDL-AMS simulation problem [2, 3]. Describing parameter variations in nearly the same way in VHDL-AMS and Modelica offers the opportunity to reduce the effort to provide random parameter data in the design process and to avoid misunderstandings.

2 SAE-Standard J 2748

Some basic requirements that are supported by the SAE J 2748 standard are summarized in the following. The basic idea is to add information to characterize the parameters. Thus, it should be possible to use existing models also for statistical analysis. In detail it is required

- Usage of the same model for nominal and Monte Carlo analysis
- Possibility to assign different statistical distributions to each constant or parameter
- Support of continuous and discrete distributions
- Permission of user-defined distributions
- Possibility to specify correlation between constants

From a practical point of view the following points should also be mentioned

- Independent random number generation for any constant

- Reproducibility of Monte Carlo simulation within the same simulation tool

Statistical distributions are characterized from an engineering point of view. That means the mathematical parameters as for instance the moments are derived from engineering parameters as nominal value, tolerances, minimum and maximum values. The standard provides implementations of basic regular distribution functions. Futhermore, standard functions are provided that allow to declare user-defined distributions. Also truncated distributions are supported that limit the random numbers to a given interval.

Table 1: Regular distribution functions [3]

UNIFORM	Uniform distributed values
NORMAL	Gaussian distributed values
PWL_CDF	Piecewise-linear description of a cumulative distribution function
PWL_PDF	Piecewise-linear description of a probability density function
BERNOULLI	Bernoulli distribution
DISCRETE_CDF DISCRETE_PDF	Tabular description of the probability of discrete values

The VHDL-AMS implementation details are online available [4].

3 Method

Methods to create random numbers are in general based on a (0,1) uniform distributed values.

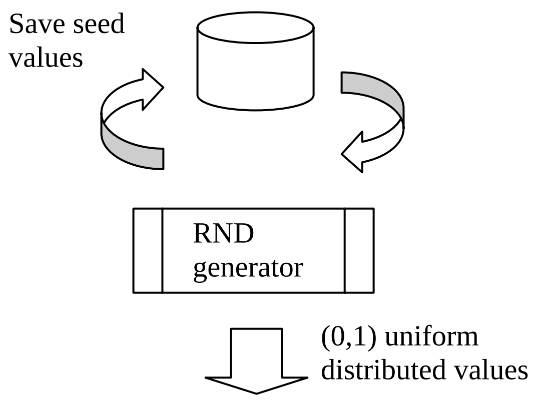


Fig. 1. (0,1) uniform random number generator

Widely used methods to generate random number with a given distribution are the inverse transformation approach based on the cumulative distribution function and its modifications for truncated distributions. The Box-Muller algorithm can be applied for normal distributed numbers [5].Thus, the main problem during parameter initialization for Monte Carlo Simulation is to generate independent (0,1) distributed values. [2] describes the requirements to a built-in random number generator provided by a tool.

The basic idea of a tool-independent random number generator is shown in Fig. 1. The seed values that are needed to generate a sequence of random numbers are immediately saved in a file.

With the help of global parameters it is possible to switch between nominal and statistical analysis either w.r.t. parts of a description or the entire simulation task.

4 Realization with Modelica

Using Modelica the idea of a tool independent random number generation is realized in the following way. As an example the uniform distribution is used which produces uniformly distributed values within the interval (nominal – tolerance*nominal, nominal + tolerance* nominal). For better reading some details compared to the final solution are simplified.

4.1 Randomly changed parameters

To supply a parameter (or a constant) with randomly generated values it is necessary to specify random distribution in the Modelica source code. Instead of

```
parameter Real p = nominal;
```

which specifies a fixed parameter, the specification of the uniform distribution function call is:

```
parameter Real p = uniform(nominal,
                           tolerance);
```

4.2 Random number generation

The Modelica function uniform is an interface to a C function. It is defined like this:

```
function uniform
  input Real Mean;
  input Real Tol;
  output Real random_value;
external "C" uniform(Mean, Tol,
                     random_value);
end uniform;
```

Within the C function the randomly distributed values have to be calculated. An example is the following function:

```
void uniform (double M, double Tol,
              double *aus)
{ double xMin = M * (1.0 - Tol);
  double xMax = M * (1.0 + Tol);
  if (xMin > xMax)
  { xMax = xMin;
    xMin = M * (1.0 + Tol);
  }
  *aus = xMin + (xMax - xMin)*RND();
}
```

The random function is a (0,1) uniformly distributed random value generator for instance according to Schrages method [8]:

```
double RND()
{ FILE   *read_fp, *write_fp;
  long   seed = 2,   M = 2147483647;
  long   A = 16807, Q = 127773;
  long   R = 2836,   k;
  double F = 1.0/ M;

  read_fp = fopen ("seed.dat","r");
  fscanf (read_fp, "%ld",&seed);
  fclose(read_fp);

  assert( seed != 0 );
  k = seed / Q;
  seed = (seed - k * Q) * A - k * R;
  if ( seed < 0 ) seed += M;

  write_fp = fopen ("seed.dat","w");
  fprintf (write_fp, "%ld", seed);
  fclose(write_fp);

  return seed * F;
}
```

By access to the file “seed.dat” , which has a fixed name, the seed value is saved between two calls of the random function.

In the final solution a global change of the seed file name is possible. In case of a nominal analysis the final function uniform would deliver the Mean value. A more convenient way would be to provide the random number generator RND by a Modelica function. This would allow to formulate the random distribution functions using Modelica language constructs only. This approach could not be realized in the used tool environment. From the language point of view it must be possible that a Modelica function called with the same arguments may deliver different results. For this reason, for instance VHDL(-AMS) distinguishes between pure and impure functions.

Furthermore, the RND function above could be replaced by the random number generator incorporated in a Modelica simulator by a tool provider. In this way the file access to seed.dat can be avoided.

4.3 Application

After having specified the parameter to be changed in the Modelica source code, the Modelica function with the foreign function interface to the C domain, and the C function “random”, the following steps are necessary:

A file “seed.dat” has to be generated, which contains an integer starting number for the sequence of random values. If a sequence shall be repeated, the same seed number must be chosen.

Then the model under investigation (which contains the parameter specification mentioned above) has to be simulated by a Modelica simulator repeatedly. The number of repetitions depends on the wanted number of trials for the Monte Carlo simulation. After each single simulation the interesting results must be saved. The results can be visualized or used in posteriori calculations.

4.4 Remarks

The method allows easily to define both correlated and dependent random values of parameters. A simple example might explain the procedure:

```
parameter Real p1 = uniform(1, 0.1);
parameter Real p2 = uniform(p1, 0.01);
```

If the same sequence of randomly generated values is desired (e.g. to investigate a special effect) the same seed number and the same seed file name have to be used at the beginning.

5 Example

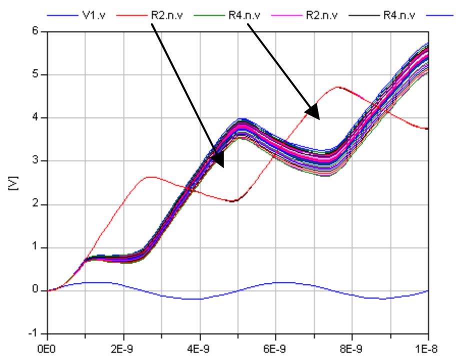


Fig. 2. Monte-Carlo-Plot for variables

In the DifferenceAmplifier of Modelica.Electrical. Analog.Examples [9]. the resistance R of the resistor R2 is randomly generated by the following formulation:

```
...
Basic.Resistor R1(R=0.0001);
Basic.Resistor R2(R=uniform(100,0.05));
Basic.Resistor R3(R=0.0001;
...
```

Repeated simulations using Dymola show that R2.n.v (the thick line pencil) is sensitive with respect to R2.R. The voltage R4.n.v (thin line) is not sensitive to that parameter. Basing on the Monte Carlo results further calculations (density distribution …) are possible.

Furthermore, the randomly chosen parameter values can also be visualized or used for further calculations. The following figure shows the above specified parameter R2.R which is uniformly distributed in the interval (95, 105) (=100 – 100 * 5%, 100 + 100 * 5%).

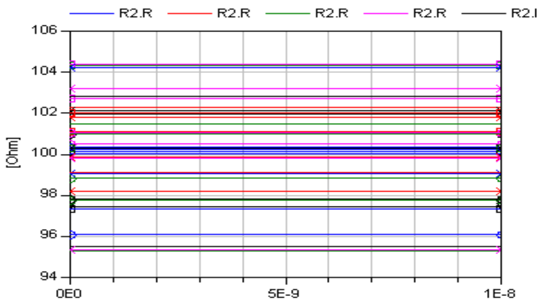


Fig. 3. Randomly chosen parameter R2.R

6 Discussion

The proposed approach realizes a simple Monte Carlo simulation based on behavioral descriptions in Modelica. Beyond the focus of this paper is the usage of the results of the Monte Carlo simulation for other purposes. For example the data could be used to create Response Surface Models. This would require to save the randomly generated parameters of any simulation run. Also improved techniques to create the random numbers and reduce the simulation effort could be applied. For instance possibilities of so-called importance sampling [6] could be applied using user defined functions.

The Monte-Carlo-Simulation is also possible using the Dymola Monte-Carlo feature. The advantage of the suggested way is:

- It is a more general, tool independent approach.
- The user is free to define its own distribution based on the RND function.
- Correlations can be defined easily.
- For documentation purposes the distribution specification is part of the model files.

The approach in [7] is also simulator independent, but is uses a (firm-)specific nested toolkit. Our way is defined only using the Modelica language. Whether a language construct like ours is used in [7] is not documented.

7 Conclusions

An approach to handle statistical analysis problems within Modelica is presented. It is based on the SAE J 2748 standard. The current version allows Monte Carol simulations if the used simulation engine supports multiple runs in a simple way. If the approach is accepted it could also be the basis of efficient implementation in Modelica simulators. In this case the generation of the sequences of (0,1) distributed uniform random numbers must be supported without file access.

The applicability of the approach is demonstrated with the help of a simple example from the existing Modelica standard library. Only existing tool and language features are used. This and the orientation to the SAE standard are the main advantages of the approach compared to [7].

References

[1] O'Connor, P.D.: *Practical Reliability Engineering.* John Wiley & Sons, 2003 (5th ed.)

[2] J2748, *VHDL-AMS Statistical Analysis Packages*, The SAE Electronic Design Automation Standards Committee, Troy, MI, October 2006

[3] Christen, E.; Bedrosian, D.; Haase, J.: *Statistical Modeling with VHDL-AMS.* Proc. Forum on Specification and Design Laguages FDL '07, Barcelona, September 18-20, 2007.

[4] http://links.sae.org/j2748
http://fat-ak30.eas.iis.fraunhofer.de

[5] Saucier, R. *Computer Generation of Statistical Distributions.* US Army Research Lab ARL-TR-2168, available at http://ftp.arl.mil/random/random.pdf

[6] Robert, C. P.; Casella, G.: Monte Carlo Statistical Methods. Springer, 2004 (2nd ed.)

[7] Batteh, J.; Tiller, M.; Goodman, A.: Monte Carlo Simulations for Evaluating Engine NVH Robustness. 4th International Modelica Conference, Hamburg, March 7-8, 2005, 385-392

[8] www.physics.rutgers.edu/grad/509/random.pdf

[9] www.modelica.org/libraries/Modelica

Comparisons of Different Modelica-Based Simulators Using Benchmark Tasks

Olaf Enge-Rosenblatt[1)], Christoph Clauß[1)], Peter Schwarz[1)], Felix Breitenecker[2)], Christoph Nytsch-Geusen[3)]

1) Fraunhofer Institute for Integrated Circuits, Branch Lab Design Automation,
Zeunerstraße 38, 01069 Dresden, Germany
olaf.enge@eas.iis.fraunhofer.de
2) Vienna University of Technology, Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria
3) Fraunhofer Institute for Computer Architecture and Software Technology,
Kekuléstraße 7, 12489 Berlin, Germany

Abstract

A benchmark library is presented which collects models for testing and comparing different analog and hybrid simulators as well as their numerical simulation algorithms. Many of these models are described with Modelica and simulated with Dymola and the Modelica-related simulator Mosilab. But VHDL-AMS descriptions are also used to compare simulation results of Modelica simulators with those of other types of simulators. The motivation of the selection of benchmark problems, the modeling and documentation "style guide", and some small examples from electronics and mechanics are described.

1 Motivation

The development of new simulators and model libraries has to be accompanied by intensive simulations of test examples and their comparison. The first reason for collecting a new benchmark library was the development of a Modelica-based simulator Mosilab [1] and accompanying test examples to ensure the Mosilab functionality. But, there are some other objectives:

- comparison of Mosilab with commercial Modelica simulators: Dymola, SimulationX;
- potential extension to comparisons with other analog simulators (e.g. VHDL-AMS, Verilog-AMS, SystemC-AMS);
- getting experiences with the numerical properties of the implemented solvers and their robustness (e.g., influenced by simulator control parameters);
- testing extreme cases (e.g., depending on the number of variables and equations as well as numerical parameter values);
- collecting models with a special focus on systems with variable structure;
- preparation of regression tests;
- and, last but not least, pedagogical aspects: for use in lectures and tutorials.

Therefore, the construction or selection of benchmark models has to fulfill many criteria. The ARGESIM comparisons ([2], [3]), published in the journal Simulation News Europe (SNE) and via http://www.argesim.org/ , have a similar goal. They are considered here from a common point of view.

Further suggestions are expected from benchmarks in other disciplines ([7], [8]) or with a general methodological background ([9]).

2 Types of simulation problems

The benchmark models are selected with respect to the following tasks:

- simple tests of keywords and other language constructs (especially for compiler tests and version checking in the new Mosilab simulator),
- simple but non-trivial electric circuits (from RLC circuits up to transformers and rectifiers),
- testing typical numerical simulation problems (e.g. stiff differential equations, discontinuities, simulation of ideal oscillators)
- more complicated transistor models which lead in many cases to numerical simulation problems in simulators which are not specialized for electronic applications,
- test of advantageous description means (e.g. object oriented approaches)

- erroneous models (e.g. parallel ideal voltage sources) to check the simulator's behavior in error cases
- inclusion of some "classical", mostly non-electrical ARGESIM comparisons in new or updated form (until now: C1, C3, C5, C7; in preparation: C11, C12),
- testing the capability of simulating systems with variable structures (also called "structural variability" or "model structure dynamics", see [3], [4], [5], [6]): rectifiers with ideal diodes, voltage duplexers with two ideal diodes, constrained pendulum C7, string pendulum,
- modeling with embedded statecharts (as a potential extension of the Modelica language), especially for the Mosilab capabilities of handling variable structures.

1	**Introduction**	**5**
2	**Small examples, taken from Modelica**	**6**
2.1	**Damped RLC oscillator (rlc_osci)**	**6**
2.2	**Transformer (trafo)**	**8**
2.3	**Ideal diode (ideal_diode)**	**12**
2.4	**MOS inverter (mos_inv)**	**15**
2.5	**MOS oscillator (mos_osci)**	**18**
2.6	**Switched capacitor integrator (sc_integrator)**	**23**
2.7	**Test of integrator's error (integr_error)**	**27**
2.8	**Voltage duplexer (Vduplexer)**	**30**
2.9	**Graetz rectifier with resistive load (ideal_graetz)**	**34**
2.10	**Graetz rectifier with RC load (ideal_graetz2)**	**38**
3	**ARGESIM examples**	**43**
3.1	**Comparison 1 (comparison_1)**	**43**
3.2	**Comparison 3 (comparison_3)**	**48**
3.3	**Comparison 5 (comparison_5)**	**55**
3.4	**Comparison 7 (comparison_7)**	**61**
4	**Examples with structural variability**	**63**
4.1	**Ideal diode using events (ideal_diode_event):**	**63**
4.2	**Ideal diode using statecharts**	**64**
4.2.1	**1st realisation (ideal_diodeSC)**	**64**
4.2.2	**2nd realisation (ideal_diodeSCcon)**	**67**
4.2.3	**3rd realisation (ideal_diodeSCdyn)**	**69**
4.2.4	**4th realisation (ideal_diodeSCvN)**	**71**
4.3	**Voltage duplexer using statecharts**	**75**
4.3.1	**1st realisation (VduplexerSC)**	**75**
4.3.2	**2nd realisation (VduplexerSCcon)**	**78**
4.3.3	**3rd realisation (VduplexerSCdyn)**	**81**
4.4	**Comparison 7 using statecharts**	**84**
4.4.1	**1st realisation (comparison_7SC)**	**84**
4.4.2	**2nd realisation (comparison_7SCdyn)**	**86**
4.5	**String pendulum with free motion (fadenpendel)**	**88**
4.6	**String pendulum with free motion using statecharts**	**96**
4.6.1	**Realisation fadenpendelSC**	**96**
4.6.2	**Realisation fadenpendelSCaF**	**102**
4.6.3	**Realisation fadenpendelSC1**	**103**
4.6.4	**Realisation fadenpendelSC3a**	**103**
4.6.5	**Realisation fadenpendelSCdyn**	**106**
4.7	**Switched capacitor integrator in QV formulation**	**112**
4.7.1	**Realisation without statecharts (sc_integrator_QV)**	**112**
4.7.2	**Realisation with statecharts (sc_integrator_QVSC)**	**116**

Table 1: Benchmark library

ARGESIM continues the comparisons by benchmarks with extended information and prepares special benchmarks with emphasis on various modeling approaches. In 2008, benchmarks for *hybrid* modeling and simulation will be published, addressing different modeling techniques for four or five systems (constrained pendulum, rotating pendulum, heat diffusion with different regimes, rotor dynamics).

The content of the actually implemented benchmark library is summarized in Table 1. It consists of three main sections. In the first section, some electrical examples are collected. The second section deals with a selection of the ARGESIM benchmarks, which are mainly published in the journal Simulation News Europe (SNE). The third section collects examples which are characterized by a variation of the model structure. Such systems lead to different sets of differential-algebraic equations and the need of exchanges between them from time to time during the simulation process ([1], [4], [5], [10]).

3 Documentation

Each test example is documented in the same manner:

- short description of the problem and the reason for selecting this model,
- graphical description (schematic/sketch),
- definition of relevant physical quantities and dimensions,
- interface description (e.g., type of signals and quantities),
- textual input description in the Modelica language,
- applied simulator control parameters,
- graphical simulation results and some additional textual information,
- discussion of results (e.g., accuracy, runtime behavior) and detected problems.

If the models should be used for regression tests, further regimentations are necessary.

4 Examples

In this section, some interesting benchmark tasks are collected and discussed shortly. All examples are characterized by variable structure because serious numerical problems consist yet in very small systems.

4.1 Electric example

The electric example shall illustrate the application of different models of a diode component. For this purpose, the diode is used within two different well-known set-ups: a one-way rectifier with an ohmic load (shown in Fig. 1) and a Graetz rectifier with an ohmic-capacitive load (depicted in Fig. 2).

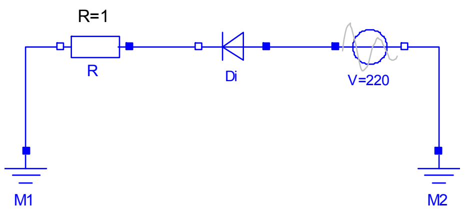


Figure 1: One-way rectifier with ohmic load

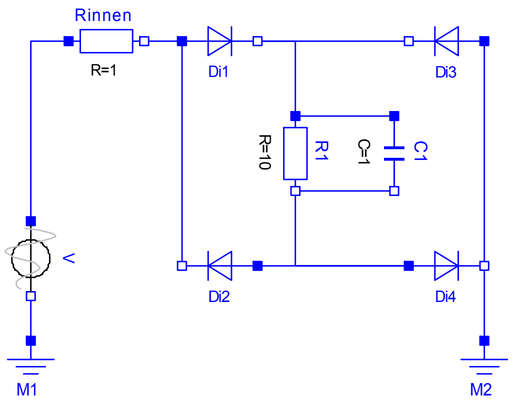


Figure 2: Graetz rectifier with ohmic-capacitive load

First, the piecewise-linear (PWL) diode model of the Modelica Standard Library is used. The relevant source code is shown in Table 2. This model implements the behavior of an idealized switching diode consisting of a piecewise-linear voltage-current characteristic. A so-called auxiliary variable is used which implements a parametric representation of the length of both straight lines [11], [12], [14].

```
model IdealDiode
  extends OnePort;
  parameter Real Ron= 1.E-5,
  parameter Real Goff= 1.E-5;
  Boolean off(start=true);
  Real s;
equation
  off = s < 0;
  v = s*(if off then 1 else Ron);
  i = s*(if off then Goff else 1);
end IdealDiode;
```

Table 2: Source code of diode using auxiliary variable

Second, an ideal diode model was implemented:

- The voltage in flow direction is zero (conducting state).
- The current in the blocking direction is zero (cut-off state).

Conditional equations are used for voltage and current always forcing at least one of them to zero. The source code is shown in Table 3. This implementation requires an event handling by the simulator.

```
model IdealDiodeEvent
  extends OnePort;
  Boolean blocking(start=true);
equation
  blocking = if pre(blocking)
               then v<0 else i<0;
  if blocking then
    i = 0;
  else
    v = 0;
  end if;
end IdealDiodeEvent;
```

Table 3: Source code of diode using conditional equation

With all simulators under test, very similar simulation results were received for the one way rectifier. Exemplarily, Fig. 3 shows simulation results for some voltages calculated by Mosilab using the PWL diode model. The results of the other simulators are the same. This statement also holds for the ideal diode model no matter which simulator is tested. Of course, the current of the blocking diode is now exactly equal to zero or, vice versa, the voltage of the conducting diode now vanishes completely.

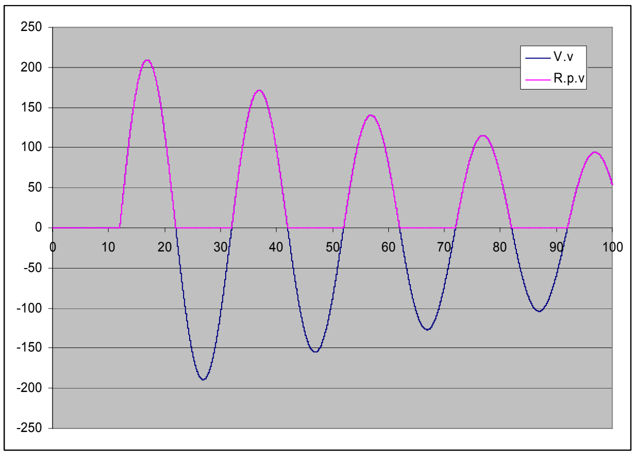


Figure 3: Simulation result from Mosilab for a one-way rectifier

In contrast, the electric circuit of the Graetz rectifier can only be simulated using the PWL diode model (Table 2). The property of such a circuit, that two diodes of the four must unconditionally be closed (or opened) at the same time, is the reason for this fact. This conclusion is valid for Dymola as well as for Mosilab. To handle a circuit with a Graetz rectifier using ideal diodes, it is necessary to qualify a simulator with the feature of finding a valid new model structure from the complete set of structures at each switching point in time.

Some simulation results for the Graetz rectifier using the PWL diode model are shown in Fig. 4 and Fig. 5. Fig. 4 depicts some voltages while the corresponding currents are shown in Fig. 5.

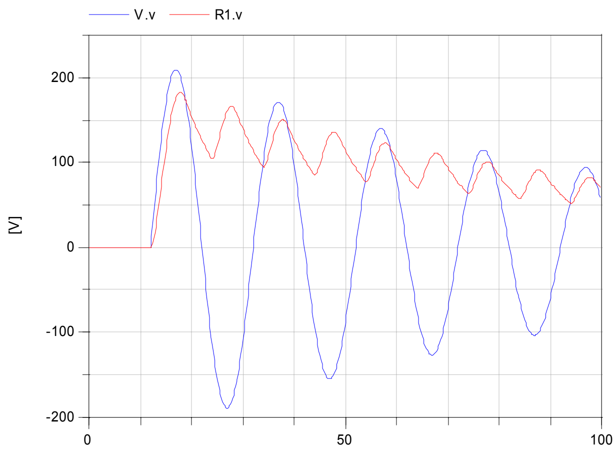


Figure 4: Voltages of the Graetz rectifier circuit

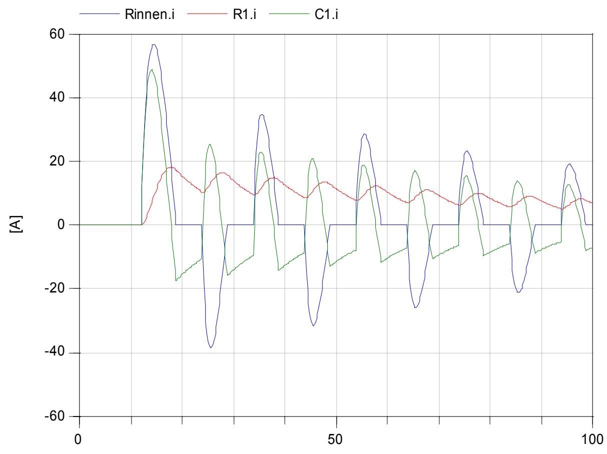


Figure 5: Currents of the Graetz rectifier circuit

4.2 Two-state model

The two-state model considered here is the ARGESIM comparison C5 which is of high interest regarding to the numerical behavior of each simulator. The problem consists of the two simple differential equations:

$$\begin{aligned}\dot{y}_1 &= c_1(y_2 + c_2 - y_1),\\ \dot{y}_2 &= c_3(c_4 - y_2).\end{aligned} \tag{1}$$

In Equ. (1), the parameters c1 and c3 are fixed while c2 and c4 have different values depending on the actual state of the system. State 1 is valid as long as $y_1 < 5.8$. Reaching this value, the system state is changed to state 2 which, then, is valid until y_1 goes below 2.5. All parameters and initial conditions were chosen in a very sophisticated manner. This way, the numerical accuracy of the simulators under test can be investigated by looking at the switching points in time, especially at the last one (denoted with `t5`) which appears generally at about 5 Seconds.

The Dymola result computed by the DASSL solver using the highest possible numerical accuracy (tolerance is set to `1E-12`) shall be taken as reference for other simulators. The last switching point in time appears at `t5=4.999999646`. Other solvers, like Runge-Kutta methods, are less suitable for such kind of a simulation task.

With Mosilab, the switching point in time is found very well if using the IDA solver which is very similar to the DASSL method. With an absolute tolerance of `1E-14` and a relative tolerance of `1E-10`, the switching point in time can be determined to `t5=4.999999645`. Surely, this is a very good result. But using lower tolerances or using one of the other numerical solvers of the Mosilab simulator leads to more inexact results.

Exemplarily, Fig. 6 shows the time behavior of y_1 using Dymola with the DASSL method as mentioned above.

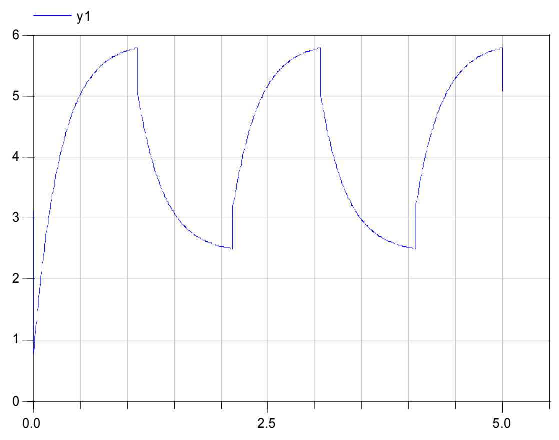


Figure 6: Time behavior of state variable y_1

4.3 String pendulum

A string pendulum is shown in Fig. 7. A point mass is able to perform circular or free (downfall) movements – so-called phases (see Fig. 7A). The circular movement is characterized by a stretched (but non-widening) thread, i.e. the mass has the maximal possible distance to the fixing point. In contrast, the mass has a smaller distance and the thread is folded during the free movement. This is an extension of the well-known mathematical pendulum with small elongations and without the downfall phase.

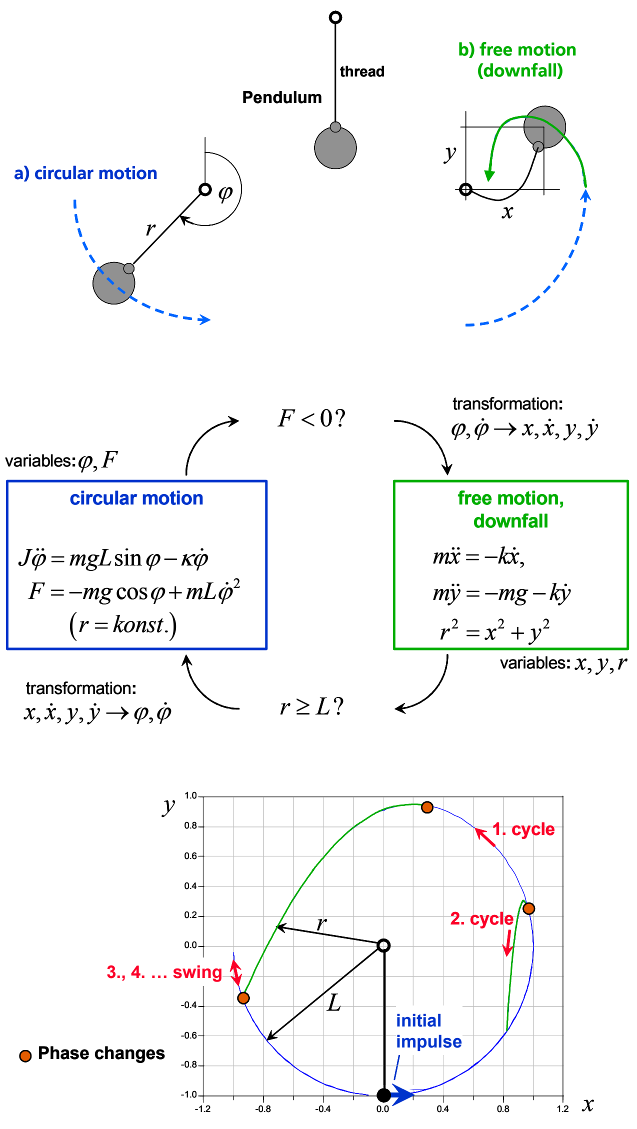

Pendulum
thread
a) circular motion
r
φ
b) free motion (downfall)
y
x
$F<0?$
transformation: $\varphi, \dot{\varphi} \rightarrow x, \dot{x}, y, \dot{y}$
variables: φ, F
circular motion
$J\ddot{\varphi} = mgL\sin\varphi - \kappa\dot{\varphi}$
$F = -mg\cos\varphi + mL\dot{\varphi}^2$
$(r = konst.)$
free motion, downfall
$m\ddot{x} = -k\dot{x},$
$m\ddot{y} = -mg - k\dot{y}$
$r^2 = x^2 + y^2$
variables: x, y, r
transformation: $x, \dot{x}, y, \dot{y} \rightarrow \varphi, \dot{\varphi}$
$r \geq L?$
y
1. cycle
2. cycle
3., 4. ... swing
r
L
initial impulse
● Phase changes
x

Fig. 7: String pendulum

A) Geometrical configuration

B) Mathematical problem formulation of both phases

C) Simulation results

The simplest description of the circular motion uses polar coordinates; the downfall motion may be described with Cartesian coordinates. There are two differential-algebraic equation systems with two and three variables, respectively, describing both phases (see Fig. 7B). In phase 1 (circular movement), the stretching force F in the thread is greater zero. In phase 2 (free movement), the distance r between point mass and fixing point is less than the length L of the thread. The “indicator functions” ($F < 0$ and $r \geq L$) are used to detect the points in time of a necessary switching between the phases.

A large initial impulse results in a sequence of circular and free movements. This is illustrated in Fig. 7C. The point mass performs two “circles” followed by some swinging movements. The time behavior of the mass position and the corresponding force F are shown in Fig. 8 and Fig. 9, respectively.

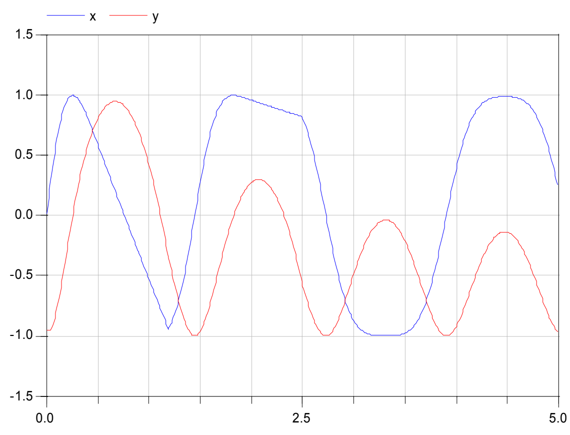


Figure 8: Pendulum’s mass position (x and y)

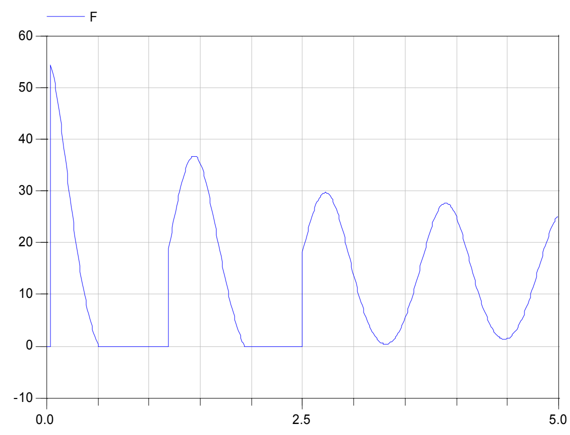


Figure 9: Force F in the thread

This description is closely related to a statechart description, which can be formulated with the State Graph Library and simulated using Dymola or with an extended Modelica description for the Mosilab simulator.

The model implementation depends strongly on the applied simulator. In Dymola, it is necessary to use the same number of equations in both phases. Therefore, some dummy equations have to be introduced. In simulators like AnyLogic or Mosilab, different numbers of equations are allowed in various model states.

5 Summary and outlook

This collection of benchmark problems is under development in connection with the Mosilab development [1] and has its roots in a Fraunhofer-internal research project GENSIM. Some parts of these examples will be published in connection with new Modelica-oriented projects. Problems of more general interest will be prepared for the widely-distributed ARGESIM comparisons.

The collection of benchmarks presented here has proved as a powerful tool for testing the numerical behavior and the modeling limits of different simulators. In this paper, only some examples of general interest are described.

The collection is under continuous development. The pool of tasks as well as the tested simulators and the different modeling languages have to be extended.

It is intended to include parts of the benchmark examples into the regression test library ModelicaTest, which is used by the Modelica Design Group for developing the Modelica Standard Library.

References

[1] Nytsch-Geusen C et al. Mosilab: Development of a Modelica based generic simulation tool supporting model structural dynamics. 4th Int. Modelica Conf., Hamburg, Germany, March 2005.

[2] Breitenecker F et al. Education in modeling and simulation using ARGESIM comparisons/benchmarks with physical modeling. MATHMOD (5th IMACS Symposium on Mathematical Modelling), Vienna, Austria, February 2006.

[3] Pawlik S et al. A classification of modeling and simulation approaches based on the ARGESIM benchmarks. EUROSIM, Ljubljana, Slovenia, September 2007.

[4] Schwarz P. Simulation of systems with dynamically varying model structure. MATHMOD (5th IMACS Symposium on Mathematical Modelling), Vienna, Austria, February 2006.

[5] Breitenecker F et al. Structure of simulators for hybrid systems – general development and introduction of a concept of external and internal state events. EUROSIM, Ljubljana, Slovenia, September 2007.

[6] Nytsch-Geusen C. The use of the UML within the modelling process of Modelica-models. Proc. 1st Int. Workshop EOOLT, Berlin, Germany, August 2007.

[7] Benchmarks for water supply modelling, simulation and optimisation. http://www.eng.dmu.ac.uk/~wss/

[8] Iwnicki S (Ed.). The Manchester Benchmarks for Rail Vehicle Simulation (Supplement Vehicle System Dynamics (SVD)). Swets & Zeitlinger, July 1999.

[9] Gelfert A. Simulating many-body models in physics: rigorous results, 'benchmarks', and cross-model justification. LSE/CPNSS Conf. on Models and Simulations, Paris, France, June 2006.

[10] Enge-Rosenblatt O et al. Numerical Simulation of Continuous Systems with Structural Dynamics. EUROSIM, Ljubljana, Slovenia, September 2007.

[11] Mattsson SE, Otter M, Elmqvist H. Modelica Hybrid Modeling and Efficient Simulation. 38th IEEE Conference on Decision and Control, CDC'99, Phoenix, Arizona, USA, December 1999.

[12] Otter M, Elmqvist H, Mattsson SE. Objektorientierte Modellierung physikalischer Systeme, Teil 8. at – Automatisierungstechnik 47(1999)9, A29-A32.

[13] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley, New York, 2004.

[14] Clauß C, Haase J, Kurth G, Schwarz P. Extended admittance description of nonlinear n-poles. Archiv für Elektronik und Übertragungstechnik / Int. J. Electronics and Communications, Vol. 40, pp. 91-97, 1995.

Modelica Wind Turbine Models with Structural Changes Related to Different Operating Modes

Olaf Enge-Rosenblatt, Peter Schneider
Fraunhofer Institute Integrated Circuits, Branch Lab Design Automation,
Zeunerstraße 38, 01069 Dresden, Germany
olaf.enge@eas.iis.fraunhofer.de

Abstract

Investigation of large technical systems by simulation of long time periods requires effective methods. One possibility to handle such problems is the implementation of simulation models which use suitably simplified descriptions of the real behaviour of technical systems. In some cases, however, operating modes with highly dynamic processes have to be investigated. These processes may occur suddenly within long time periods of behaviour with none or very low dynamics, which can be considered as static behaviour. In such cases, it would be advantageous to be able to switch from the simplified model mentioned above to a more complex model describing the real behaviour in more detail.

In the paper, four different Modelica models for wind turbines are presented. On the one hand, two static models – the "simple static model" and the "static mechatronic model" – are shown representing two different instances of a simplified behaviour. On the other hand, two dynamic models – the "mechanical model" and the "dynamic mechatronic model" – are presented which describe the dynamic behaviour of a wind turbine in more detail. Furthermore, a method will shortly be proposed to exchange one model with another one at certain points in time (see also [5]). Such structural changes allow the application of that particular model of behaviour which suits the current situation best. Using this method, the simulation of a complex mechatronic system like a wind turbine can very effectively be carried out. Additionally, some simulation results will be given to show the advantage of the method proposed.

1 Introduction

The proportion of renewable energy in industrial countries is growing with increasing speed. The usage of wind turbines plays an important role among these forms of power generation. A wind turbine is a complex mechatronic system consisting of mechanical parts, electrical components, and a very complex control strategy.

Investigation of wind turbines using numerical simulation becomes more and more important. Therefore, design, construction and scheme of operation of the turbine under investigation must be taken into account. The level of detail which is necessary for a special model depends on the questions which are to be answered by the simulation results. On the one hand, we have to distinguish between models of single turbines and whole wind parks. In the paper, model types suitable for both situations will be presented. On the other hand, behavioural models describing only the flow of electrical energy stand in opposition to models which use voltage and current as time-depending electrical quantities. Again, both types of models are introduced here.

Every type of a wind turbine model presented in this paper is suitable for a well determined level of detail. Every model uses a particular set of physical quantities to describe the corresponding physical behaviour. All models are equipped with interfaces that allow a simple exchange of one model with another one at arbitrary points in time. This property makes it possible to investigate a complex mechatronic system like a wind turbine as exact as necessary depending on the current situation of operation simply by using the actually best suiting model of behaviour.

In the following section, the general logical scheme of operation of the construction type of wind turbine considered in this paper is outlined. The four models are presented in section 3. Some simulation results are given in section 4.

2 Scheme of operation

There is a great variety of types of existing wind turbines (see e.g. [6], [7], [8], [14]). All of them have advantages and disadvantages. However, the most widely used type of a wind turbine is equipped with a so-called pitch control and an asynchronous generator ([6], [9]). With such a turbine, the energy harvested from the wind can be influenced by controlling the pitch angle which is the angle of the rotor blade across

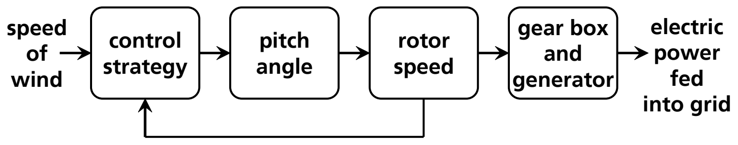


Figure 1: Wind turbine's logical scheme of operation

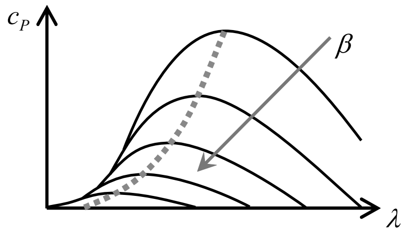


Figure 2: Array of curves of power coefficient

its longitudinal axis. The simplified logical scheme of operation of such a wind turbine is shown in **Fig. 1**.

The controller always tries to harvest as much as possible energy from the actual wind. To this end, the controller uses the actual speed of wind and the actual speed of rotor as input signals to calculate the pitch angle. This angle then again mainly determines the angular momentum acting on the rotor. Hence, the rotor speed and, therefore, the speed of the electric generator, is influenced by the controller. Going into more detail, the controller endeavours to put the point of operation into a maximum of the power coefficient's array of curves, which are exemplarily depicted in **Fig. 2**. In this figure, c_P is the power coefficient, λ stands for the speed ratio between blade's tip and wind ($\lambda = R\omega / v_W$, R – radius of rotor, v_W – speed of wind), and β denotes the pitch angle. In **Fig. 2**, five curves of the whole array for fixed values of β are shown (solid lines). The dotted line depicts an approximation of the connecting curve of the maximum points of all c_P-curves using the pitch angle β as a parameter. Using such an array of curves, the controller chooses a pitch angle which determines the rotor speed in such a way that as much as possible energy can be harvested from the actual wind. A realistic array of c_P-curves – implemented in the models of the next section – was taken from [15].

3 Wind turbine models

In this section, four different models of a wind turbine characterized by a pitch angle and an asynchronous induction generator are presented. All these models use the speed of wind as an input variable. Number and physical quantity of the output variables depend on the particular model. The direction of the wind (and the variation of the direction) is not considered in any turbine model presented here. Hence, investigations of changing wind directions, their measurement, as well as the dynamic behaviour of a turbine when rotating across its vertical axis (i.e. when "turning into the wind") are not included in the models considered in this paper.

The range of applicability of every model depends on its level of detail. The simplest one is called "simple static model". It is suitable for energy flow considerations of whole wind parks. The "mechanical model" allows simple dynamic investigations of the mechanical part of a single turbine. With both models, no interaction between the turbine and the energy grid can be considered. Compared with this, the "static mechatronic model" and the "dynamic mechatronic model" are physical models with a more sophisticated design. They use characteristic quantities of both mechanical and electric domain. Because of the usage of electric quantities like current and voltage of the generator, many interations between turbine and energy grid can be taken into account. Hence, these models are well suitable for investigations of the mutual influence of different turbines within a wind park.

3.1 Simple static model

The "simple static model" is the simplest possible model describing the physical behaviour of a wind turbine. The only input is the actual speed of wind. The output quantity is the electric power which can be harvested from the actual wind under the assumption of an optimal operation of the turbine's controller.

The relation between speed of wind and electric power is shown in **Fig. 3**. It consists of two main areas: the partial load range and the full load range (see e.g. [16]). Within the partial load range, the speed of wind is slower than a value v_{Wnom} which is called the nominal speed of wind. Here, the electric power is a cubic function of the speed of wind. The full load area is the range

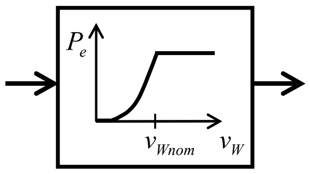


Figure 3: Characteristic curve of the "simple static model"

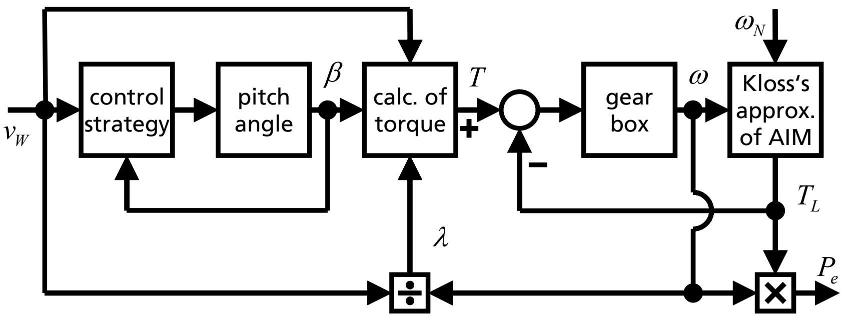


Figure 4: Logical scheme of the "mechanical model"

of wind speeds which are higher than the nominal value. Here, the electric power does not depend on the actual speed of wind. Instead, it is assumed to be constant. Finally, the electric power is set to zero for both very small values and very high values of v_W. Within these ranges, the system is not in operation because of inefficiency and safety, respectively.

The "simple static model" describes a simple relation between speed of wind and electric power without any dynamics. No more characteristic quantities of a turbine are used. Therefore, the model can only be used if all components of the turbine work correctly. Of course, behavioural simulations with this model are really very fast. Hence, the model is suitable for considerations of energy flows with single turbines as well as with whole wind parks (consisting e.g. of 100 or more installations). The determination of bottle necks within the energy grid while assuming typical wind profiles for the park location may be of special interest in this context.

3.2 Mechanical model

The "mechanical model" implements the main properties of the turbine's mechanical subsystem. Like with the "simple static model", the actual speed of wind is used as the only input and the electric power is the output. In the model, some dynamics of mechanical components are included.

The appropriate logical scheme is shown in **Fig. 4**. The pitch angle is governed by the controller according to the maximum power coefficient principle (see **Fig. 2**). Depending on the actual speed ratio λ between the blades' tip and wind, the nominal pitch angle is choosen so that the power coefficient becomes a maximum value (i.e. the point of operation is located on the dotted line in **Fig. 2**). After a change of wind speed, the pitch angle has to be readjusted. This has to be done in consideration of the limited angular velocity and acceleration of the rotor blades. The profile of angular velocity assumed here is a so-called trapezoid profile (see **Fig. 5**, where the angular velocity $\dot{\beta}$ is plotted against time t). It consists of an acceleration region, a range with constant speed and a deceleration region. Using this profile, the pitch angle is changed if necessary. This way, the so-called pitch dynamics is included in the "mechanical model". Then, the actual pitch value influences the driving torque via the array of curves of the so-called torque coefficient. A sketch of this array is shown in **Fig. 6**. In this figure, c_T denotes the torque coefficient, where λ is again the speed ratio and β is the pitch angle. The realistic array of c_T-curves implemented within the "mechanical model" is taken from [15]. The same array is also applied within both mechatronic models (see sections 3.3 and 3.4). Using the actual value of c_T at a time, the driving torque T is calculated according to

$$T = c_T \frac{\rho}{2} \pi R^3 v_W^2 \tag{1}$$

(ρ – air density). After computation of driving torque, the rotor acceleration is determined using the following torque balance

$$\tilde{J}\dot{\omega} = T - T_L - \kappa\omega \tag{2}$$

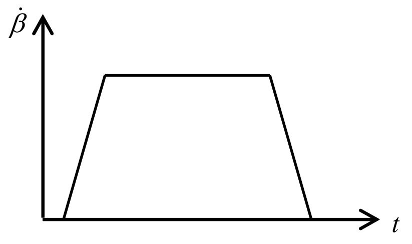


Figure 5: Profile of pitch angle velocity

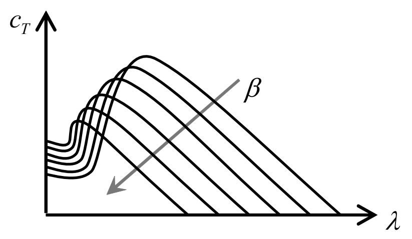


Figure 6: Array of curves of torque coefficient

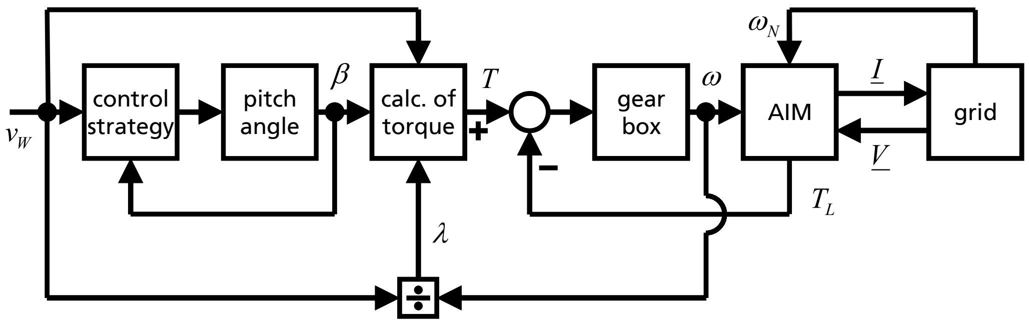


Figure 7: Logical scheme of the "static mechatronic model"

($\tilde{J}$ – rotor's moment of inertia related to gear box ratio, T_L – load torque, κ – damping coefficient). For this purpose, the generator's load torque is calculated by Kloss's approximation for an asynchronous induction machine (see e.g. [13])

$$T_L = \frac{T_B}{s/s_B + s_B/s}, \tag{3}$$

where $s = (\omega_N - \omega)/\omega_N$ denotes the slip (T_B – breakdown torque, s_B – breakdown slip, ω_N – grid's angular frequency). Finally, the electric power P_e fed into the grid (the model's output) is assumed to be equal to the mechanical power (a given efficiency factor may be taken into account).

The "mechanical model" describes the electrical power fed into the grid as a function of the speed of wind. This description includes the main dynamics of the wind turbine's mechanical subsystem and takes into account the correct calculation of the driving torque using the pitch-depending torque coefficient. Therefore, many of the mechanical characteristic quantities are provided for a dynamic simulation by the model. The model is suitable for investigations of the dynamic behaviour of the mechanical part of a single wind turbine if the dynamics of the electrical part is either negligable or not of interest. An example for such investigations is e.g. the problem of finding the optimal time interval for measuring the speed of wind and – corresponding to this question – the optimal strategy for controlling the pitch angle.

3.3 Static mechatronic model

The "static mechatronic model" extends the "mechanical model" mentioned before by an electrical subsystem. Like with both models before, the actual speed of wind is used as an input. But additionally, the voltage of the energy grid is used as input, too. The output is the electrical current fed into the grid. Therefore, the mechatronic models (the static one here and the dynamic one in the next section) implement a fully bidirectional connection between the turbine's electrical subsystem and the energy grid.

The appropriate logical scheme is shown in **Fig. 7**. Most of the mechanical subsystem is realized in the very same way like in the "mechanical model". This concerns the pitch angle adjusting with its dynamics, the calculation of driving torque, and the determination of rotor's acceleration via torque balance. Only Kloss' approximation of an asynchronous induction machine is substituted by an equivalent circuit.

The electrical subsystem of the "static mechatronic model" realizes only its steady state behaviour. Considering only steady states, the phasor description of sinusoidal quantities leads to an adequate mathematical model for the electrical subsystem (see e.g. [2], [12]). An appropriate equivalent circuit for the asynchronous induction generator (see **Fig. 8**) is used. Please note that all underlined symbols in this figure denote phasors ($\underline{V}$ is a voltage phasor, $\underline{I}$ is a phasor of an electric current – both are also used in **Fig. 7**) whereas R, L, ω, and s denote ohmic resistor, inductance, angular frequency, and slip, respectively.

The electrical subsystem is implemented using a special Modelica library for phasor domain-based systems. This library was already presented at the last Modelica conference (see [3]). Hence, details to the phasor description and the special library shall not be given here. In [3], we also pointed out that – with such a model – a so-called quasi-stationary mode can be described under some weak assumptions. With a wind turbine, such an operating mode is characterized by slow dynamics of the mechanical subsystem and a sequence of steady states of the electrical subsystem. See [3] for more details.

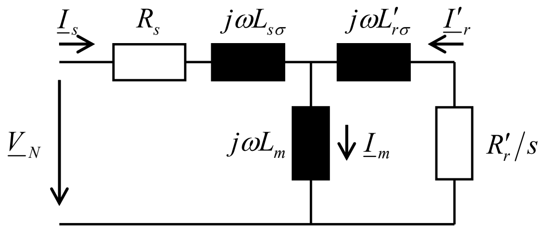


Figure 8: Equivalent circuit of an asynchronous induction machine using phasor description

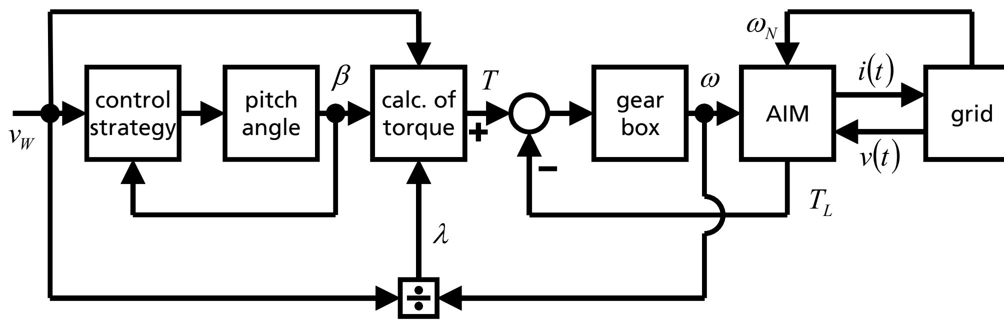


Figure 9: Logical scheme of the "dynamic mechatronic model"

The "static mechatronic model" realizes a complete mechatronic system of a wind turbine consisting of a controlling part, a mechanical part, and an electrical part. Due to the application of phasor domain-based electrical quantities, the high dynamics of the electrical subsystems (usually the 50 Hz or 60 Hz sinusoidal oscillations) do not carry any weight concerning dynamic simulations of the whole system. Hence, this model is well suitable for investigations of the behaviour of many turbines of a wind park, especially for considerations of mutual interactions between the turbines and the grid or between different turbines connected with the same part of the grid.

3.4 Dynamic mechatronic model

The "dynamic mechatronic model" is the most complex one described within this paper. Like with the "static mechatronic model", the actual speed of wind and the voltage of the grid are used as inputs while the output is the electric current fed into the grid. Hence, the model implements a fully bi-directional connection between turbine and grid.

The appropriate logical scheme is shown in **Fig. 9**. The mechanical submodel is completely equal to that of the static mechatronic model. The important difference to this model mentioned above is the implementation of the fully dynamic behaviour of an asynchronous-type generator. Please note that time-depending electrical quantities ($v(t)$, $i(t)$) are used in **Fig. 9** instead of phasors. In the usual case of a three phase grid, such a model of a generator consists of six time-depending electrical currents (three stator currents and three rotor currents) which require, of course, six differential equations to calculate them. One extra (algebraic) equation is necessary to determine the load torque produced electrically (see e.g. [4]). Because of the generator equations and the sinusoidal electrical quantities appearing there, high dynamics is involved in the turbine's model. Hence, a dynamic simulation using such kind of model needs small solver steps. This fact leads to time-consuming simulation experiments.

The "dynamic mechatronic model" realizes a fully dynamic model of the mechatronic system of a wind turbine. Both subsystems (mechanical and electrical) are described by differential-algebraic equations. Merely, the power electronics with its switching effects is neglected. Hence, this model is well suitable for investigations of the behaviour of a single wind turbine taking into account many dynamic effects from mechanical and electrical domain. Especially, the interaction between a wind turbine and the energy grid can be considered in a detailled way with this model. Enormous simulation times because of the high dynamics of many electrical quantities are a disadvantage of this model.

3.5 Model exchange

Investigations of interesting questions concerning wind turbines often require dynamic simulations over very long time periods. To carry out such analysis in a conveniently effective manner, special simulation methods are necessary. The main influence to the dynamic behaviour of a turbine is exerted by the wind. On the one hand, there are long time periods with only few variations of its speed. Within these periods, a simulation model consuming as less as possible calculation time is of interest. On the other hand, there are short time intervals, where the speed of wind is changing very fast. In such critical cases, the compliance of given conditions of operation is very important. Hence, a dynamic simulation with a sufficient level of detail is of essential importance.

To handle the problem of changing demands to the level of detail of a model, the exchange of one submodel with another one at proper points in time is proposed. The points in time of a necessary change from the simple model to the detailled one can e.g. be found by monitoring the acceleration of the wind (i.e. the variation of the speed of wind). If the accerelation value exceeds a well defined border then the model change is necessary. Switching on and off of main consuming devices may also be of interest. Here, the points in time are predetermined. The switching back from the de-

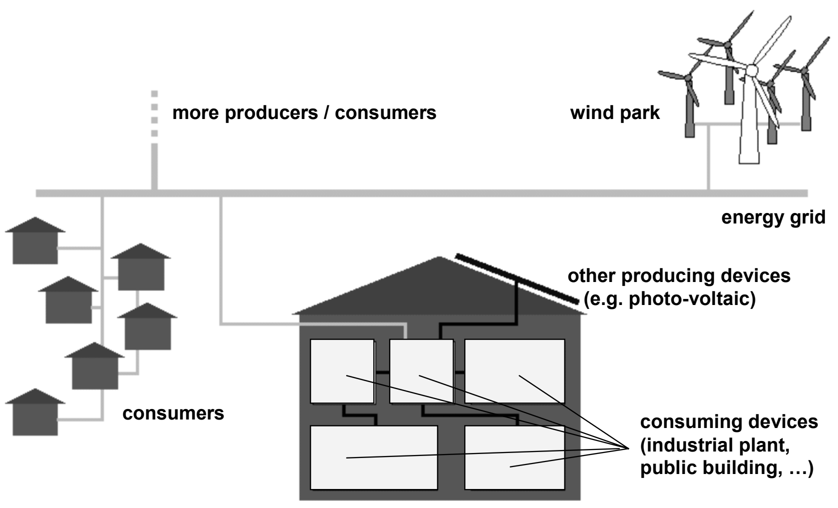

more producers / consumers
wind park
energy grid
other producing devices (e.g. photo-voltaic)
consumers
consuming devices (industrial plant, public building, ...)

Figure 10: Energy grid with wind park and consumers

tailled model to the simple one may be carried out if the dynamics of the complete system is faded away.

Both switching operations – from low level to high level of detail and vice versa – have to be performed taking into account the possibly changing number of differential and algebraic equations. That means that three steps are to be done:

- The dynamic simulation may be terminated at a certain point in time.
- The actual state of the old model has to be transformed into the new model.
- Consistent initial values for the complete set of equations of the new model have to be found.

For more information concerning this way of realisation, please refer to [5].

3.6 Model implementation

The models presented here have been implemented using the Modelica Standard Library, extended by some physical relations and algorithms in order to provide an arbitrary wind profile, to model the whole turbine's control strategy, to handle the pitch angle adjustment, as well as to carry out some approximations concerning the coefficient's arrays of curves (power coefficient, torque coefficient) included in the models. Additionally, a Modelica library for phasor domain-based description (see [3]) is used in case of the "static mechatronic model".

Unfortunately, a real switching between different levels of detail – i.e. an exchange of model parts in such a way that the equations of the "inactive" part at a time are excluded from the equation set of the numeric solver – is not supported by most Modelica simulators until now. For this reason, parts of the following results are achived by a kind of "step-wise" simulation.

4 Simulation results

Considering the four models of wind turbines presented in section 3, the mechatronic models are the most interesting ones. Therefore in this section, some simulation results are shown which were reached using these two models.

Please imagine a little wind park connected to some consumers. A similar (but simplified) scenario is shown in **Fig. 10**. Dynamic simulations of such a complex system using the "dynamic mechatronic model" would require a huge simulation effort. An investigation of the system's behaviour for, say, one year would hardly be possible. The only way to earn some results within a reasonable time effort is to operate with changing submodels. To this end, the "static mechatronic model" and the "dynamic mechatronic model" are alternately applied. Depending on the actual situation, either the static model or the dynamic model is used to describe the complete system.

4.1 Functionality test

First, a functionality test for the two mechatronic models is presented. This this end, a rapid change of speed of wind – a zooming ramp which is nearly a step – is assumed as input signal at time $t = 1\,\mathrm{s}$ (see **Fig. 11**). Such a sudden step is admittedly very unlikely for a real wind turbine. But the functionality test was intentionally performed under extreme conditions.

The step responses of the two wind turbine models are shown in the following figures (**Fig. 12** ... **Fig. 15**). In all these figures, the prefix "smm" (corresponding to a solid line) means that the result originate from the "static mechatronic model" while the string "dmm" (corresponding to a dashed line) indicates the "dynam-

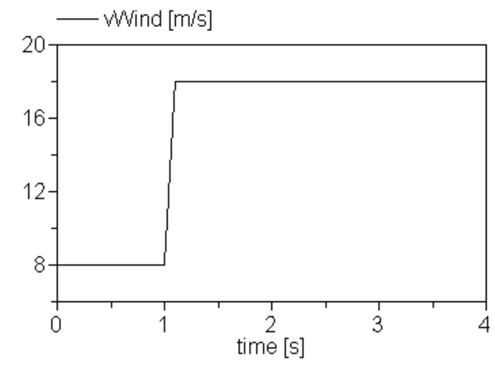


Figure 11: Sharp change of wind speed

ic mechatronic model". In **Fig. 12**, the time progress of the pitch angles is depicted. Both angles are very fast justified by the controller from 0° to 20°. The small difference of the ramp's increase is caused by the fact that the controller uses both the speed of wind and the rotor speed as input signals. The rotor speed is shown in **Fig. 13** for both models. Here, the different behaviour of both models is illustrated. The static model calculates significantly higher values than the dynamic model. This is valid in the time interval of the changing pitch angle as well as in the time of constant rotor speed. The same behaviour is demonstrated in **Fig. 14**.

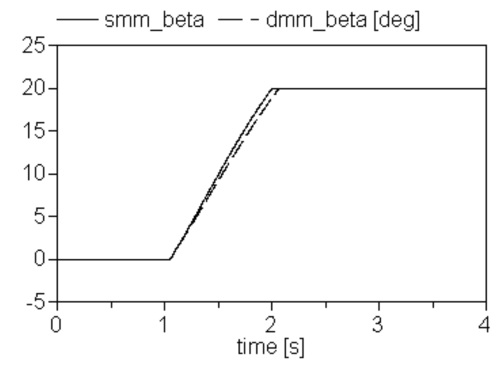


Figure 12: Pitch angle

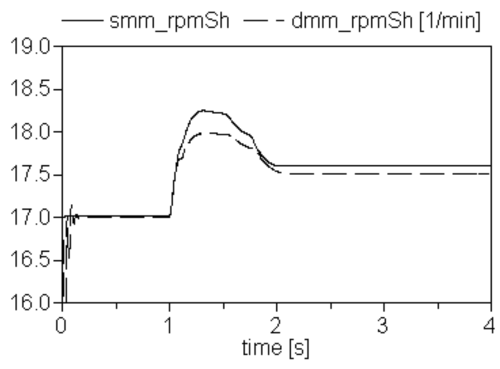


Figure 13: Turbine's rotor speed

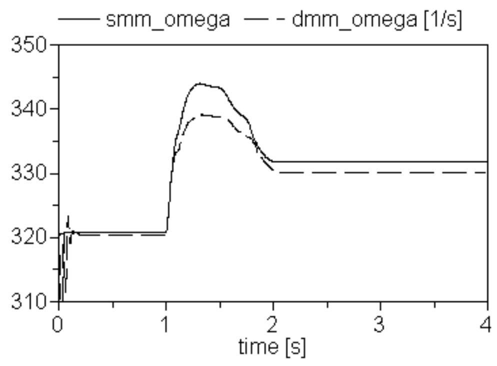


Figure 14: Generator's angular velocity

This figure contains the curves of the angular velocity of the generator which is connected to the rotor via an ideal gear with a speed ratio of 1:180. **Fig. 13** and **Fig. 14** show after a very close look that the dynamic model needs less more time to react to the sharp change of wind speed. That means on the other hand that the static model does not yield correct results in such cases. Finally, the same effect is shown in **Fig. 15** which depicts the corresponding time history of the electric power produced by the turbine and fed into the grid. Though in this diagram, the difference between both results is not such significant like with the turbine's rotor speed of with the generator's angular velocity. However, the dynamic model needs less more time to reach the area of constant electric power.

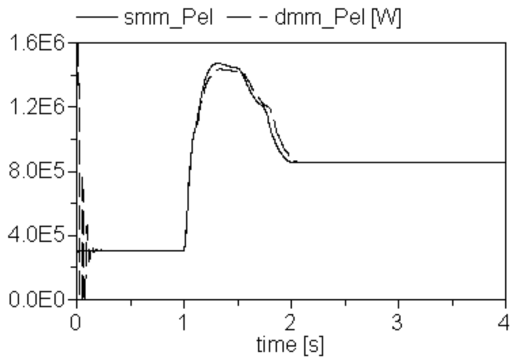


Figure 15: Electric power

4.2 Long-term simulation

In this section, results of a long-term simulation are given. Using such investigations, on the one hand the suitability of different models and on the other hand the rate of effectiveness of model exchange can be determined. As already pointed out in section 3.6, a "step-wise" simulation method is necessarily applied here because of the inability of most Modelica simulators to handle models with exchanging parts correctly. In this context, "step-wise" simulation method means that the three step mentioned in section 3.5 were carried out not driven by the simulator but forced by the user. In other words, different tasks had to be performed where the model exchanges were done by transforming the actual state into the new model and starting a further simulation task. Possibly, new developments (see e.g. [1], [5], [10], [11]) will improve the situation in the near future.

The simulation period shall have a length of 1200 s. The used wind profile along the complete time interval is a realistic profile near to wind data meassured in reality. The shape of the wind profile is depicted in **Fig. 16**. It has three regions with relatively low wind speeds between 5 m/s and 10 m/s (time intervals: 0-30 s, 60-80 s, 100-120 s). In contrast, there are two regions with high or middle speeds of wind of about 20 m/s and 15 m/s, respectively (time intervals: 33-55 s, 80-100 s).

First, the complete task was computed using the "static mechatronic model". On a nowadays standard PC (Intel T2400 dual-core CPU with 1.8 GHz each), the simulation took only 1.8 s. But the results can only be understood as a sequence of steady states (see [3]). In highly dynamic situations, the numeric error of such a calculation method may not be neglected. But if performing the complete task using the "dynamic mechatronic model", it takes much more time to finish. On the same PC, a time effort of 43.5 s was needed.

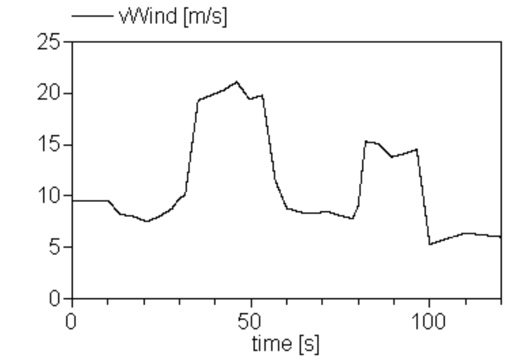


Figure 16: Realistic shape of wind speed

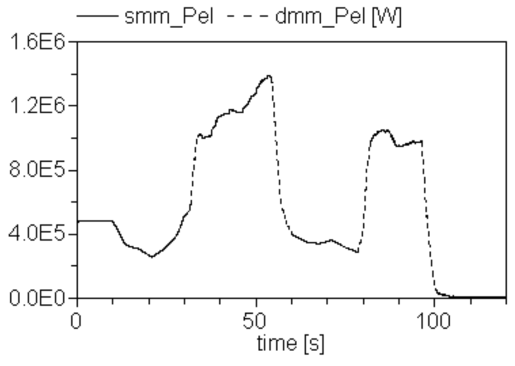


Figure 17: Electric power with switching models

A compromising solution is shown in **Fig. 17**. The five regions mentioned above are investigated using the "static mechatronic model" because the wind shows only low dynamics there. The corresponding time history of the turbine's electric power is indicated by solid lines. However if monitoring high wind dynamics, the "dynamic mechatronic model" is used. The corresponding power curves are indicated by dashed lines. The dynamic model is used during the four short time intervals between the five steadied regions. This way, a model exchange is needed at eight points in time. These are marked in **Fig. 17** by changing line types.

5 Summary

A wind turbine is a complex mechatronic system consisting of mechanical parts, electrical components, and a very complex control strategy. The article deals with a widely used type of wind turbines which is equipped with a so-called pitch control and an asynchronous generator. Four different models for describing the static and/or dynamic behaviour of such a wind turbine are presented. Every model implements a well determined level of detail and uses a particular set of physical quantities to describe the corresponding physical behaviour. All models are equipped with interfaces that allow model exchanges. This property makes it possible to investigate a complex mechatronic system like a wind turbine as exact as necessary depending on the current situation of operation simply by using the actually best suiting model of behaviour.

In the paper, two static models are shown representing two different instances of a simplified behaviour (a simple characteristic curve and a static model using mechanical and electrical components). Furthermore, two dynamic models are presented which describe the dynamic behaviour of a wind turbine in more detail (respecting only the dynamics of the mechanical subsystem or taking into account the dynamics of mechanical and electrical components). In addition, a method of model exchange at certain points in time is proposed. Such structural changes allow the application of that particular model of behaviour which suits the current situation best. Using this method, the simulation of a complex mechatronic system like a wind turbine could very effectively be carried out.

Additionally, some simulation results using the two mechatronic models are given. Both a functionality test performed under extreme conditions as well as an investigation using a realistic wind profile are included.

References

[1] Breitenecker F et al. Structure of simulators for hybrid systems – general development and introduction of a concept of external and internal state events. *6th EUROSIM Congress on Modelling and Simulation – EUROSIM 2007*, Ljubljana, Slovenia, September 9-13, 2007, Proceedings.

[2] Desoer CA, Kuh ES. *Basic Circuit Theory.* McGraw-Hill, 1966.

[3] Enge O, Clauß C, Schneider P, Schwarz P, Vetter M, Schwunk S. Quasi-stationary AC Analysis Using Phasor Description With Modelica. *5th International Modelica Conference – Modelica 2006*, Vienna, Austria, September 4-5, 2006, Proceedings, pp. 579-588.

[4] Enge O, Kielau G, Maißer P. Modelling and Simulation of Discrete Electromechanical Systems. 3rd Conference on Mechatronics and Robotics, Paderborn, Germany, October 4-6, 1995, In Lückel, J. (Ed.): *Mechatronics and Robotics - From Design Methods to Industrial Applications*, Teubner, Stuttgart, 1995, pp. 302-318.

[5] Enge-Rosenblatt O, Bastian J, Clauß C, Schwarz P. Numerical Simulation of Continuous Systems with Structural Dynamics. *6th EUROSIM Congress on Modelling and Simulation – EUROSIM 2007*, Ljubljana, Slovenia, September 9-13, 2007, Proceedings.

[6] Garsch R, Twele J. *Wind Power Plants – Fundamentals, Design, Construction and Operation*. James and James, 2002.

[7] Gasch R (Ed.). *Windkraftanlagen – Grundlagen und Entwurf.* B.G. Teubner, 1996.

[8] Hau E. *Windkraftanlagen – Grundlagen, Technik, Einsatz, Wirtschaftlichkeit.* Springer, 1996.

[9] Heier S. *Grid Integration of Wind Energy Conversion Systems*. John Wiley & Sons, 1998.

[10] Nytsch-Geusen C et al. Mosilab: Development of a Modelica based generic simulation tool supporting model structural dynamics. *4th International Modelica Conference – Modelica 2005*, Hamburg, Germany, March 7-8, 2005, Proceedings, pp. 527-535.

[11] Schwarz P. Simulation of systems with dynamically varying model structure. *5th IMACS Symposium on Mathematical Modelling – MATHMOD 2006*, Vienna, Austria, February 8-10, 2006.

[12] Steinmetz CP. Theory and Calculations of Alternating Current Phenomena. 1e, W.J. Johnson Comp., 1897, or 4e, McGraw, 1908.

[13] Woodson HH, Melcher JR. *Electromechanical Dynamics – Part I: Discrete Systems*. John Wiley & Sons, 1968.

[14] Zhang T. *Energieertrag und dynamische Belastungen an einer Windkraftanlage mit stufenlosem leistungsverzweigten Getriebe bei aktiver Dämpfung.* Dissertation, TU Chemnitz, GUC-Verlag, Chemnitz, 2004.

[15] Zhao X. *Simulation des dynamischen Verhaltens einer Windenergieanlage als mechatronisches System.* Dissertation, TU Chemnitz, Shaker, Aachen, 2004.

[16] http://www.windpower.org/en/tour.htm. Guided Tour on wind energy. Danish Wind Industry Association, seen on October 25, 2007.

ExcelInterface – A Tool for Interfacing Dymola through Excel

Kristian Tuszynski,
Modelon AB, Ideon Science Park, SE-22370 Lund, Sweden

kristian.tuszynski@modelon.se

Abstract

This paper presents a tool created in Excel which enables interfacing with Dymola. The tool was created to simplify batch simulations and allow easy post processing of a large number of simulations. The interface handles both steady state sweeps of a model as well as continuing from a previous simulation. Support for calibration using linear regression is also implemented which allows calibration of simpler models.

Keywords: Excel; Simulation; DDE; Scripting; Dymola; Batch simulation; Steady State; Interface

1 Introduction

When simulating a large number of cases, either to validate a model against measurement data or when acquiring experimental results, there is strong need to be able to organize and get a good overview of both the experiment setup and the result of the simulations. The ExcelInterface greatly improves and simplifies both the post processing and setup involved when running a batch of simulations with changing boundary conditions between the cases. The tool allows the user to define a number of cases to run and then get the result from the simulations presented in Excel for easy comparison. This gives a good overview of what has been set in the model, without actually changing the model code allowing the model stored in Dymola to be generic and instead all different simulation cases are defined in the Excel sheet.

Doing the same thing using Dymola directly would force the user to make model changes for each parameter set, create multiple models where each uses a different parameter set or make a custom made script file where the simulation cases are defined. All these options are quite time consuming and do not provide a good overview.

Having the result in Excel also enables the use of the tools included in the program. Excel and its tools have the advantage that the knowledge and use of them are wide spread which means that it is not necessary for a person with Modelica or Dymola knowledge to analyze and make further post processing of the result. This simplifies the result exchange when working with someone without any prior Modelica knowledge.

2 Overview

The tool is built using VBA (Visual Basic for Applications) which comes with Excel. The communication between Excel and Dymola is performed both using files and a DDE connection established between Dymola and Excel.

When simulating, the interface works by creating a Dymola script based on chosen settings in the ExcelInterface. This script is executed in Dymola, using DDE commands sent from Excel. For each simulation, specified output values are saved in temporary files which are read by Excel after all simulations have completed. In Excel the result is presented at position and with appearance defined by the user through the interface. The communication is illustrated in Figure 1.

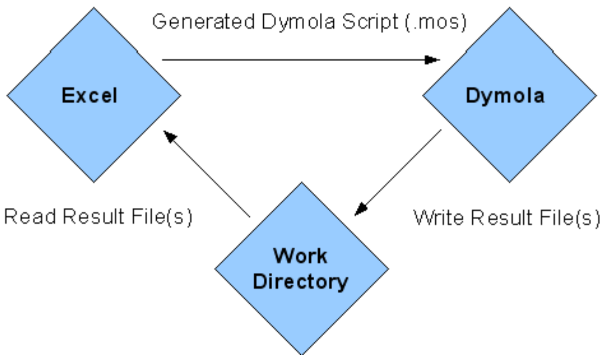
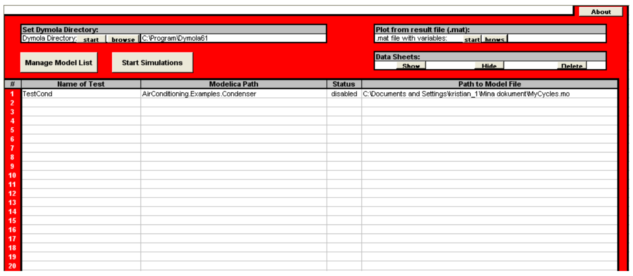


Figure 1 Communication between Excel and Dymola

3 Setup

The ExcelInterface contains a setup sheet, seen in Figure 2, where all cases to be simulated are specified. A case is defined by a unique name used in the interface, a path to a model file and the Modelica path to the model to be simulated within the file. Each case can be enabled and disabled deciding if they are run or not when starting the simulations.

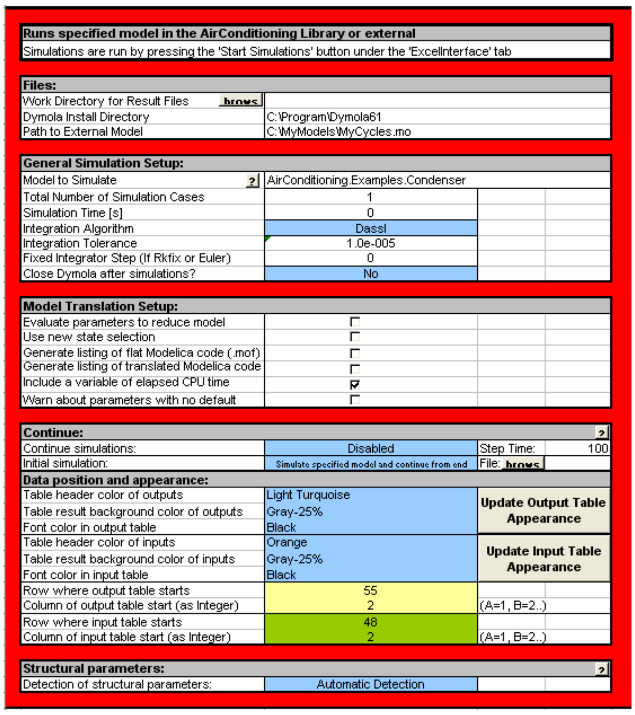


Set Dymola Directory:	
Dymola Directory:	C:\Program\Dymola61
Plot from result file (.mat):	
.mat file with variables:	
Data Sheets:	

#	Name of Test	Modelica Path	Status	Path to Model File
1	TestCond	AirConditioning.Examples.Condenser	disabled	C:\Documents and Settings\kristian_1\Mina dokument\MyCycles.mo
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

Figure 2 Setup sheet of the ExcelInterface

For each case a new excel sheet is created where the user has to specify a number of parameters including work directory, integrator, tolerance, number of simulations cases and simulation time as seen in Figure 3.



Runs specified model in the AirConditioning Library or external

Simulations are run by pressing the 'Start Simulations' button under the 'ExcelInterface' tab

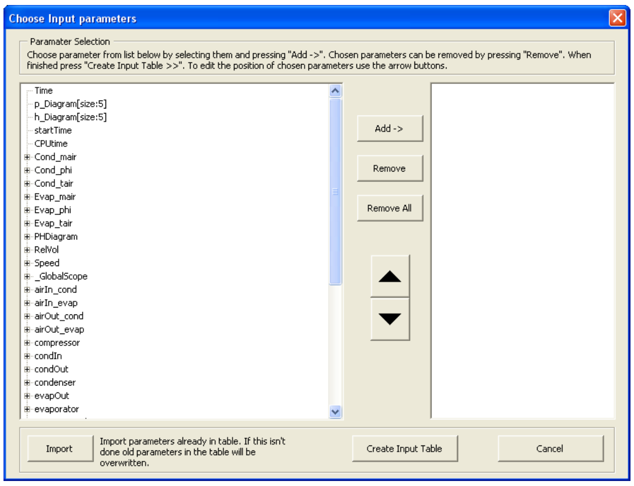
Files:			
Work Directory for Result Files			
Dymola Install Directory	C:\Program\Dymola61		
Path to External Model	C:\MyModels\MyCycles.mo		
General Simulation Setup:			
Model to Simulate	AirConditioning.Examples.Condenser		
Total Number of Simulation Cases	1		
Simulation Time [s]	0		
Integration Algorithm	Dassl		
Integration Tolerance	1.0e-005		
Fixed Integrator Step (If Rkfix or Euler)	0		
Close Dymola after simulations?	No		
Model Translation Setup:			
Evaluate parameters to reduce model	☐		
Use new state selection	☐		
Generate listing of flat Modelica code (.mof)	☐		
Generate listing of translated Modelica code	☐		
Include a variable of elapsed CPU time	☑		
Warn about parameters with no default	☐		
Continue:			
Continue simulations:	Disabled	Step Time:	100
Initial simulation:	Simulate specified model and continue from end	File:	
Data position and appearance:			
Table header color of outputs	Light Turquoise		
Table result background color of outputs	Gray-25%		
Font color in output table	Black		
Table header color of inputs	Orange		
Table result background color of inputs	Gray-25%		
Font color in input table	Black		
Row where output table starts	55		
Column of output table start (as Integer)	2	(A=1, B=2..)	
Row where input table starts	48		
Column of input table start (as Integer)	2	(A=1, B=2..)	
Structural parameters:			
Detection of structural parameters:	Automatic Detection		

Figure 3 Sheet specific for a case

The input and output variables to/from the model are selected from menus in Excel. The first time a model is to be simulated through the ExcelInterface, the model has to be analyzed to find all parameters and variables contained in the model. A DDE connection is used between Excel and Dymola to execute the commands necessary to perform these operations which include:

- translation and simulation of the model
- parameter and variable names extraction from the generated result file
- saving extracted parameter and variable names in a user specified file

This procedure only has to be performed once for every model as any following need to add parameters/variables the saved file is used.



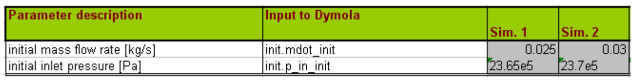
Choose parameter from list below by selecting them and pressing "Add ->". Chosen parameters can be removed by pressing "Remove". When finished press "Create Input Table >>". To edit the position of chosen parameters use the arrow buttons.

```
Time
p_Diagram[size:5]
h_Diagram[size:5]
startTime
CPUtime
⊞ Cond_mair
⊞ Cond_phi
⊞ Cond_tair
⊞ Evap_mair
⊞ Evap_phi
⊞ Evap_tair
⊞ PHDiagram
⊞ RelVol
⊞ Speed
⊞ _GlobalScope
⊞ airIn_cond
⊞ airIn_evap
⊞ airOut_cond
⊞ airOut_evap
⊞ compressor
⊞ condIn
⊞ condOut
⊞ condenser
⊞ evapOut
⊞ evaporator
```

Import parameters already in table. If this isn't done old parameters in the table will be overwritten.

Figure 4 Tree view menu with parameters and variables

Input and output variables are then selected from the generated tree view menu, seen in Figure 4, and finally values are set in the generated input table such as the one shown in Figure 5.



Parameter description	Input to Dymola	Sim. 1	Sim. 2
initial mass flow rate [kg/s]	init.mdot_init	0.025	0.03
initial inlet pressure [Pa]	init.p_in_init	23.65e5	23.7e5

Figure 5 Set boundary conditions

4 Running Simulations

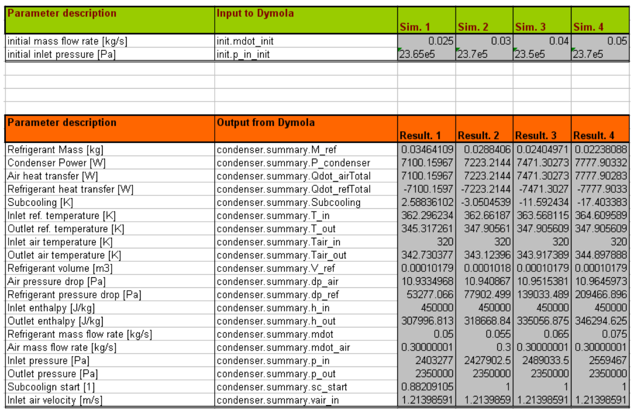
There are two ways to run multiple simulations using the ExcelInterface:

- Steady State Simulations
- Continue Simulation
  - Continue from First
  - Continue from Previous

4.1 Steady State Simulations

This option simulates the specified cases one after another and the results at the specified end time are returned. In case of a model that initializes in steady state, the simulation time should be set to zero for faster execution, for all other cases the user has to determine a simulation time that is long enough for the simulation to reach steady state.

Structural parameters are parameters which force a re-compilation of the model as they change the generated code structure. A good example of structural parameters is discretization parameters. If all selected input parameters are non-structural the model is only translated and compiled once, enabling fast simulations. As it may not always be trivial to know which parameters in a model that are structural, the ExcelInterface automatically detects if a structural parameter was selected as an input parameter. If one or more structural parameters are detected the model has to be retranslated between each run case.



Parameter description	Input to Dymola	Sim. 1	Sim. 2	Sim. 3	Sim. 4
initial mass flow rate [kg/s]	init.mdot_init	0.025	0.03	0.04	0.05
initial inlet pressure [Pa]	init.p_in_init	23.65e5	23.7e5	23.5e5	23.7e5

Parameter description	Output from Dymola	Result. 1	Result. 2	Result. 3	Result. 4
Refrigerant Mass [kg]	condenser.summary.M_ref	0.03464109	0.0288406	0.02404971	0.02238088
Condenser Power [W]	condenser.summary.P_condenser	7100.15967	7223.2144	7471.30273	7777.90332
Air heat transfer [W]	condenser.summary.Qdot_airTotal	7100.15967	7223.2144	7471.30273	7777.90283
Refrigerant heat transfer [W]	condenser.summary.Qdot_refTotal	-7100.1597	-7223.2144	-7471.3027	-7777.9033
Subcooling [K]	condenser.summary.Subcooling	2.58836102	-3.0504539	-11.592434	-17.403383
Inlet ref. temperature [K]	condenser.summary.T_in	362.296234	362.66187	363.568115	364.609589
Outlet ref. temperature [K]	condenser.summary.T_out	345.317261	347.90561	347.905609	347.905609
Inlet air temperature [K]	condenser.summary.Tair_in	320	320	320	320
Outlet air temperature [K]	condenser.summary.Tair_out	342.730377	343.12396	343.917389	344.897888
Refrigerant volume [m3]	condenser.summary.V_ref	0.00010179	0.0001018	0.00010179	0.00010179
Air pressure drop [Pa]	condenser.summary.dp_air	10.9334968	10.940867	10.9515381	10.9645973
Refrigerant pressure drop [Pa]	condenser.summary.dp_ref	53277.066	77902.499	139033.489	209466.896
Inlet enthalpy [J/kg]	condenser.summary.h_in	450000	450000	450000	450000
Outlet enthalpy [J/kg]	condenser.summary.h_out	307996.813	318668.84	335056.875	346294.625
Refrigerant mass flow rate [kg/s]	condenser.summary.mdot	0.05	0.055	0.065	0.075
Air mass flow rate [kg/s]	condenser.summary.mdot_air	0.30000001	0.3	0.30000001	0.30000001
Inlet pressure [Pa]	condenser.summary.p_in	2403277	2427902.5	2489033.5	2559467
Outlet pressure [Pa]	condenser.summary.p_out	2350000	2350000	2350000	2350000
Subcoolign start [1]	condenser.summary.sc_start	0.88209105	1	1	1
Inlet air velocity [m/s]	condenser.summary.vair_in	1.21398591	1.2139859	1.21398591	1.21398591

Figure 6 Input and output in Excel

Figure 6 shows an example on how the output in Excel can look like after a successful simulation. In the example four different cases were run and two parameters (*init.mdot_init* and *init.p_in_init*) were changed between the simulations.

4.2 Continue Simulation

Besides running each simulation as a separate case, the ExcelInterface offers two other ways for series of steady state calculations. The most common reasons for using these options are that the model can not successfully initialize at every steady state point and/or that the initialization phase of the model is very time consuming making it practical to continue from a initialized model that has reached steady state.

By connecting ramp blocks to the boundary conditions of the simulated model where the start values of the output signals equal the end value from the previous simulation it is possible to start each new simulation from steady state and then change the boundary conditions by setting desired height of the ramp blocks.

Continue from First simulates the model for a specified time and then the remaining simulations continue from this point. This makes it possible to define a number of transients using, for instance, ramp blocks and sweep any number of steady state points.

Figure 7 shows the outlet evaporator temperature in an AC-cycle simulation where the model was first simulated until it was in steady state. Once this point was reached (after 200 seconds) five simulations were executed from the end of the first simulation where the inlet air temperature was changed between the simulations.

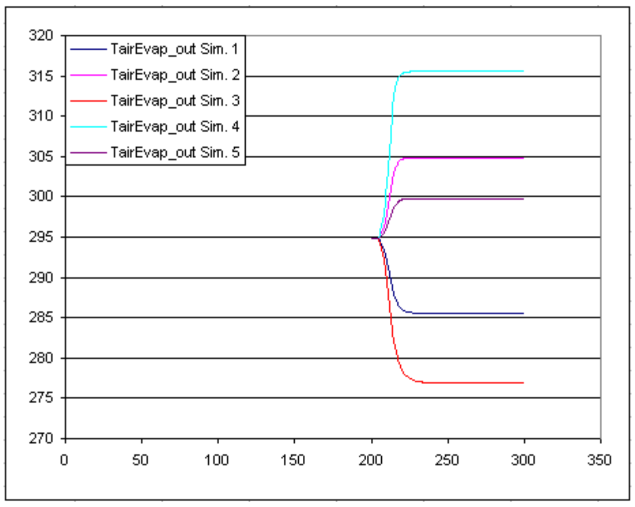


Figure 7 Example result when using "Continue from First"

The second option **Continue from Previous** also simulates the model for a specified time and then each specified case continues from the previous simulation.

A **Continue from Previous** run is illustrated in Figure 8 where the outlet evaporator temperature of an AC-cycle is shown. The initial simulation is continued after 200 seconds. After this time 5 simulations are run where each one is 100 seconds long.

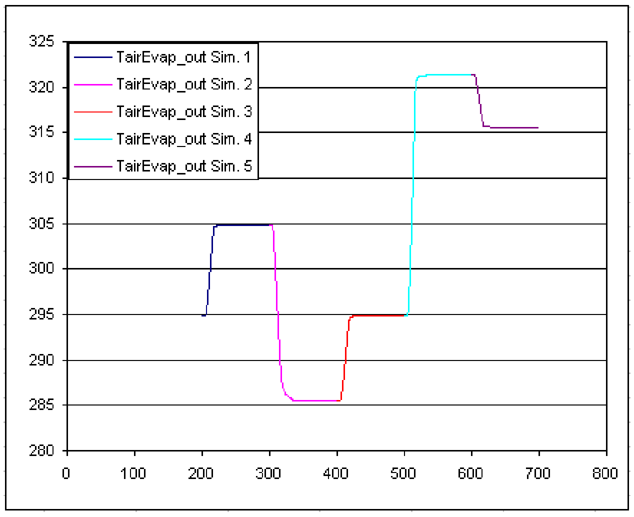


Figure 8 Example result when using “Continue from Previous”

For both these options it is possible to start the simulations from a saved result file. Using this option the initial simulation, which takes the model to steady state, is skipped. Instead all initial values are taken from the result file.

4.3 Plotting and Dynamic Result

For all simulations performed using the interface it is optional to include plots of chosen variables in Excel. Enabling this option is useful to get a quick visual comparison of the different simulation results and when it is necessary to verify that the model really reached steady state after an initial transient.

The plots are created by extracting wanted trajectories from the result files and saving them in sheets within the work book making the trajectories easily accessible.

5 Usage Example – Charge Optimization

To find the optimal charge of an AC-Cycle the cycle is first almost completely drained and then filled in multiple steps until the accumulator of the cycle is over filled. At each step important values such as the power, pressures, subcooling and superheat temperatures are measured.

Simulating this procedure in one continuous simulation might prove difficult as it is often necessary to simulate between 8 and 15 points altogether and there is a risk that the simulation will fail during the transition between, at least, two of the points. If this happens it is quite time consuming to re-run the simulation and there are no guarantees it will work the second time around either.

Using the ExcelInterface the risk is minimized when simulating the charge optimization using the continue feature of Dymola. The experiment is setup by adding a controlled flow source, to the cycle, which fills the accumulator with refrigerant at specified time as shown in .

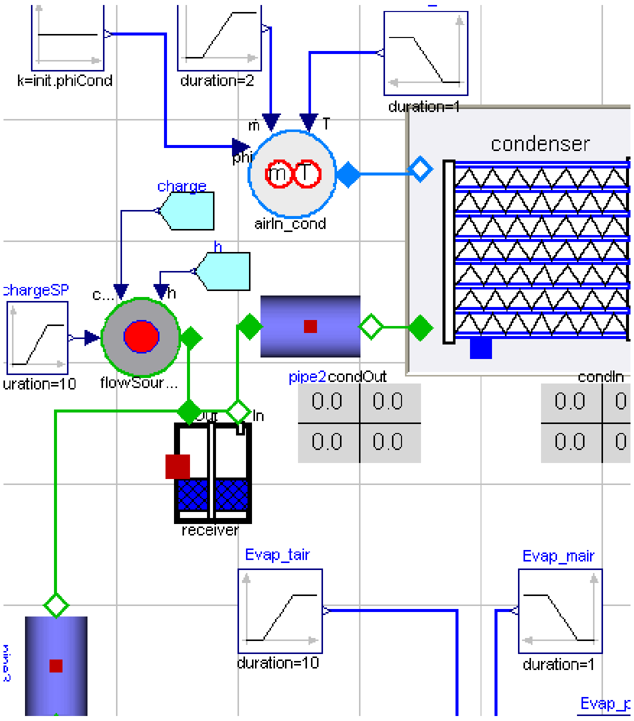
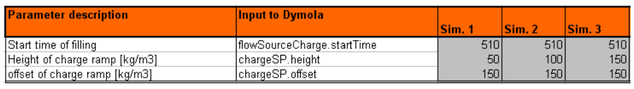


Figure 9 Cycle with controllable flow source

In the ExcelInterface the start time of the filling, height and offset of the set-point ramp block is selected as input parameters and the experiment is run using **Continue from First**. Finally, the cycle model is parameterized to begin the continue simulations having a charge of 150 kg/m^3.



Parameter description	Input to Dymola	Sim. 1	Sim. 2	Sim. 3
Start time of filling	flowSourceCharge.startTime	510	510	510
Height of charge ramp [kg/m3]	chargeSP.height	50	100	150
offset of charge ramp [kg/m3]	chargeSP.offset	150	150	150

Figure 10 Setup in the ExcelInterface

Assuming that the initial simulation, which controls the charge down to 150 kg/m^3 passes the whole experiment will not fail if a single simulation fails. Instead of risking having to redo the whole experiment the worst case scenario is now that some of the simulations have to be redone because they crashed.

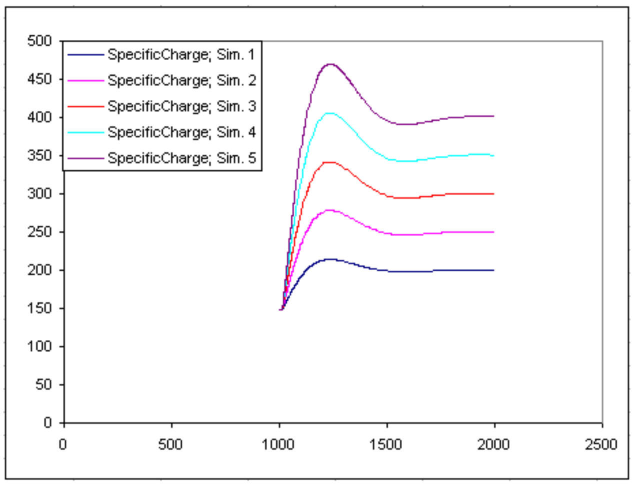


Figure 11 Plot of specific charge. 5 points were simulated from 200-400 kg/m3

6 Summary

The ExcelInterface has proven to be an efficient tool to use when doing batch simulations over a large number of steady state points.

The interface gives the user a good overview of the cases to simulate and simplifies the post processing of the result as well as speeding up the setup of the experiments. This in combination with the fact that Excel is a well known program which many people have experience working with gives the interface great flexibility and a broad user base.

References

[1] Excel VBA Language Reference, www.microsoft.com

[2] Dymola User Manual

Modeling of Cold Plates for Power Electronic Cooling

Karin Dietl[*] Jens Vasel[†] Gerhard Schmitz[‡] Wilson Casas Christian Mehrkens
Hamburg University of Technology
Institute of Thermo-Fluid Dynamics,[§] Applied Thermodynamics
21071 Hamburg, Germany

Abstract

This paper deals with the cooling of high power electronic devices. Usually those devices dissipate 5 - 10% of their electrical power, therefore (convective) cooling is needed. Power electronics can be cooled directly by air or a non-conductive fluid via (forced) convection. However discharging the heat of the power electronics via convective heat transfer with air leads often to a large cooling elements due to the poor heat transfer coefficient of air. Also in most applications the direct contact between the electronic and the cooling fluid is undesirable.
For these applications the use of cold plates can be an option. The fluid flows through a plate (see. fig.1) which is directly connected to the electronic. This type of cooling is far more effective than air cooling, since the cold plate can be designed in order to cool also high power density electronics without resulting in a disproportional increase of the space envelope. The fluid temperature can be increased with respect to the air temperature, without decreasing reliability and durability of the power electronics components.
Using cold plates open up possibilities of decentralised cooling which can improve the efficiency of the cooling system.
This paper presents a model library developed in order to model power electronics cooling. The library provides on the one hand heat loss models for basic power electronics equipment itself, like IGBTs, and on the other hand thermodynamic models for different cold plates. Lumped models of the cold plates can be used in large system simulations whereas cold plate models using a distributed approach are foreseen for more detailed analysis.
To be able to calculate the temperature distribution in the cold plate, the solid and fluid parts of the cold plate have to be discretised in all directions (see fig.2).

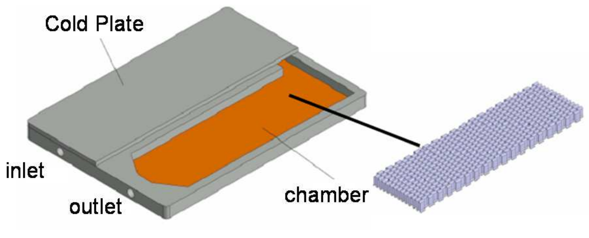


Figure 1: Cold plate

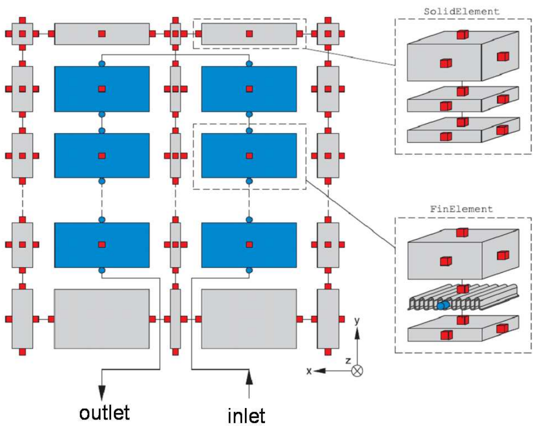


Figure 2: Cold plate model

The library is based on Modelica.Fluid, however for the modeling of the single phase cooling medium, the compressibility of the liquid is considered in order to avoid large non-linear system of equations.
An important aspect of the library is the coupling of the power electronic models to the cold plate model. Hereby an efficient algorithm is needed which enables the user to connect an unlimited number of power electronic components of any size to arbitrary places on the cold plate.

[*]email: karin.dietl@tu-harburg.de, Tel:+49 4042878 3765
[†]email: vasel@tu-harburg.de, Tel:+49 4042878 3765
[‡]email: schmitz@tu-harburg.de, Tel:+49 4042878 3144
[§]www.tt.tu-harburg.de

Additional to the simulations a test rig is built, where the cold plates are tested. Whereas the models can be used for both, single phase and two phase cooling, on the test rig only single phase cooling is investigated. Since the fluid channels often have a complicated finned structure, where the geometric parameters are usually not accessible, the measurements are needed to validate the cold plate models. Hereby a large emphasis is placed on validating the pressure drop and heat transfer correlations, as well as the time constants.

Keywords: Modelica; Simulation; Cold plate; Cooling; Power Electronics

Heavy Vehicles Modeling with the Vehicle Dynamics Library

Niklas Philipson   Johan Andreasson
Magnus Gäfvert   Andrew Woodruff

Modelon AB
Ideon Sience Park, SE-22370 Lund, Sweden

Abstract

This paper presents and describes recent extensions to the Vehicle Dynamics Library (VDL) for heavy and commercial road-vehicle modeling and simulation (VDL/Trucks). Until now, the VDL was targeted mainly at passenger cars applications (VDL/Cars). Users in this domain have been particularly enthusiastic about the openness, flexibility, and extensibility compared to many competing solutions. These advantages which are inherent to Modelica technology are even more important for heavy vehicles applications, where a much larger set of vehicle configurations and variations must be supported. It has therefore been natural to extend the scope of the library also into this field with the VDL/Trucks options presented in this paper. New components and templates have been introduced to reflect many standard chassis layouts. A number of new experiment templates are also supplied to make standard analysis tasks easy to perform.

Keywords: heavy vehicles; trucks; vehicle dynamics; Vehicle Dynamics Library

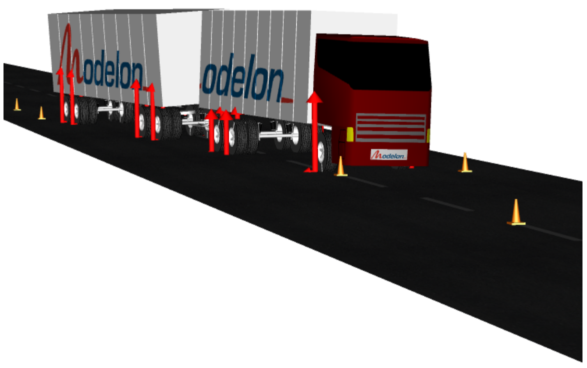


Figure 1: Truck-fulltrailer in a double lane-change

1 Introduction

The Vehicle Dynamics Library (VDL) [1, 2] was originally designed for studies on vehicle handling for passenger cars (VDL/Cars). It was early clear that an extension into the heavy vehicles domain would be natural. The inherent flexibility and extensibility of the Modelica-based solutions offers great benefits in this domain where a vast set of vehicle configurations and variants must be handled, such as combinations of trucks, tractors, full trailers, semi-trailers, tankers, with various axle and powertrain configurations, and also a wide range of payload conditions. This paper introduces the VDL/Trucks option of VDL aimed at modeling and simulation of heavy vehicles.

Vehicle dynamics analysis of heavy vehicles and passenger cars have many common inputs such as a human driver model with similar driver-vehicle interface, road and environment properties, etc, and outputs of interest such as tire forces at the contact patches, chassis and suspension motion. Joints, links, springs, dampers, drivers, roads, and tires all produce similar types of constraints on the model. A large set of model components are therefore common for cars and truck modeling. There are, however, some major differences between heavy commercial vehicles and passenger cars when it comes to chassis layout. The number of axles, tires and trailers are some of the many parameters that are combined to form a heavy vehicle configuration, while cars have a more static setup. This requires an even more flexible interface and template design for heavy vehicles than for cars.

The heavy vehicles option has been developed from the same library base as the car option. This means that the new heavy vehicle models can benefit from an already well tested and mature overall design.

2 Heavy vehicle components

As mentioned above, there are many low level components that are shared between VDL/Cars and VDL/Trucks, but there are of course many examples of new components and components that are used differently in the context of heavy vehicles [3]. Essentially, this is due to the difference in weight and dimensions. The higher over-all weight requires different solutions and very large load variations means that good performance have to be achieved for a wide variety of load cases. The higher center-of-gravity makes rollover rather than road adhesion the handling limit in many situations. This section highlights some of the extensions made to VDL for heavy vehicle simulation to address these differences. Figure 2 shows a screen shot of parts of the library, indicating some important new additions.

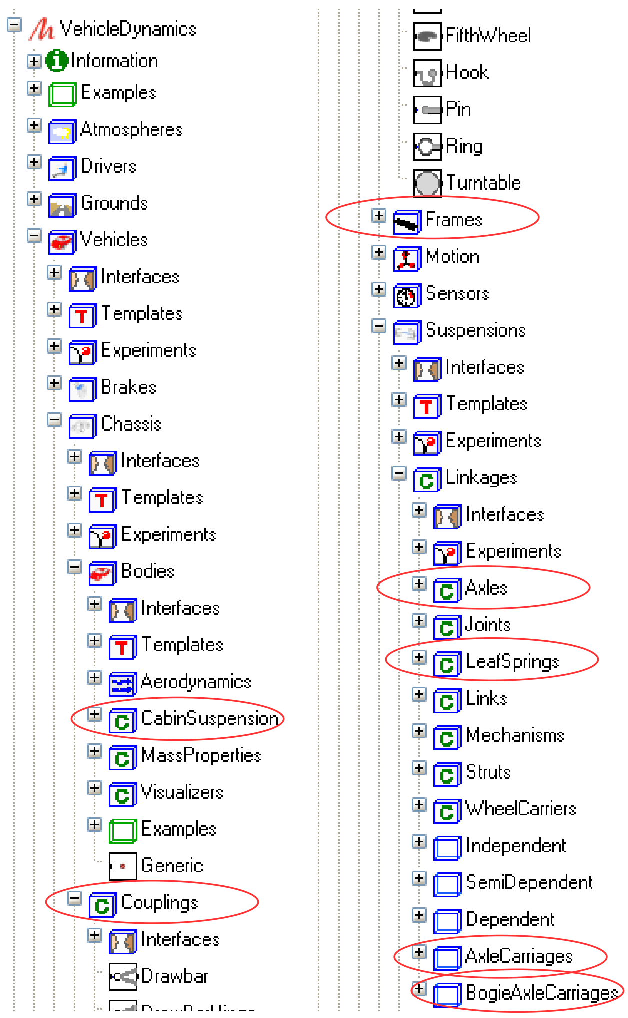


Figure 2: Screen shot of parts of VDL as it appears in Dymola. Some main extensions to VDL for heavy vehicles are indicated.

Suspension The suspension designs in heavy trucks are usually axle-based for the steerable and non-steerable wheels. Leaf springs are commonly used for both axle guidance and load support and are implemented as described in [4]. To meet the requirement of high load variations the leaf springs are often mounted in such a way that the effective length of them decreases when they are subjected to load. There are also leaf spring versions that are equipped with helper springs that becomes active when the vehicle is loaded. Air springs are often used in heavy vehicle suspensions in conjunction with trailing arms to easily adjust for different load cases, see Figure 3. Air springs can be used to change the ride height of the vehicle by increasing the air mass inside the spring, which also results in a stiffer spring that can carry more load.

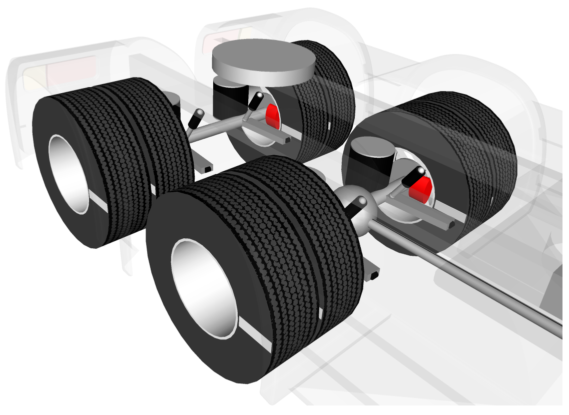


Figure 3: Typical heavy vehicle rear axles.

Frame The frame elasticity influences the load distribution between the axles, and therefore the available grip from the tires. The elastic frame included in VDL trucks has a torsional degree of freedom. It is easy to add or change the degrees of freedom in the frame by extending the interface so the common template connectors are used.

Payloads The payloads can be static (e.g. a crane), dynamic (e.g. a tank for liquid load) or have varying masses or mass distributions (e.g. cargo containers). These different cases are supported with user-friendly configuration setup. The existing liquid payload model considers the dimensions of the tank and a rotational damped degree of freedom for slosh.

Cabin The truck cabin is usually suspended for driver comfort since the chassis suspension must be

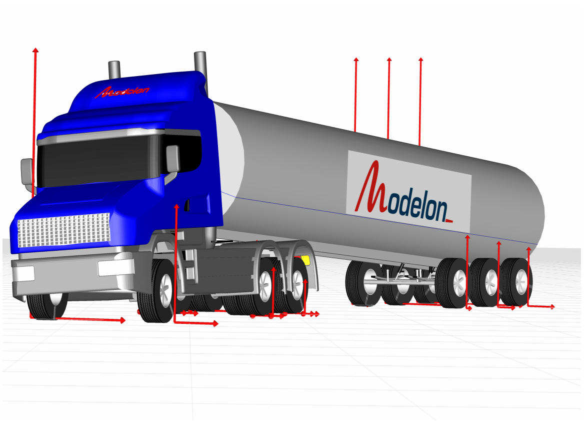


Figure 4: Sine-excitation of a tractor-trailer combination with the liquid load in the tank modelled with with a one degree of freedom to capture dynamic load distributions.

stiff to accommodate the high loads. The cabin suspension is a linkage mechanism equipped with springs, dampers and antiroll bars. The suspension also causes a relative motion between the steering wheel and the steering gear since the steering wheel moves with the suspended cabin. This is incorporated in the vehicle templates and ensures that it is easy to change the different subsystems such as the cabin suspension or the steerable axle linkage in an flexible way.

Couplings Heavy vehicle combinations often have tractor (driven) and trailer vehicle units. The attachment to guide and constrain the trailer can vary, but has a significant effect on the handling and vehicle behavior. One of the most common couplings is a fifth wheel for the tractor/semi-trailer combination. Full trailers and dollies usually have a draw bar and hook to attach to the tractor, driven truck, or preceding trailer (in the case of road trains). The coupling must have a mass on both sides of the joint or be locked to avoid a singular setup when no unit is attached on one side. This is conveniently handled without much user intervention by the available components and templates.

3 Heavy vehicle templates

The variation and configuration space of heavy vehicle combinations are much larger then for normal passenger cars. Also, the components can be of varying fidelity depending on the design and purpose of the model. A new set of templates for heavy vehicle components has therefore been developed to sustain the user-friendliness offered in VDL/Cars. These new templates are based on the same usage principles as the car templates, where templates for aggregate models are built by connecting replaceable components that can be parameterized depending on application. An example of a tractor template is given in Figure 5.

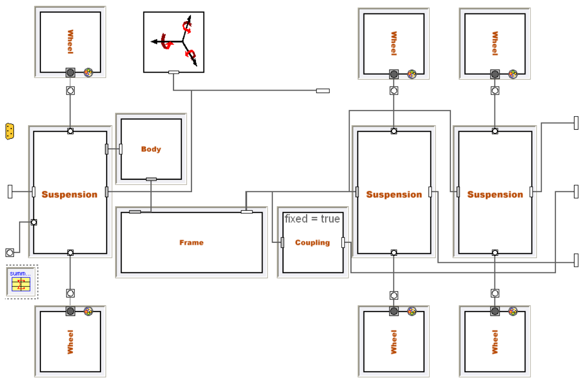


Figure 5: Tractor template with two rear axles.

The heavy vehicle interfaces for basic components are largely the same as those for cars. Some changes include extra connections to incorporate the frame and suspended cabin. The axle-based suspension models typically have connectors for the axle and chassis, instead of using separate connectors for the left and right suspension linkage models. The templates still have all connections and parameters predefined and propagated between models so they only require the replaceable components to be redeclared from the graphical Dymola user interface.
The main chassis components include a number of suspension models that contain one or more axles, frame, coupling, wheels, and a body or payload. Trailers can also include components for a dolly.
The suspension templates are based on axle constraints. The axle can be a steerable or non-steerable version. The axle connects through the linkage to the chassis. The linkage has external or internal force elements such as coil/air springs or leaf springs, respectively, to support the chassis. An anti-roll bar is attached to the axle and chassis. The suspension components vary from the most basic bounce and roll degrees of freedom to detailed elasto-kinematic setups.

4 Experiments

Simulation experiments for passenger cars and heavy vehicles have many similarities and correspondingly share setup of e.g., drivers, roads and grounds, and environments. Just as for cars, both the open and closed loop driver models are available. The double lane-

change road maneuver as seen in Figure 1 is useful for emergency handling evaluation since it excites the roll motion which may cause roll-over [5]. The experiment is set up using a driver model that follows the road path defined by RoadBuilder [2]. The sine steering excitation experiment shown in Figure 4 is instead realized using an open loop steering robot while a drive robot is keeping the speed constant.
For out-of-plane frequency response, the shaker table can be used [2]. It is implemented as a ground model containing patches with time dependent altitudes, defined by inputs. Since a heavy vehicle can have more than two axles, the shaker table has a configurable number of patches to suite any number of axles and wheel locations. Correspondingly, several suspension rigs can be used together for detailed analysis of bogie axles, see Figure 6.

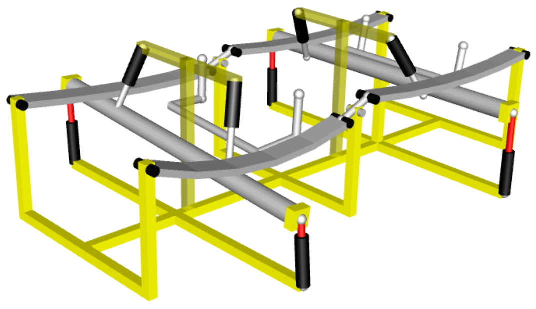


Figure 6: Twin axle with load distribution linkage in a suspension rig.

5 Customization

Just as for passenger cars and light vehicles, VDL/Trucks is extended with a set of examples for heavy vehicles. This includes both truck with full trailer and tractor with semitrailer as seen in Figures 1 and 4, respectively. Thanks to the flexibility inherent in the library, it is straightforward to re-configure these and even build completely different equipages, as illustrated by the examples in this section.

5.1 Road Train

Equipages with combinations of a truck or tractor with two or more trailers forms a road train of the type commonly used in e.g. Australia. These vehicle combinations allow for one driver to freight a larger amount of cargo compared to a tractor pulling a single trailer. Unlike rail-carried trains that are self steered by the rail-wheel interaction, road trains are more sensible to disturbances and may even exhibit instability if care is not taken. Additionally, road trains are heavy and thereby hard to stop which requires them to be able to steer to avoid accidents. This puts high demands on the design of trucks and trailers so that they safely can be combined into road trains under a variety of load conditions. In VDL, these configurations can be defined and tested conveniently. Figure 7 shows a set-up with tree trailers pulled by a tractor.

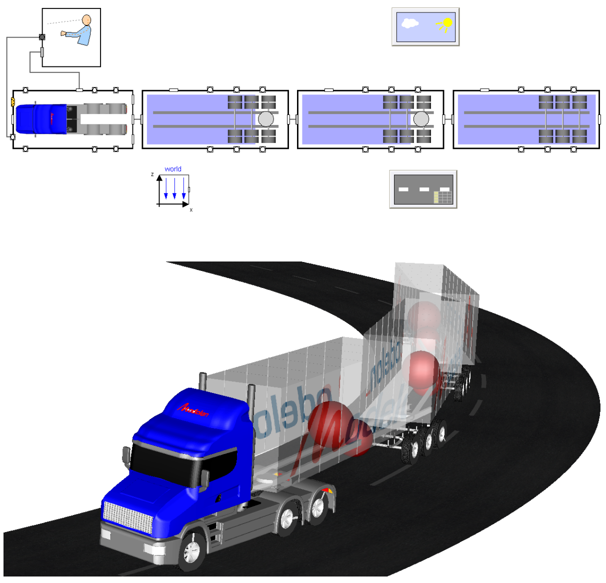


Figure 7: Road train with three trailers, diagram view (top) and animation screen shot (bottom).

5.2 Moving tire test rig

Tire test rigs are can be subdivided into two main categories depending on if the tested wheel or the ground surface is moving. For the latter case, the ground is typically implemented as drum or a belt. The drawback with these two concepts are on one hand that the belt only makes it possible to use elastic surface material such as steel and on the other hand that a drum has to have a curvature which impacts the tire-surface contact. To avoid this and to enable testing on real road surfaces such as gravel, asphalt, and ice under different conditions with respect to moisture, temperature, and so on, the tested wheel can be mounted on a moving rig, typically attached to a heavy truck. However, a moving rig is harder to control, especially since the forces generated from the tested wheel will affect the course of the truck. To investigate both the static and dynamic effects of the total system of truck, rig, and tire on resulting measurements, a moving tire test rig was implemented by mounting a test rig with wheel onto a truck model as illustrated in Figure 8. The results were then compared to standard test rig simulations and real mobile-rig mesurement results and provided insight into the interpretation of sensor signals.

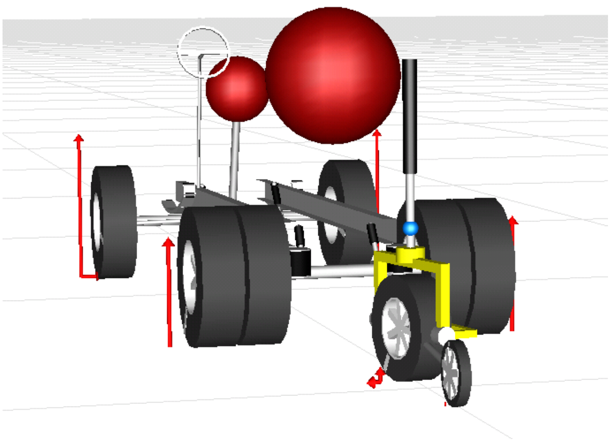


Figure 8: Mobile tire test rig mounted on a truck.

6 Simulink

Just as for passenger cars, heavy vehicles modeled with VDL can be imported into the Simulink [7] environment. Figure 9 shows an experiment layout in Dymola [6] used for the yaw control application in Simulink shown in Figure 10. In applications like this VDL/Trucks can provide models that are of great use in the design and validation of various chassis control functions.

7 Future

Currently the VDL/Trucks option is focused on the chassis and covers well the most commonly used vehicle types. VDL/Cars have more complete support for full vehicle modeling with templates for powertrains, drivelines, brakes, engines, etc. Future development will move in the direction of complete vehicle modeling also for heavy vehicles. Until then, many components are still available to build those subsystems from base classes, but without extensive templates.

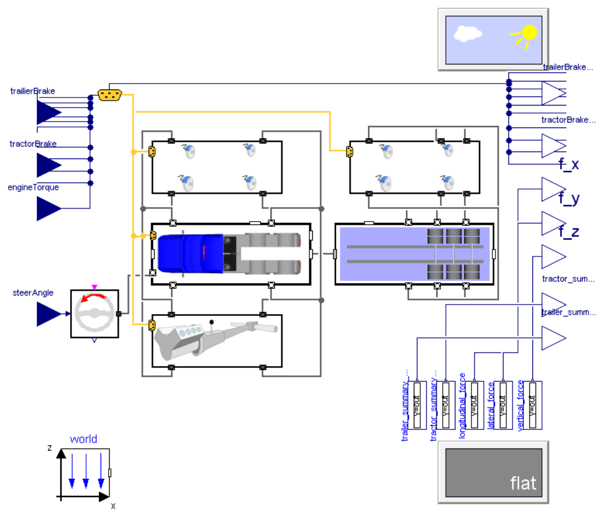


Figure 9: Experiment with in- and outputs for Simulink. Inputs: Steering wheel angle, engine torque, gear, wheel brake clamp forces. Outputs: Vehicle states, tire forces and wheel spin velocities.

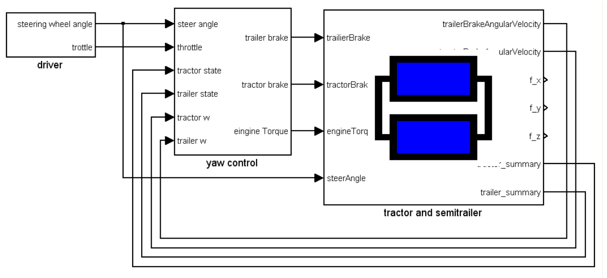


Figure 10: Yaw control application for the tractor-semitrailer combination shown in Figure 9.

8 Summary

This paper shows how the Vehicle Dynamics Library is extended with the VDL/Trucks option for heavy and commercial road-vehicle modeling and simulation. An overview of the recent additions is given and it is shown with several examples how the openness, flexibility, and extensibility from VDL/Cars is maintained and extended.

References

[1] Modelon AB, Lund, Sweden. The VehicleDynamics library, User's Guide, Version 1.2, 2007.

[2] J. Andreasson and M. Gävert, The VehicleDynamics Library - Overview and Applications. In: *Proceedings of the 5th Modelica Conference*, Vienna, Austria, Modelica Association, 4-5 September 2006.

[3] N. Philipson, Extension of a tool for vehicle dynamics studies to handle heavy vehicle configurations. Master Thesis Report, KTH-AVE, 2007.

[4] N. Philipson, Leaf spring modeling. In: *Proceedings of the 5th Modelica Conference*, Vienna, Austria, Modelica Association, 4-5 September 2006.

[5] E. Dahlberg and A. Stensson, The Dynamic Rollover Threshold of Commercial Vehicles - a Sensitivity Study. International Journal of Vehicle Design, Vol. 40, No. 1-3, pp. 228-250, 2006.

[6] Dymola - Dynamic Modelica Laboratory, http://www.dynasim.se

[7] The MathWorks: Matlab and Simulink http://www.mathworks.com

Session 6a

Language, Tools and Algorithms

Compiling and Using Pattern Matching in Modelica

Kristian Stavåker, Adrian Pop, Peter Fritzson
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{krsta, adrpo, petfr}@ida.liu.se

Abstract

Pattern matching is a well-known, powerful language feature found in functional programming languages. In this paper we present the implementation of pattern matching for Modelica. A pattern matching construct is useful for classification and decomposition of (possibly recursive) hierarchies of components such as the union type structures in the MetaModelica language extension. We argue that pattern matching not only is useful for language specification (as in the MetaModelica case) but also to write concise and neat functional-style programs. One useful application is in list processing (lists are currently missing from Modelica but are part of MetaModelica). Other possible applications are in the generation of models from other models, e.g. the generation of models with uncertainty equations.

Keywords: Pattern Matching, Modelica

1 Introduction

Pattern matching is a general operation that is used in many different application areas. Pattern matching is used to test whether things have a desired structure, to find relevant structure, to retrieve the aligning parts, and to substitute the matching part with something else.

In term pattern matching terms are matched against incomplete terms with variables and in, for instance, string pattern matching finite strings are matched against regular expressions (a typical application would be searching for substrings). Term pattern matching (which we will only consider henceforth) can be stated as: given a value v and patterns p1,…,pN is v an instance of any of the p's? Language features for pattern matching (over terms) are available in all functional programming languages, for instance Haskell [3], OCcaml [9], and Standard ML [10].

However, pattern matching is currently missing from Object-Oriented Equation-Based (EOO) Languages such as Modelica [2],[5], VHDL-AMS [7], and gPROMS [8]. Pattern matching features are also rare in imperative object-oriented languages even though some research has been carried out ([12],[13],[14] and [15]). In [15], for instance, the JMatch language which extends Java with pattern matching is described.

[16] promotes the use of pattern matching constructs in object-oriented languages as a mean of exploring class hierarchies. One could for instance apply the visitor pattern to solve the same problem but as [17] notes this require a lot of code scaffolding and nested patterns are not supported.

The pattern matching construct for Modelica was presented in a paper on Modelica Metaprogramming extensions [4]. This paper discusses the implementation of pattern matching in the OpenModelica Compiler (OMC) [1]. We start by introducing the pattern matching construct in section 2 (syntax and semantics) followed by some programming examples in section 3. In section 4 we discuss type systems and problems with matching over class hierarchies. In section 5 we give an overview of implementation issues, followed by conclusions in section 6.

2 Pattern Matching

In this section we present the design of the pattern matching expression construct. Pattern matching expressions may occur where expressions can be used in Modelica code.

2.1 Syntax

We begin by giving the grammar rules.

```
match_keyword :
    match
  | matchcontinue

match_expression :
    match_keyword expression
    [ opt_string_comment ]
    local_element_list
    case_list
    case_else
    end match_keyword ";"

case_list :
    case_stmt case_list
  | case_stmt
```

```
equation_clause_case :
    equation equation_annotation_list
  | (* empty *)

case_stmt :
    case seq_pat
    [ opt_string_comment ]
    local_element_list
    equation_clause_case
    then expression ";"

case_else :
    else [ opt_string_comment ]
    local_element_list
    equation_clause_case
    then expression ";"
  | (* empty *)

local_element_list :
    local element_list
  | (* empty *)
```

The grammar rules that have been left out are rather self-describing (except the rule for patterns, `seq_pat`, which will not be covered here). An `equation_annotation_list`, for instance, is a list of equations. Only local, time-independent equations may occur inside a pattern matching expression and this must be checked by the semantic phase of the compiler. The difference between a pattern matching expression with the keyword `match` and a pattern matching expression with the keyword `matchcontinue` is in the fail semantics (see Section 2.3). The syntax can also be given (approximately) as follows.

```
matchcontinue (<var-list>)
  local
    <var-decls>
    ...
   case (<pat-expr>)
  local
    <var-decls>
  equation
    <equations>
  then <expr>;
  ...
end matchcontinue;
```

The `<pat-expr>` expression is a sequence of patterns. A pattern may be:

- A wildcard pattern, denoted _.
- A variable, such as x.
- A constant literal of built-in type such as 7 or true.
- A variable binding pattern of the form `x as` pat.
- A constructor pattern of the form C(pat*1,...,p*atN), where C is a record identifier and pat1,...*,patN* are patterns. The arguments of C may be named (for instance field1=pat1) or positional but a mixture is not allowed. We may also have constructor patterns with zero arguments (constants).

2.2 Semantics

The semantics of a pattern matching expression is as follows: If the input variables match the pattern-expression in a case-clause then the equations in this case-clause will be executed and the matchcontinue expression will return the value of the corresponding then-expression. The variables declared in the uppermost variable declaration section can be used (as local instantiations) in all case-clauses. The local variables declared in a case-clause may be used in the corresponding pattern and in the rest of the case-clause. The matching of patterns works as follows given a variable v.

- A wildcard pattern, _, will succeed matching anything.
- A variable, x, will be bound to the value of v.
- A constant literal of built-in type will be matched against v.
- A variable binding pattern of the form `x as` pat: If the match of pat succeeds then x will be bound to the value of v.
- A constructor pattern of the form C(pat*1,...,p*atN): v will be matched against C and the subpatterns will be matched (recursively) against parts of v.

2.3 Pattern Matching Fail Semantics

If a case-clause fails in an expression with the keyword `matchcontinue` then an attempt to match the subsequent case-clause will take place. If we have an expression with the keyword `match`, however, then the whole expression will fail if there is a failure in one of the case-clauses. We will henceforth only deal with `matchcontinue` expressions.

3 Examples of Pattern Matching

As mentioned earlier a pattern matching construct is useful for language specification (meta-programming) but also as a tool to write functional-style programs. We start by giving an example of the latter usage.

```
function fac
  input Integer inExp;
  output Integer outExp;
algorithm
  outExp := matchcontinue (inExp)
    case (0) then 1;
    case (n) then if n>0 then n*fac(n-1)
             else fail();
  end matchcontinue;
end fac;
```

The above function will calculate the factorial value of an integer. If the number is less than zero the function will fail.

A fundamental data structure kind in MetaModelica is the union type which is a collection of records containing data, see example below.

```
uniontype UT
record R1
String s;
end R1;

record R2
Real r;
end R2;
end UT;
```

The pattern matching construct makes it possible to match on the different records. An example is given below.

```
function elabExp
  input Env.Env inEnv;
  input Absyn.Exp inExp;
  output Exp.Exp outExp;
  output Types.Properties outProperties;
algorithm
  (outExp,outProperties):=
  matchcontinue (inEnv,inExp)
    local
      ...
    case(_,Absyn.INTEGER(value=x))
      local Integer x;
      then (Exp.ICONST(x),Types.
             PROP(Types.T_INTEGER({})));
      ...
    case(env,Absyn.CREF(cRef = cr))
      equation
        (exp,prop) = elabCref(env,cr);
      then (exp,prop);
      ...
    case(env,Absyn.IFEXP(ifExp =
             e1,trueBranch=e2,eBranch=e3))
      local Exp.Exp e;
      equation
        (e1_1,prop1)=elabExp(env,e1);
        (e2_1,prop2)=elabExp(env,e2);
        (e3_1,prop3)=elabExp(env,e3);
        (e,prop)=elabIfexp(env,e1_1,prop1,
             e2_1,prop2,e3_1,prop3);
      then (e,prop);
  end matchcontinue;
end elabExp;
```

The function `elabExp` is used for elaborating expressions (type checking, constant evaluation, etc.). The union type `Absyn.Exp` contains a record representing an integer, a record representing a component reference, and so on. There is an environment union type, `Env.Env`, for component lookups.

Another situation where pattern matching is useful is in list processing. Again, lists are lacking from Modelica but are an important construct in MetaModelica (see the conclusions section for a small discussion). The following function selects an element that fulfills a certain condition from a list.

```
function listSelect
  input list<Type_a> inTypeALst;
  input Func_anyTypeToBool inFunc;
  output list<Type_a> outTypeALst;
  replaceable type Type_a subtypeof Any;
  partial function Func_anyTypeToBool
    input Type_a inTypeA;
    output Boolean outBoolean;
  end Func_anyTypeToBool;
algorithm
  outTypeALst:=
  matchcontinue
  (inTpeALst,inFunc)
  local
    list<Type_a> xs_1,xs; Type_a x;
    Func_anyTypeToBool cond;
  case ({},_) then {};
  case ((x :: xs),cond)
    equation
      true = cond(x);
      xs_1 = listSelect(xs, cond);
    then (x :: xs_1);
  case ((x :: xs),cond)
    equation
      false = cond(x);
      xs_1 = listSelect(xs, cond);
    then xs_1;
  end matchcontinue;
end listSelect;
```

The symbol :: is syntactic sugar for the cons operator. The function goes through the list one element at the time and if the condition is true the element is put on a new list and otherwise it is discarded. Another example of pattern matching with lists is given below. The function listThread takes two lists (of the same type) and interleaves them together.

```
function listThread
  input list<Type_a> inTypeALst1;
  input list<Type_a> inTypeALst2;
  output list<Type_a> outTypeALst;
  replaceable type Type_a subtypeof Any;
algorithm
  outTypeALst:=
  matchcontinue (inTypeALst1,inTypeALst2)
    local
      list<Type_a> r_1,c,d,ra,rb;
      Type_a fa,fb;
    case ({},{}) then {};
    case ((fa :: ra),(fb :: rb))
      equation
        r_1 = listThread(ra, rb);
        c = (fb :: r_1);
        d = (fa :: c);
      then d;
  end matchcontinue;
end listThread;
```

Yet another application for pattern matching might be to pattern match over class hierarchies. Modelica is an object-oriented language and as mentioned in the introduction and in [16] pattern matching is a powerful way to explore a hierarchy of classes. Thus we would like to be able to write something like this:

```
record Expression
```

```
  ...
end Expression;

// Defining new expressions
record NUM
  extends Expression;
  Integer value;
end NUM;

record VAR
  extends Expression;
  Integer value;
end VAR;

record MUL
  extends Expression;
  Expression left;
  Expression right;
end MUL;

matchcontinue(inExp)
  case (NUM(x)) ...
  case (VAR(x)) ...
  case (MUL(x1,x2)) ...
end matchcontinue;
```

Here we could use the fact that `MUL` extends `Expression` when we do the pattern matching and in the static type checking. However, there are difficulties with this approach. Modelica features a structural type system thus type-equivalence is done over the structure of a record and not the name. In the implementation of pattern matching over union types, both in the implementation described in this paper (see Section 5.4) and in the RML-implementation of MetaModelica, we have disregarded these type rules.

Also, decomposition of records can be done via the dot-notation. No further implementation work has been done on pattern matching over record hierarchies.

3.1 Generating Uncertainty Equations

One application of pattern matching is the generation of models with uncertainty equations [18] from ordinary models. This has applications in sensitivity analysis.

4 Discussion on type systems

Modelica features a structural type system [5]. Another class of type systems is nominative type systems. In a structural type system two types are equal if they have the same structure and in a nominative type system this is determined by explicit declarations or the name of the types. Consider the following two records:

```
record REC1
  Integer int1, int2;
end REC1;

record REC2
  Integer int1, int2;
end REC2;
```

In a structural type system these two types would be considered equal since they have the same components. In a nominative type system, however, they would not be equal since they do not have the same names. Also in a nominal type system a type is a subtype of another type only if it is explicitly declared to be so (nominal subtyping). Consider the following three records.

```
record A
  Integer B, C;
end A;

record E1
  Integer B, C, D;
end E1;

record E2
  extends A;
  Integer D;
end E2;
```

In a structural type system record E1 would be a subtype of record A while in a nominative type system this would not be the case. Record E2, however, would be considered to be a subtype of record A in a nominative type system since an inheritance relation is explicitly declared. Java is an example of a language that uses nominative typing while C, C++ and C# use both nominative and structural (sub)-typing. [19]

5 Pattern Matching Implementation

Since a pattern matching expression may contain complex nested patterns and partial overlaps between cases it should be compiled into a simpler, less complex form. Thus, a pattern matching expression is compiled into intermediate form (typically if-elseif-else nodes).

5.1 Overview

The pattern matching construct has been implemented in OMC using an algorithm described in [6]. Here a pattern is viewed as an alternation and repetition-free regular expression over atomic values, constructor names and wildcards. The algorithm first transforms a matchcontinue expression into a Deterministic Finite Automata (DFA) with subpatterns on the arcs. This DFA is then transformed into if-elseif-else nodes. The main goal of the algorithm is to unify overlapping patterns into common branches in the DFA in order to reduce code replication. This algorithm will also try to construct branches to already existing states in order to reduce code replication further. The end result is no nested patterns and no overlap between different if-cases.

The algorithm is composed of four steps: Preprocessing, Generating the DFA, Merging of equivalent states and Generating Intermediate Code. The preprocessing step takes all the match rules and produces a matrix of (preprocessed) patterns and a vector of final states (one for each row of patterns). In the next step the DFA is generated from the matrix and the vector of final states. In the following step equivalent states are merged and finally, in the last step, the intermediate code is generated.

We give a small example to illustrate the intuitive idea behind the algorithm (we use RML [6] style syntax).

```
case xs
   of C(1)   => A1
    | C(2)   => A2
    | C2()   => A3
```

The corresponding matrix and right-hand side vector:

```
| xs=C(ys=1) |    | A1 |
| xs=C(ys=2) |,   | A2 |
| xs=C2()    |    | A3 |
```

We select the first column (the only column). The constructor C matches the first two cases and the constructor C2 matches the last case. Since C2() does not contain any subpattern we are done on this "branch" and we reach the final state. We must continue to match on C's subpatterns, however, and we introduce a new variable ys. The variable ys is a pattern-variable, such a variable will be introduced for every sub-pattern.

```
case xs
  of C(ys)   => …
   | C2()    => A3
```

The rest of the matrix and vector:

```
| ys=1 |    | A1 |
| ys=2 |,   | A2 |
```

We match the rest of the matrix and vector and we get the result:

```
case xs
    of C(ys) =>
       ( case ys
              of 1 => A1
              |  2 => A2)
     | C2()       => A3
```

Note that in the real algorithm a DFA would first be created (with a state for each case and right-hand side and arcs for C, C2, '1' and '2'). This DFA would then be transformed into simple-cases.

5.2 OMC implementation

The specific OpenModelica translation path for Modelica code with matchcontinue constructs is presented in Figure 1. The matchcontinue expression has been added to the abstract syntax, `Absyn`. The pattern matching algorithm is invoked on the matchcontinue expression in the `Inst` module. The main function of the pattern matching algorithm is `PatternM.matchMain` which is given in a light version below.

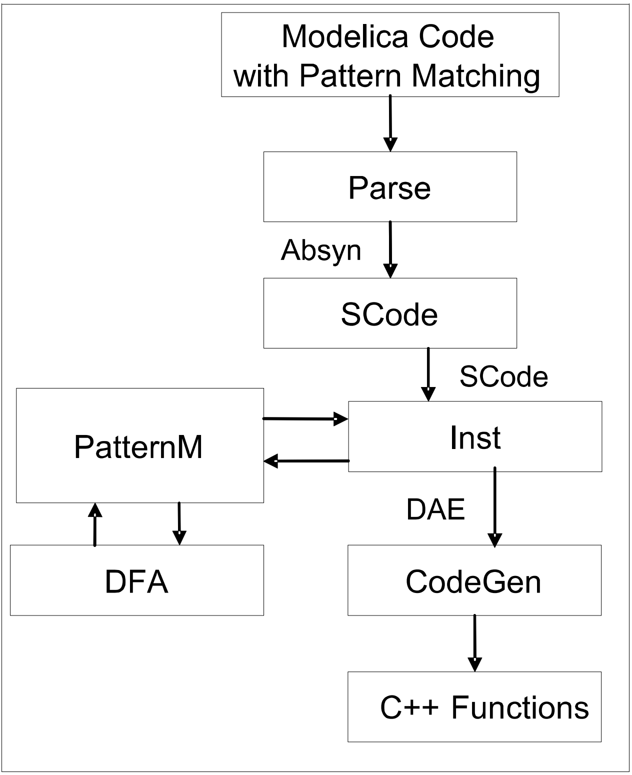


Figure 1. Pattern Matching Translation Strategy.

```
function matchMain
  input Absyn.Exp matchCont;
  input Env.Cache cache;
  input Env.Env env;
  output Env.Cache outCache;
  output Absyn.Exp outExpr;
algorithm
   (outCache,outExpr) :=
   matchcontinue (matchCont,cache,env)
   ...
case(localMatchCont,localCache,localEnv)
  local
  ...
  equation
    (localCache,...,rhList,patMat,...) =
      ASTtoMatrixForm(localMatchCont,
      localCache,localEnv);
    ...
      (startState,...)=matchFunc
     (patMat,rhList,STATE(...));
    ...
    dfaRec=DFA.DFArec(...,startState,
    ...);
    ...
    (localCache,expr) =
      DFA.fromDFAtoIfNodes(dfaRec,...,
      localCache,localEnv,...);
  then (localCache,expr);
end matchcontinue;
end matchMain;
```

The first function to be called in `matchMain` is `AST-ToMatrixForm` which creates a matrix out of the patterns as well as a list of right-hand sides (the code in a case clause except the actual pattern). This corresponds to step 1 of the above described algorithm. The list of right-hand side will actually only contain identifiers, and not all code in a right-hand side, so that the match algorithm won't have to pass along a lot of extra code. The code in the right-hand sides is saved in another list and is later added.

After this the function `matchFunc` is called with the matrix of patterns, the right-hand side list and a start state. This function will single out a column, create a branch and a new state for all matching patterns in the column and then call itself recursively on each new state and a modified version of the matrix, as described in [6]. The function (roughly) distinguishes between three cases:

- All of the patterns in the uppermost matrix row are wildcards.
- All of the patterns in the uppermost matrix row are wildcards or constants.
- At least one of the patterns in the uppermost matrix row is a constructor call.

However, due to the fail semantics of a matchcontinue expression we cannot simply discard all cases below a row with only wildcards as is done in [6]. This is due to the fact that a case-clause with only wildcards may fail and then an attempt to match the subsequent case-clause should be carried out.

Finally, the created DFA is transformed into if-else-elseif nodes (intermediate code) in the function `fromD-FAtoIfNodes`. This corresponds to step 3 and 4 of the algorithm described above. The pattern matching algorithm returns a value block expression containing the if-else-elseif nodes (see section 5.2). C++ code is then generated for the value block expression in the Codegen module.

5.3 Example

We first give an example of the compilation of a matchcontinue expression over simply types. In the next section we discuss the compilation of pattern matching over more complex types (union types, lists, etc.).

```
function func
  input Integer i1;
  input Integer i2;
  output Integer x1;
algorithm
  x1 :=
  matchcontinue (i1,i2)
    local
      Integer x;
    case (x as 1,2)
      local
      equation
        false = (x == 1);
      then 1;
    case (_,_) then 5;
  end matchcontinue;
end func;
```

The code above is first compiled into intermediate form as seen in figure 1. The following C++-code is then generated from the intermediate code (note that the code is somewhat simplified):

```
{
 modelica_integer x;
 modelica_integer LASTRIGHTHANDSIDE__;
 integer_array BOOLVAR__; /* [2] */
 alloc_integer_array(&BOOLVAR__, 2, 1, 1);
  while (1) {
    try {
        state1:
        if ((i1 == 1) && (BOOLVAR__[1]
                || BOOLVAR__[2])) {
          state2:
          if ((i2 == 2) && BOOLVAR__[1]) {
            goto finalstate1;
          }
          else {
              state3:
              if (BOOLVAR__[2]) {
                goto finalstate2;
              }
          }
        }
        else {
            goto state3;
        }
        break;
        finalstate1:
        LASTRIGHTHANDSIDE__ = 1;
        x = i1;
        if (x == 1) {
          throw 1;
        }
        x1 = 1;
        break;
        finalstate2:
        LASTRIGHTHANDSIDE__ = 2;
        x1 = 5;
        break;
        }
        catch(int i) {
        BOOLVAR__[LASTRIGHTHANDSIDE__]=0;
        }
      }
 }
```

Each state label corresponds to a state in the DFA (which was the intermediate result of the pattern matching algorithm) and each if-case corresponds to a branch. See figure 2 for the generated DFA.

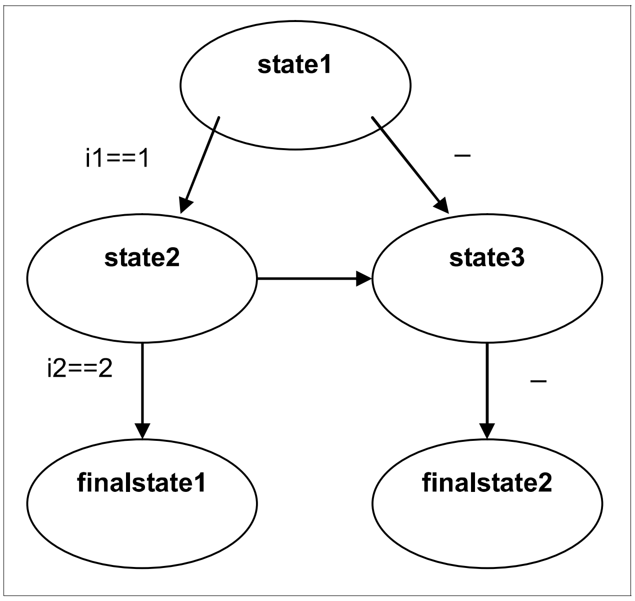


Figure 2. Code Example Generated DFA.

Note that if a case-clause fails then the next case-clause will be matched, since we have a matchcontinue expression. There is an array `(BOOLVAR__)` with an entry for each final state in the DFA. If a fail occurs an exception will be thrown and the catch-clause at the bottom will be executed. The catch-clause will set the array entry of the case-clause that failed to zero so that when the pattern matching algorithm restarts (notice the while(1) loop) this case-clause will not be entered again.

5.4 Pattern matching over union types, lists, tuples and option types

The remaining MetaModelica constructs (that are lacking from Modelica) are currently being added to OMC: lists, union types, option types and tuples. Compilation of MetaModelica code should only be performed if a special compile flag is set. Lists and union types were introduced in section 3. An option type can be seen as a union type with two records, SOME and NONE, where SOME may contain data and NONE is an empty record. A tuple type can be viewed as an unnamed record. See [4] for more details about MetaModelica.

We briefly discuss pattern matching over variables holding these types. Consider first an example with union types given below.

```
uniontype UT
record REC1
Integer field1;
Integer field2;
end REC1;

record REC2 ... end REC2;
end UT;
```

```
matchcontinue (x)
case (REC1(1,2)) ...
case (REC1(1,_)) ...
  ...
end matchcontinue;
```

The example above will result in the following intermediate code.

```
if (getHeaderNum(x) == 0) then
   Integer $x1 = getVal(x,1);
   Integer $x2 = getVal(x,2);
   if ($x1 == 1) then
      if ($x2 == 2) then
      ...
      elseif (true) then
      ...
      end if;
   end if;
elseif (...)
 ...
end if;
```

Note that static type checking is performed by the compiler to make sure that REC1 is a member of the type of variable x and that it contains two integer fields etc.. Union types are represented as boxed-values, with a header and subsequent fields, in C++. Each record in a union type is represented by a number (an enumeration). Since REC1 is the first record in the union type it is represented by 0. The function getHeaderNum is a builtin function that retrieves the header of variable x. The function getVal is also a builtin function that retrieves a data field (given by an offset) from the variable x.

Lists are compiled in a similar fashion.

```
matchcontinue (x)
case (1 :: var) ...
...
end matchcontinue;

=>

if (true) then
Integer $x1 = car(x,1);
list<Integer> $x2 = cdr(x);
      if ($x1 == 1) then
      ...
      elseif (...)
      ...
      end if;
end if;
```

The symbol :: is the cons constructor. The functions car and cdr are builtin functions for fetching the car and cdr parts of a list. Lists are also implemented as boxed values in the generated C++ code so this can be done in a straightforward way. An example of pattern matching over tuples is given below.

```
matchcontinue (x)
case ((5,false)) ...
case ((5,true)) ...
```

```
...
end matchcontinue;

=>

if (true) then
Integer $x1 = getVal(x,1);
Boolean $x2 = getVal(x,2);
      if ($x1 == 5) then
      ...
      elseif (...)
      ...
      end if;
end if;
```

Tuples are, just as union types and lists, implemented as boxed values in C++. The builtin function getVal takes an index and offsets into a boxed value in order to obtain the correct field.

Finally, option types are dealt with in a similar manner as union types.

Note that the reason why we need a run-time type check of union types is that a union type variable may hold any of several record types, which one can only be determined at run-time. When it comes to lists and tuples only one type can exist in a matchcontinue column, if this is violated it will be detected by the static type checker leading to a compile-time error.

5.5 Value block Expression

The value block expression allows equations and algorithm statements to be nested within another equation or algorithm statement. A value block expression contains a declaration part, a statements or equations part and a return expression. The return value of the value block is the value of the evaluated return expression. A value block has been added to OMC mainly because of its use as an intermediate data structure for the pattern matching expression.

6 Conclusions

We have briefly presented the design and implementation of pattern matching for Modelica. We believe that adding this language feature to Modelica will result in a more powerful and complete language. Pattern matching is useful for traversing hierarchies of components, for writing functional-style programs, traversing lists, etc.. Pattern matching would be most useful if MetaModelica constructs such as lists and union type were merged with Modelica. If these constructs were to be included in Modelica, however, several type-related issues must be dealt with. Another possibility is to be able to pattern match over record-hierarchies (as shown in one of the examples). However, decomposition of records can already be done in a straightforward way through the dot-notation.

7 Acknowledgements

This work has been supported by the Swedish Foundation for Strategic Research (SSF), in the RISE and VISIMOD projects, and by Vinnova in the Safe and Secure Modeling and Simulation project.

References

[1] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. Simulation News Europe, 44/45, Dec 2005. http://www.ida.liu.se/projects/OpenModelica

[2] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., Wiley-IEEE Press, 2004. See also: http://www.mathcore.com/drmodelica/

[3] Paul Hudak. The Haskell School of Expression. Cambridge University Press, 2000.

[4] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005.

[5] The Modelica Association. The Modelica Language Specification Version 3.0, September 2007. http://www.modelica.org.

[6] Mikael Pettersson. Compiling Natural Semantics. PhD thesis, Linköping Studies in Science and Technology, 1995.

[7] Christen E. and K. Bakalar. VHDL-AMS-a hardware description language for analog and-mixed-signal applications, In 36th Design Automation Conference, June 1999

[8] Oh Min and C.C. Pantelides (1996) "A Modeling and Simulation Language for Combined Lumped and Distributed Parameter System." Computers & Chemical Engineering, vol 20: 6-7. pp. 611-633 1996.

[9] Xavier Leroy et al., The Objective Caml system. Documentation and user's manual, 2007, http://caml.inria.fr/pub/docs/manual-ocaml

[10] Robin Milner, Mads Tofte, Robert Harper and David MacQueen, The Definition of Standard

ML, Revised Edition, MIT University Press, May 1997, ISBN: 0-262-63181-4

[11] Peter Fritzson. Modelica Meta-Programming and Symbolic Transformations, MetaModelica Programming guide, Version June 2007

[12] P.E. Moreau, C. Ringeissen, M. Vittek: A Pattern Matching Compiler for Multiple TargetLanguages. In: In Proc. of Compiler Construction (CC), volume 2622 of LNCS. (2003) 61–76

[13] M. Oderksy, P. Wadler: Pizza into Java: Translating theory into practice. In: Proc. of Principles of Programming Languages (POPL). (1997)

[14] M. Zenger, M. Odersky: Extensible Algebraic Datatypes with Defaults. In: Proc. of Int. Conference on Functional Programming (ICFP). (2001)

[15] J. Liu, A.C. Myers: JMatch: Iterable Abstract Pattern Matching for Java. In: Proc. of the 5th Int. Symposium on Practical Aspects of Declarative Languages (PADL). (2003) 110–127

[16] Burak Emir, Martin Odersky, and John Williams, Matching Objects With Patterns, LAMP-REPORT-2006-006,EPFL, Lausanne, Switzerland, Language Computer Corporation, Richardson Texas

[17] Martin Odersky, Raising Your Abstraction: In Defense of Pattern Matching, June 29, 2006. http://www.artima.com/weblogs/viewpost.jsp?thread=166742 [Last referenced on January 19, 2008]

[18] European standard ENV 13005.

[19] Benjamin C. Pierce, Types and Programming Languages. The MIT Press Massachusetts Institute of Technology Cambridge, Massachusetts 02142 http://mitpress.mit.edu ISBN 0-262-16209-1

Patterns and Anti-Patterns in Modelica

Dr. Michael M Tiller
Emmeskay, Inc.
Plymouth, MI, USA
mtiller@emmeskay.com

Abstract

In 1977, Christopher Alexander, Sara Ishikawa and Murray Silverstein published the book "A Pattern Language: Towns, Buildings, Construction" [1]. Although the topic of the book was architecture, it inspired Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides in their approach to pattern based software development. This ultimately led to the publication, in 1994, of the book "Design Patterns: Elements of Reusable Object-Oriented Software" [2] (also known as the "Gang of Four" or "GoF" book) which launched a major movement in the software development community toward pattern based software design. The idea behind the pattern movement is to formally identify sound design solutions to common problems.

Since the publication of "Design Patterns" there have been numerous books published on the topic of software patterns. Several of these books dealt with the sub-topic of anti-patterns [3,4]. In contrast to a normal pattern, anti-patterns are an attempt to identify common bad practices and ways they can be refactored using sound design patterns.

The emphasis of the pattern community is, understandably, on object-oriented languages with procedural semantics. This paper will build on previous work [5] identifying patterns in Modelica. These design patterns include how medium properties can be handled in a flexible way, how to deal with systems with varying causality and differential index, idealized plant control and, finally, coordination between models. In addition, this paper includes some extensive discussion of anti-patterns to avoid redundant code, awkward data management and inflexible models.

This paper continues the discussion on patterns within the Modelica community with the hope that this will encourage others to contribute patterns of their own. One obvious benefit of such efforts will be additional resources for Modelica developers to make the process of developing models in Modelica easier. In addition, we expect that many of the patterns discussed will also generate proposals for improving the Modelica language through new features and semantics.

Keywords: patterns, anti-patterns

1 Background

When Alexander *et. al.*, published their work, each pattern included four principle aspects, the pattern name, the context in which the pattern applied, the problem the pattern attempted to address and the proposed solution. This paper will focus primarily on the problem and solution.

In September of 2006, Mark Dominus wrote an essay in his blog [6] in which he concluded with the following statement:

> *"Patterns are signs of weakness in programming languages. When we identify and document one, that should not be the end of the story. Rather, we should have the long-term goal of trying to understand how to improve the language so that the pattern becomes invisible or unnecessary."*

This statement triggered quite a bit of controversy and many people argued with this assertion, not the least of which was Ralph Johnson [7], co-author of the original "Design Patterns" book who argued that patterns are simply manifestations of high level concepts beyond the scope of language semantics.

In this paper the assumption will be that the truth lies somewhere in between. Some patterns are simply manifestations of design decisions made in the development of a given language. Other patterns appear to address missing expressiveness in the underlying language. In some cases, patterns are simply introduced to encourage consistency and readability above and beyond what is really the purview of language designers. Along with the patterns themselves some discussion will be included indicating to what degree each pattern (or anti-pattern, as the case may be) could be mitigated by changes in the language or standard library.

2 Design Patterns

2.1 Architecture Pattern

2.1.1 Problem

While building models to support a variety of systems and/or subsystems a large collection of models with many structural similarities have been developed. Adding additional models involves constructing models either by copy and pasting large chunks from previous models or dragging and dropping the complete model from scratch.

There are two distinct issues being discussed. The first is the amount of work required to create a new model. The second is about redundancy between models. This pattern focuses on the former and the latter is discussed as part of the DRY anti-pattern in Section 3.1.

2.1.2 Solution

When significant structural similarities exist between system or subsystem models then these models can be formulated in terms of architectures. In doing so, each model becomes simply a variation of the architecture with the various interfaces replaced by implementations that are appropriate for that specific model.

For example, building vehicle models by simply dragging all the usual constituents (*e.g.* engine, transmission, chassis, *etc*) into a diagram can be quite time consuming and tedious whereas building them as variations from a standard vehicle architecture (e.g. [8]) can greatly reduce the overhead of creating and managing such models.

2.2 Singleton Pattern

2.2.1 Problem

When building libraries of models it is sometimes necessary to design the library in such a way that there is a single instance somewhere that includes a definitive reference for some information. The basic idea is that within some scope there is exactly one such instance. The challenge is not simply how to access that "singleton" object but how to design the library so that this is handled well for users.

2.2.2 Solution

In languages like Java and C++, the use of the `static` qualifier on members provides a language supported mechanism for ensuring uniqueness within a given program. The closest equivalent in Modelica would be a variable declared as `constant`. However, the values of constants cannot be changed so while semantically similar, this is not adequate to achieve the singleton pattern. Instead, the use of `inner` and `outer` qualifiers is a more common choice. By referring to an `inner` instance it is possible for all outer references to act simply as "pointers" to a single object. The use of `inner` and `outer` has an additional advantage (or disadvantage, depending on how strict you need to be) which is that they can be nested inside each other.

Two immediate examples of the singleton pattern can be found in the Modelica Standard Library. The first is in the Multibody library. The design of the library is such that it requires that there is exactly one instance of the so-called "world" object in the system to provide a reference coordinate system. Another example, which exploits the ability to nest one subsystem (requiring its own internally unique singleton) inside another subsystem, can be seen in the StateGraph library [9].

An example where the use of `inner` and `outer` is not currently sufficient is in dealing with "many to many" interactions. For example, consider a model of the solar system. Each planet exerts a gravitational force on all the others. While it is possible to implement each gravitational force as an individual component that connects between every combination of planet instances in a system, it is more convenient and scalable to have some kind of (singleton) intermediary component that is somehow aware of all planet instances and can, within the context of that single model, handle all interactions. Similar "many to many" requirements can be found in systems where collisions are possible between multiple bodies.

2.3 Medium Model Pattern

The medium model pattern is more generally called the "abstract factory" or "kit" pattern. However in Modelica the most common use is to represent medium properties. For this reason the name "medium model" is used since it is more familiar to the target audience of this paper having appeared in previous work [10, 11].

2.3.1 Problem

In a nutshell, the medium model pattern shows up in models that include multiple configurable types that must be, in some way, consistent with each other. As already mentioned, this is something that occurs often when characterizing the medium of a given fluid system. The configurable types typically include (but are not limited to) connector definitions and some kind of property evaluation model. The essential point is that many assumptions about a fluid bind the definition of the connectors and the property evaluation together (*e.g.* the number of species). For example, it would not make sense to combine the connector a multi-species gas with the properties of oil.

2.3.2 Solution

As mentioned previously, this approach is called the “abstract factory” pattern in other languages and it is usually achieved through abstract methods that return instances abstract types. The consistency is assured by the implementation of the abstract factory. Because Modelica lacks methods or even any appreciably dynamic object creation, the same effect is achieved in Modelica using replaceable packages.

By using replaceable packages, it is possible for models to reference constants and types defined in the “constraining package” defined or implied in the replaceable definition. A given “implementation” (*e.g.* a specific medium) can then redefine these types and constants in a consistent way (*e.g.* so they all represent the same medium). The following sample code demonstrates the use of this pattern. First, an abstract model of the medium must be defined:

```
partial model AbstractMedium
  constant Integer n “# of Species”;
  connector Fluid
    Pressure p;
    flow MassFlowRate m_dot;
    MassFraction Xi[n-1];
    flow MassFlowRate mXi_dot[n-1];
  end Fluid;

  partial block Properties
    input Pressure p;
    input MassFraction Xi[n-1];
    output SpecificEnergy u;
    output SpecificEnthalpy h;
  end Properties;
end AbstractMedium;
```

Based on this abstract medium model, component models can then be written that rely on information from the medium model but without knowledge of what specific medium model is being used:

```
model Component
  replaceable package MediumModel =
    AbstractMedium;
  MediumModel.Fluid c;
  MediumModel.Properties props(
    p=c.p,X=c.X);
equation
  // equations in this component
  // can reference the pressure
  // at the connector, c.p, or
  // properties of the fluid,
  // e.g. props.h
end Component;
```

Finally, an implementation of the medium model can be created by extending from the abstract medium model:

```
package RealMedium
  extends AbstractMedium(nspecies=2);
  redeclare model extends Properties
  equation
    // This model may include things
    // like property calculations or
    // an equation of state.
  end Properties;
end RealMedium;
```

One usability issue with this pattern is that when it is used in conjunction with the transport of physical information or behavior it is somewhat counter intuitive since the redefinitions of the medium model are propagated from “top down” when users think, at least conceptually, that the information should be propagated through connections. For example, the Component model in the previous sample code would need to be instantiated with a modification specifying the medium model, e.g.

```
Component comp(
  redeclare package MediumModel =
   RealMedium);
```

whereas most users would expect that “somehow“ the type of medium was dictated by what the instace was connected to. While this is not an issue with the pattern in general, it is an important consideration for language designers and tool vendors.

2.4 Adapter Pattern

2.4.1 Problem

When working with architectures, it is necessary for the subsystem models to be developed so that they satisfy the interface prescribed by the architecture. However, there are many cases where the subsystem model might be developed independently from an architecture and as a result it does not conform to any specific interface. This situation may come about because the subsystem models were developed before the architecture or perhaps they were developed in an architecturally neutral way to avoid dependence on a particular architecture or to support multiple architectures.

2.4.2 Solution

In these circumstances, it may be necessary to develop adaptor components. Such components provide a mapping from the interface that the subsystem currently has to the interface that is to be supported. There are two variations of this pattern. In the first case, the subsystem is developed independently from any particular interface. In this case, the development of an adapter for the subsystem is a “one time only” process since other subsystems are unlikely to share the exact same interface (and if they do, they should probably be refactored as described in Section 3.1).

The other case is where the subsystem has been developed according to a specific interface (one that presumably other subsystems satisfy). In this case, a general adaptor could be constructed that maps one interface onto another. Such an adaptor could then be used as an adaptor for multiple subsystems. This kind of adaptor pattern can also be used to implement compatibility between comparable interfaces across different architectures.

The following code fragment shows an example of how the adaptor pattern is implemented. First, let us consider the one potential (and greatly simplified) interface for a vehicle model:

```
partial model VehicleInterfaceA
  RealOutput vehicle_speed;
end VehicleInterfaceA;
```

Several vehicle models might be developed using this interface, *e.g.*

```
model Vehicle1
 extends VehicleInterfaceA(
   vehicle_speed=…);
end Vehicle1;

model Vehicle2
 extends VehicleInterfaceA(
   vehicle_speed=…);
end Vehicle2;
```

Now consider an alternative vehicle model interface and system architecture definition:

```
partial model VehicleInterfaceB
 RealOutput v_vehicle;
end VehicleInterfaceB;

partial model ArchitectureB
 replaceable VehicleInterfaceB vehicle;
 …
end ArchitectureB;
```

An adaptor between the different interfaces could be developed as follows:

```
model VehicleAdaptor_A2B
 extends VehicleInterfaceB;
 replaceable VehicleInterfaceA vehicle;
equation
  connect(vehicle.vehicle_speed,
          v_vehicle);
end VehicleAdaptor_A2B;
```

Using this adaptor it is possible to build a system that utilizes `ArchitectureB` but uses an implementation of `VehicleInterfaceA` as follows:

```
model System
 extends ArchitectureB(
   redeclare VehicleAdaptor_A2B(
     redeclare Vehicle1 vehicle));
end System;
```

2.5 Parametric Behavior Pattern

2.5.1 Problem

In acausal modeling most components tend to describe the flow of some conserved quantity explicitly in terms of the across variables (*e.g.* i=v*R). Other components describe the flow of conserved quantities implicitly in terms of constraints (*e.g.* an ideal voltage). However, it is often quite useful to be able to describe components that describe the flow of conserved quantities in terms of both implicit and explicit relations depending on the state of the component. The simplest example of such a component is an electrical diode which either allows no current (explicit case) or no voltage drop (implicit case). Another slightly more complicated case would be a clutch which computes transmitted torque explicitly in terms of dynamic friction when disengaged or slipping but computes torque implicitly in terms of a kinematic relation when locked.

2.5.2 Solution

One "easy" way to describe such behavior is to compromise on the ideal nature of the behavior. For example, where an ideal diode might describe the implicit and explicit behavior using the equations v=0 and i=0, respectively, a compromise model sacrifices the idealization might use the equations v=G*i and i=v*R, where G is chosen to be very small (to approximate the v=0 case) and R is chosen to be very large (to approximate the i=0 case). The result of this compromise is that the behavior is now completely explicit in nature. However, another consequence of this "easy" solution is that the system of equations is very likely to be poorly conditioned which means the system will be stiff and slow to simulate.

A "better" solution (from the modeler's perspective at least) is to capture the ideal behavior somehow. Not only is this possible but it can be a very elegant and useful way to approach such problems. The basic premise (which is presented in greater detail in [12]) is to introduce a third variable and describe the behavior of the original variables in terms of the third parametric variable. This approach is frequently used in geometric applications where it is not possible to use a particular coordinate axis as an independent variable to describe a line or surface. The same issue is present, for example, in a diode where it is not possible to write current explicitly in terms of voltage nor is it possible to write voltage explicitly in terms of current. However, it is possible to write both in terms of a third parametric variable, *e.g.*

```
off = s<0;
v = if off then s*unitV else 0;
i = if off then 0 else s*unitC;
```

where unitV and unitC are defined as follows:

```
import Modelica.SIunits.Voltage;
import Modelica.SIunits.Current;
constant Voltage unitV=1;
constant Current unitC=1;
```

Analysis of this parametric approach shows that describing this kind of behavior is not simply an issue with the expressiveness of the underlying modeling language but with the solution method. While some basic solution techniques exist to deal with component models that are either implicit or explicit, the ability of a component to function in both ways creates additional complications for the underlying solver.

One such complication is that switching between two different sets of equations during a simulation always brings with it the risk that the differential index of the system might change. As such, the posed problem could be a variable index system. In fact, a clutch model normally leads to a variable index system when modeled using the parametric behavior pattern. However, by understanding this in advance it is possible to differentiate the equations such that the index is no longer variable. For this reason, it would be very useful if investigation into this issue showed that a general algorithm could be developed along similar lines. Such an algorithm would most likely benefit from language features that directly supported this pattern.

2.6 Perfect Control Pattern

2.6.1 Problem

Physical models typically include sensors and actuators and these are in turn normally connected to some kind of control system. One of the burdens that model developers face is to provide some kind of actuator control strategy in addition to the base physical models. In many cases, the model developer is not particularly interested in the dynamics of the controller but they need some function in the model to determine how the actuator will behave and so therefore implementation of controls is unavoidable. Such implementations often take time both to construct and calibrate and many times they do not add any significant value to the model.

2.6.2 Solution

It is important to point out that this pattern is very specific to cases where the model developer simply wants a very good controller but they don't need to be very concerned about how such a control strategy would actually be deployed or implemented in hardware. In these specific circumstances, it is often possible to rely on a "perfect" control strategy to control the device. For example, consider a simple SISO plant model defined as follows:

```
model PlantModel
  input Real u;
  output Real y;
protected
  Real dy = der(y);
equation
  2*der(dy) + dy + 4*y = u;
end PlantModel;

model ClosedLoop
  PlantModel plant;
protected
  Real ybar = max(0,time-2);
equation
  plant.u = 10*(ybar-plant.y);
end ClosedLoop;

model PerfectControl
  PlantModel plant;
protected
  Real ybar = max(0,time-2);
equation
 ybar = plant.y;
end PerfectControl;
```

The simulation results from both types of control can be seen in Figure 1. The basic idea of this pattern is rather than including an explicit equation for the command to the system an equation prescribing the output is used. This equation for the output acts as an implicit equation for the input. It should be pointed out that this type of approach is limited to cases where the plant model is sufficiently invertible.

Despite this limitation, this is a useful pattern that can be used in conjunction with some surprisingly complex systems. For example, this approach is the same approach that is employed to create "backward" drive cycle models (models where the vehicle speed is prescribed and the system resolves the torque required to meet the speed profile). In addition, this same pattern can be used in conjunction with actuators like clutches and valves.

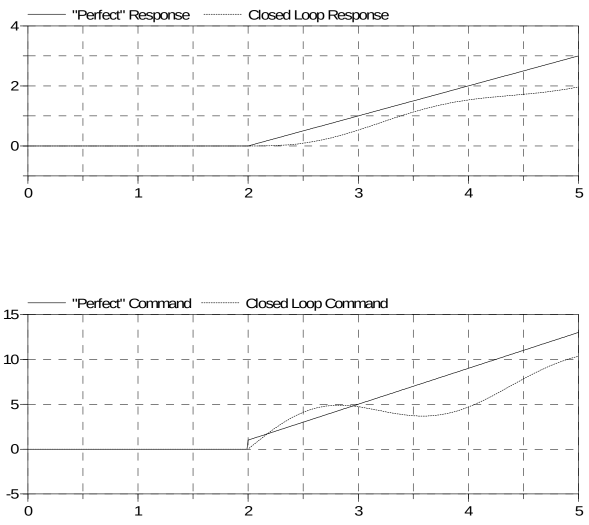


Figure 1: Example of "Perfect" Control Pattern

3 Anti-Patterns

Patterns are primarily useful for intermediate to advanced users who, having written some substantial amounts of code, are able to recognize the emergence of patterns and are interested in understanding how patterns can help them be more productive (as well as improve consistency and readability among project members).

However, Modelica is still a relatively new technology with many new users. As a result, anti-patterns are probably at least as important as patterns. The reason is that anti-patterns can help novices to recognize weaknesses in code they have written. As such, anti-patterns are almost immediately applicable. This section introduces several anti-patterns and discusses refactoring approaches associated with each pattern.

This is not to say that anti-patterns only apply to novice users. Because Modelica improves developer productivity, it is very easy to write a large volume of code only to realize in hindsight that some anti-patterns have developed. As a result, the material in this section is applicable to a wide range of users. As such, the material in the anti-patterns section should be of particular interest to tool vendors since refactoring typically requires tool support.

3.1 DRY Anti-Pattern

3.1.1 Problem

By far, the most common anti-pattern is the use of "copying and pasting" model code between models. While this happens for a wide variety of reasons most of them are ultimately because users are not aware of the various mechanisms within Modelica for code reuse. In software development there is something known as the "DRY principle" where DRY is an acronym for "Don't Repeat Yourself". The DRY anti-pattern is one where the DRY principle has not been followed.

The reason that the DRY principle is so important (and which has led to the motto that "redundancy is the root of all evil") is that redundancy creates many problems. Not only does it lead to inefficiency when building models it also means significantly more work when maintaining those same models.

3.1.2 Solution

While this is a very common anti-pattern, the good news is that Modelica contains a rich supply of language features to help combat it. The first language feature all users should become familiar with is inheritance (specifically, the `extends` keyword). Once developers understand inheritance they should investigate the architecture pattern (described previously in this paper) which hinges on the `replaceable` and `redeclare` keywords.

One issue that prevents addressing this anti-pattern is tool support for refactoring. This manifests itself in several ways. First, it should be possible for users to change the names of components and/or classes and be assured that all references that use those names are also adjusted (ideally even if they are not even currently loaded). Furthermore, refactoring of existing code often involves the exercises of identifying commonality between existing models, composing base classes that contain this common code and then extending the original models from the base classes. Without tool support, such refactoring can be very time consuming.

3.2 Kitchen Sink Anti-Pattern

3.2.1 Problem

Another common anti-pattern is the "kitchen sink" anti-pattern. There are two variations of this pattern. For component models, the anti-pattern manifests itself as component models with too many equations by lumping several distinct types of behavior together into a single component. For subsystem models, the anti-pattern manifests itself in diagrams with an unnecessarily large number of components.

3.2.2 Solution

In both of these cases, a "divide and conquer" approach is required. For the component variation, this means building component models that heed Occam's Razor, "*entia non sunt multiplicanda praeter necessitatem*". In practical terms, this means building component models that attempt as much as possible to describe individual effects (*e.g.* inertia, compliance, dissipation, *etc*).

In the case of subsystem models, refactoring is typically a matter of nesting some tightly coupled subset of components into a subsystem of their own. Again, tool support is an issue here. Simulink has a very convenient feature to take a group of selected components and lump them into a subsystem model. Modelica tool vendors would do well to recognize the value of such functionality (and users would do well to remind them).

3.3 Literal Data Overload Anti-Pattern

3.3.1 Problem

Modelica supports a wide range of ways to deal with data handling. In theory, users can bring data in from an external database, they could read it from external files, *etc*. However, the simplest way to import data into Modelica models is to enter it literally (*e.g.* `parameter Real table[:,2] = [0, 1; 1, 2; 2, 3; …]`). While there is nothing wrong with this *per se*, it leads very quickly to the literal data overload anti-pattern. The pattern is characterized by the tendency of models to rely on literal data. While this is acceptable for simple component models, this creates two problems with more complex models. The first complication is that entering tables of data is often quite inconvenient. The second complication is that often times any given parameter cannot be changed independently. For example data associated with a given electric motor might bring together the rotor inertia, internal resistance, bearing friction, etc into a set of parameters. If a different motor is to be used, it is not simply a matter of changing a single parameter value but the entire set must be exchanged for another consistent set representing a different motor.

3.3.2 Solution

Both issues of entering literal data and parameter set consistency can be handled by creating records to represent such parameter sets and including the literal data only in the context of the record definitions. In addition, it is advisable to make use of the `choices` annotation so tools understand how the data will be used. The result of such refactoring is that users will only see opaque references to complex and/or voluminous data sets rather than vast expressions containing literal data. It is also a advisable to provide useful descriptions of the data sets so tools can provide users with clear descriptions of available choices.

3.4 Parameter Data Overload Anti-Pattern

3.4.1 Problem

The previous anti-pattern addresses some of the issues associated with models that require large amounts of data. While the aggregation prescribed for refactoring reduces the number of individual parameters a complex system with many components can still contain large numbers of parameter sets (and even the aggregations themselves may have an unwieldy number of parameters). The result is parameter dialogs that contain large numbers of parameter values and/or choices. In these cases, further consolidation doesn't make sense (since we do not want to aggregate data together that is actually independent or unrelated) as a way to address the overload.

3.4.2 Solution

In cases where aggregation is not an appropriate remedy the standard annotations for grouping parameters by tab and group can be utilized. Rather than aggregate the data, the result of using the tab and group directives is to organize the data into a "tree" (i.e. the data is presented in a hierarchy where the first layer is determined by the tab and the next layers is determined by group). In particular, common parameters should be organized such that they appear in the default tab and less common parameters are assigned to later tabs. Tab labels are also an important consideration since users should be able to determine quickly, based on the name, whether they need to look in a particular tab.

4 Language Implications

Many of the "normal" patterns found in [2] do not appear in this paper. This is primarily because Modelica does not include concepts like pointers and methods which are fundamental to many of the patterns. Furthermore, it has been observed that many of the traditional patterns in software development essentially boil down to adding an additional level of indirection to an abstraction. Since there are very few ways to express this indirection in Modelica, the number of patterns is fairly limited.

One of the lingering questions from this discussion is to what extent these patterns (or lack of patterns) represent deficiencies in the language. For the patterns and anti-patterns that are related to redundant code (*i.e.* Sections 2.1, 3.1 and 3.2) the language is well equipped to address these issues although there are certainly ways that tools can assist model developers in more effectively utilizing those language features.

Although the Singleton pattern is being used in several libraries it is this author's opinion that the semantics of the language do not mesh as well with the pattern and modeler needs. The use of inner and outer in this way has implications for robust model checking and the dependency on inner elements is not easily recognized or represented. In addition, the "many to many" issue mentioned in Section 2.2.2 requires improved expressiveness in the language.

In the case of the medium model pattern, the inability to express type constraints through physical connections is a serious limitation in the language and one that is recognized in the design group. Hopefully this deficiency will be addressed soon.

Section 2.5.2 discusses how behavior can be described parametrically. However, there are many different ways to "phrase" this kind of behavior and they cannot necessarily be easily recognized by tools. Having language elements for describing parametric relationships could not only bring consistency how such behavior is described but it could also allow tools to automatically deal with variable index issues that currently burden developers (equation differentiation, continuity concerns, finite state machines, *etc*).

5 Conclusion

The goal of this paper is to identify common patterns and anti-patterns to help users identify easy solutions for common problems as well as to prompt discussions within the Modelica design group on ways the language can be enhanced to either institutionalize some of the best practices in these patterns or add language features to eliminate the need for these patterns.

References

1. Alexander, C., Ishikawa, S., and Silverstein, M., "A Pattern Language: Towns, Buildings, Construction", Oxford University Press, ISBN 0-19-501919-9, 1977.
2. Gamma, E., Helm, R., Johnson, R. and Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, ISBN 0-201-63361-2, 1994.

3. Brown, J. W., Malveau, R. C. and Mowbray, T. J., “AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis”, John Wiley and Sons, ISBN 0-471-19713-0, 1998.
4. Laplante, P. A., Neill, C. J., “Antipatterns: Identification, Refactoring, and Management”, CRC Press, ISBN 0-8493-2994-9, 2006
5. Clauss, C., Leitner, T., Schneider, A. and Schwarz, P., “Object-oriented Modelling of Physical Systems with Modelica using Design Patterns”, Fraunhofer Institute, 2000
6. Dominus, M., “Design Patterns of 1972”, http://blog.plover.com/prog/design-patterns.html
7. Johnson, R., “Design patterns and language design”, http://www.cincomsmalltalk.com/userblogs/ralph/blogView?entry=3335803396
8. Tiller, M., Bowles, P. and Dempsey, M., “Development of a Vehicle Modeling Architecture in Modelica”, 3rd International Modelica Conference, 2003.
9. Otter, M., °Arz´en, K.-E., Dressler I., “StateGraph-A Modelica Library for Hierarchical State Machines”, 4th International Modelica Conference, 2005.
10. Newman, C. E., Batteh, J. J., Tiller, M., ”Spark-Ignited Engine Cycle Simulation in Modelica”, 2nd International Modelica Conference, 2002. http://www.modelica.org/events/Conference2002/papers/p17_Newman.pdf
11. Elmqvist H., Tummescheit, H., Otter, M., “Object-Oriented Modeling of Thermo-Fluid Systems”, 3rd International Modelica Conference, 2003.
12. Tiller, M. M., “Introduction to Physical Modeling with Modelica”, Kluwer Academic Publishers, ISBN 0-7923-7367-7, 2001.

Comment- and Indentation Preserving Refactoring and Unparsing for Modelica

Peter Fritzson, Adrian Pop, Kristoffer Norling, Mikael Blom
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{petfr, adrpo, x06krino, x06mikbl}@ida.liu.se

Abstract

In this paper we present a strategy for comment- and indentation preserving refactoring and unparsing for Modelica. The approach is general, but is currently being implemented for Modelica in the OpenModelica environment. We believe this to be one of the first unparsing approaches that can preserve all user-defined indentation and comment information, as well as fulfilling the principle of minimal replacement at refactorings.

Keywords: Refactoring, comments, unparsing,, Modelica.

1 Introduction

Integrated programming environments, e.g. InterLisp [11] and Eclipse [12] provide various degrees of support for program transformations intended to improve the structure of programs – so-called *refactorings* [5] (see also Section 10).

Such operations typically operate on abstract syntax tree (AST) representations of the program. Therefore the program needs to be converted to tree form by *parsing* before refactoring, and be converted back into text by the process of *unparsing*, also called *pretty printing* This is supported by a number of environments (Section 10).

However, a well-known problem is that of preserving comments and user-defined indentation while performing refactorings. Essentially all current environments either loose the comments (except for special comments that are part of the language syntax and AST representation), or move them to some other place. User-defined indentation is typically lost and replaced by machine-generated standard indentations. This is accepted by some developers, but judged as unacceptable by others. However, if the objective only is to improve indentation, then a semi-automatic indenter can be used instead (Section 8.3).

Currently Modelica-based tools are handling only declaration comments that are part of the model and are discarding or moving all the other comments, i.e. the ones between `/* */` and after `//….` Such behavior is highly undesirable from a user perspective and heavily affects the ease-of-use of code-versioning tools.

A goal for the work presented here is to support Modelica code refactoring with minimal disruption of user-defined comments and indentation. In this paper we present such an approach for unparsing in conjunction with refactorings.

2 Comments and Indentation

Regard the following contrived Modelica example. It has one declaration comment which is part of the language syntax, and two "textual" comments `Itemcomm` and `MyComm` which would be eliminated by a conventional parser. It is also nicely hand formatted so that the start positions of each component name in the text are vertically aligned.

```
record MODIFICATION  "Declaration comment"

  Boolean            finalItem; //Itemcomm
  Each /* MyComm */  eachRef;
  ComponentRef       componentReg;

end MODIFICATION;
```

Assume that this is parsed and unparsed by a conventional (comment-preserving) unparser, putting two blanks between the type and the component name of each component. The manual indentation would be lost, and the "textual" comments would be moved to some standard positions (or be lost):

```
record MODIFICATION "Declaration comment"

  Boolean  finalItem; //Itemcomm
  Each  eachRef; /* MyComm */
  ComponentRef  componentReg;

end MODIFICATION;
```

3 Refactorings

Below we make some general observations and give examples of refactorings.

3.1 The Principle of Minimal Replacement

For a refactoring to have minimal disruption on the existing code, it is desired that it supports the principle of minimal replacement:

- *When replacing a subtree, the minimal subtree that contains the change should be replaced.*

This also has the consequence of minimal loss or change of comments. For example, if a name (an identifier) is changed, only the identifier node in the tree should be replaced, not the surrounding subtree.

3.2 Some Examples of Refactorings

Here we mention a few common refactorings. There are also numerous, more advanced and specialized refactorings.

- *Component name change*. Change name of a component name in a record. For example:

```
record MODIFICATION  "Declaration comment"
  Boolean            finalItem; //Itemcomm
  Each /* MyComm */  eachRef;
  ComponentRef       componentReg;
end MODIFICATION;
```

  The name of the component reference name is currently `componentReg`, which is an error. It should be `componentRef`. We would like to change the name both in the declaration and all its uses, thus avoiding updating all named references by hand, which would be quite tedious.

- *Function name change*. Change the name of a function, both the declaration and all call sites.

- *Add record component*. Add a new component declaration to record. In MetaModelica, that would also mean putting an underscore '_' at the correct position in all patterns for that record type with positional matching.

- *Add function formal parameter*. Add an input or output formal parameter to a function. The question is, how much is possible to do automatically? Adding arguments to recursive calls to the function itself is no great problem, but calls from other functions can be more problematic since meaningful input data needs to be provided. This can be handled easily in those cases a default value can be passed to the function's new formal parameter.

4 Representing Comments and User-Defined Indentation

How should information about comments and user defined indentation be represented in the internal (AST) program representation? There are basically two possibilities for a chunk of code, e.g. a model:

- *Tree*. The AST representation is the main storage (the TRUTH). Comments and indentation as extra nodes/attributes in the AST.
- *Text*. The text representation, including indentation and comments, is the main storage (the TRUTH).

The tree approach may seem natural, since the refactorings and the compiler operate on the tree representation. However, it has some disadvantages:

- Since white space and comments can appear essentially anywhere, between nodes, associated with nodes, the AST will become cluttered and increase the required memory usage and complexity of the tree, perhaps by a factor 2-3.
- The large number of extra nodes in the AST may complicate code accessing and traversing the tree.

Regarding the text representation we make the following observations:

- The text representation exists from the start, since this is the storage form used in the file system. Environments like Eclipse use text buffers for direct interaction with the programmer.
- The text representation includes all indentation and comment information, and is compact.
- The structure of the program in the text representation is not apparent, and cannot be easily manipulated.

Why not combine the advantages of each representation, and try to avoid the disadvantages?

- Use the text representation as the basic storage format including indentation and comment information. The text might be conceptually divided into chunks, where for example each class definition gives rise to a text chunk.
- Use the tree representation for compilation and refactoring. Create it when needed and keep it during the current session. Create it piece-wise, e.g. for one class at a time.
- Create a mapping from the tree representation to the text representation; each node in the tree has a corresponding position and size in the text representation. Create this mapping when needed, for appropriate pieces (e.g. class definitions) of the total model.

5 Implementation

The following strategy is used for the implementation

5.1 Base Program representation

The text representation is the TRUTH, the source, and the AST representation is a secondary representation derived from the source, used during compilation and refactoring.

The class information attribute of a class definition in the AST should be extended, e.g. with the byte start position (directly addressing within a file), or by a text chunk corresponding to the text of a class declaration. A package which contains classes would instead refer to the definitions of those classes.

Text positions and text sizes of each AST node should be indirectly associated with each AST node.

5.2 The Parser

The following special considerations need to be addressed by the parser:

- In order not to clutter the produced AST tree, the parser produces two trees: a standard AST tree, and a positioning tree (produced in parallel) with the same number of nodes, containing text positions and sizes of each subtree.
- The parser should return the start text position and text size of each built AST tree. Moreover, if there are any comments within the AST tree text range, a list of the start positions and sizes of these comments should be associated with the parallel tree node.
- The pure AST tree should be clean and not cluttered with position and comment information.
- As mentioned, a text position tree with the same number of nodes and children as the AST is created in parallel to the AST. The positioning tree is only produced when needed for refactorings or text positioning, and thrown away when not needed.

For example, a child nr 3 of a node at level 2, will find its text positions in the parallel tree in the node at level 2 and child nr 3.

5.3 The Scanner

The text position and size of each token is returned together with the token itself.

5.4 The New Unparser

The new unparser will use a combined strategy as follows, combining existing text with new text generated by the tree unparser:

- If there exist already indented text associated with a node, use this text to produce the unparsing text.
- If there is no existing text, this must be a new tree node produced by the refactoring tool. Call the tree unparser to convert this subtree into text that is inserted into the final unparsing result.

6 Refactoring Process

The following steps are to performed in this order during the actual refactoring:

- Traverse the AST and perform insertion/deletion/replacement of subtrees.
- For each insertion/deletion/replacement operation, put each such an operation descriptor in a list, together with the text position and size of the text of the subtree to be replaced/deleted etc.
- After traversal, sort these operations according to text position, and perform the operations in the text in backwards order (take those at the highest text position first).

7 Example of Function Name Refactoring

The example below is used to illustrate the refactorings and the used combined tree and text chunk representation.

All loaded models (including the `Modelica` package) reside in an un-named top-level scope that we can call `Top`. A model may be a top-level model, but more typically a package which in turn may consist of subpackages:

```
01 within ParentPackage;
02 package pack
03  function addOne "function that adds 1"
04   input Real x = 1.0; // line comment
05   output Real y;      /*  multiple
06                           line
07                           comment */
08  algorithm
09   y := x + 1.0;
10  end addOne;
11
12  class myClass
13    Real y;
14  equation
15    y = addOne(5); // Call to addOne
16  end myClass;
17 end pack;
```

Line numbers are given to help the reader follow the example. The position tree constructed by the parser is given in the appendix as it is quite large. A portion of the abstract syntax tree is also shown in order to understand the example.

A function name refactoring will be applied to the example which will change the name of the function "addOne" to "add1", The refactoring can be performed in the OpenModelica environment by loading the example and calling the interactive API function:

```
loadFileForRefactoring("Example.mo");
refactorFunctionName(pack.addOne, "add1");
```

The compiler will execute the first command by calling the new parser that also builds the position tree together with the AST:

```
(ast,posTree) = Parse.refactorParse(file);
```

The result of the load command is two trees. The second (`posTree`) is the position tree presented (partly) in the appendix. The first (`ast`) is the abstract syntax tree of the loaded file which is presented also in the appendix entirely. Here is just a overview picture of the AST:

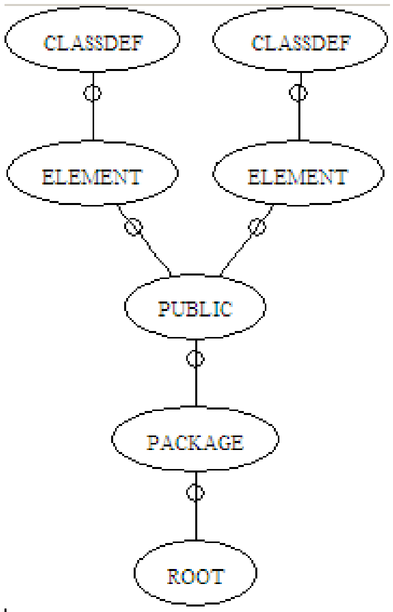


Figure 1. AST of the Example.mo file.

The figure shows that the program has one package with two public elements which are class definitions.

Actually only two refactoring operations are needed to implement any refactoring: add and delete or add and replace.

When `refactorFunctionName` is called the compiler will perform these operations:

7.1 Lookup pack.addOne

Lookup of a class definition is performed by walking the AST while keeping track of a numbered path in the tree. To reach the `addOne` identifier, the path: 1, 6, 1, 1, 1, 5, 2, 1, 1 is applied. The path goes via the following AST nodes: `PROGRAM [1] / CLASS [6] / PARTS [1] / PUBLIC [1] / ELEMENTITEM [1] / ELEMENT [5] / CLASSDEF [2] / CLASS [1] / IDENT("addOne") [1]`.

7.2 Lookup Any Uses of pack.addOne

Lookup of the uses are performed by walking the AST, keeping track of the scope, while keeping track of a numbered path. To reach the function call of `addOne`, the path: 1, 6, 1, 1, 1, 5, 2, 1, 1 is applied. The path goes via the following AST nodes:

`PROGRAM [1] / CLASS [6] / PARTS [1] / PUBLIC [2] / ELEMENTITEM [1] / ELEMENT [5] / CLASSDEF [2] / CLASS [6] / PARTS[1] / EQUATIONS [1] / EQUATIONITEM [1] / EQ_EQUALS [2] / CALL[1] / CREF_IDENT [1] / IDENT("addOne") [1]`.

7.3 Apply the Refactoring to the Actual Text

Now that the paths needed for the minimal refactoring were discovered in the AST, apply these paths to the position tree and fetch the positions of the elements at the end of the paths:

- Function name: `IDENT, Start:047, End:053`
- Function use: `IDENT, Start:313, End:319`

The text operations are applied bottom-up because otherwise the character positions of the elements below an applied operation would change. Ordering of text operations is needed to have them applied in a bottom-up fashion:

- `ReplaceText(file, 319, 313, "add1");`
- `ReplaceText(file, 53, 47, "add1");`
- `Close(file);`
- `(ast, posTree) =  // re-parse the file`
  `Parse.refactorParse(file);`

After the file is closed either a reparsing is performed to load the new AST (as exemplified here) or the refactoring operations are perfomed on the tree already in the memory. Of course the best alternative would be to perform the refactoring during lookup as we have implemented it in the OpenModelica compiler.

As one can notice the comments stay in place so there is minimal disruption to the text representation. This is very valuable from a user point of view but also for code-versioning tools.

7.4 Calculation of the Additional Overhead

There is not too much overhead for the refactoring both with respect to memory usage and time spent walking the tree. In the following table we discuss such overhead and give specific numbers for needed memory size and time complexity of the refactoring procedure.

Memory overhead	**Time overhead**
Space is required for storing the position tree. The size of this space is two integers (of 4 bytes) for each AST node. Also the list of operations to be applied to the text needs memory for storing the paths and the operations themselves, but this memory is negligible compared to the AST and position tree and can also be freed. **Example**: there are about 50 nodes in the example, which means an additional memory of ~ 50NrNodes x 2Positions x 4Bytes = 400Bytes are needed for the position tree. Or course, the position tree could be built on demand and the freed when memory is needed.	Walking two trees while performing the refactoring has a time impact of *NumberOfNodesWalked* x O(1) to walk a node: O(*NrOfNodesWalked*). Walking the position tree while and applying the text operations to the file is negligible compared to the refactoring operation. **Example**: it took about 0.2 seconds to perform the function name refactoring for the example file using the OpenModelica system. Refactoring old graphical annotations of the Modelica Standard Library version 1.6 to the new style graphical annotations took about 9.6 seconds, which is very good for such a demanding refactoring.

8 Unparsers/Prettyprinters versus Indenters

As mentioned previously, an unparser converts an AST program representation into (nicely indented) text. A reformatting indentation tool uses another approach, it operates directly on the text representation to produce a more nicely indented text.

8.1 Pretty printers/Unparser Generators

An unparser generator produces an unparser from a specification, a grammar-like description of unparsing related aspects of the language. A number of systems mentioned in Section 8 support unparsing or generation of unparsers from such specifications.

8.2 OpenModelica Tree Unparser

The current OpenModelica version 1.4 unparser is hand implemented in MetaModelica, recursively traversing the AST while generating the Modelica text representation. It can be invoked by the OpenModelica `list` command. Comments are currently lost (except for declaration comments).

8.3 Reformatting Indentation in the OpenModelica Eclipse Plugin

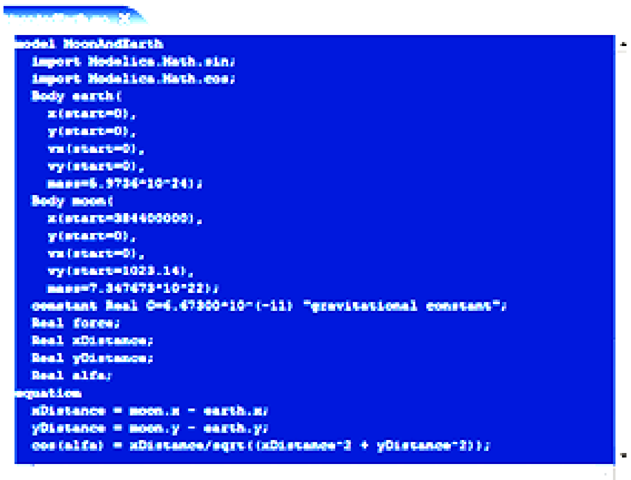
A *text reformatting indentation* tool operates directly on the text representation, and analyzes the text by a combination of scanning and piecemeal heuristic partial parsing to recognize certain combinations of tokens. It inserts or removes white space in order to produce a nice indentation, or improve an existing one. Such mechanisms are typically invoked by the user on a few lines at a time, and are not completely automatic, the user is often required to perform the final adjustments. An advantage with this approach is that comments are not lost.

This kind of indentation tool is for example available for a number of languages in their respective Emacs modes, or as part of Eclipse plugins, e.g. for C++, Java, and more recently for Modelica in the OpenModelica MDT Eclipse plugin.

MDT includes support for automatic indentation, as described here and in [13]. When typing the Return (Enter) key, the next line is indented correctly. The user can also correct indentation of the current line or a range selection using CTRL+I or “Correct Indentation” action on the toolbar or in the Edit menu.

Indentation can be applied to incomplete code as a heuristic Modelica scanner is used and the indentation is based only on the tokens generated by this scanner. The indenter indents one line at a time. For example, consider that line four (4) in Figure 2 should be indented. The indenter asks the heuristic scanner to give tokens from the starting token in backwards direction to the start of the file until a scope introducer is recognized, which for this particular file is `model MoonAndEarth`. The reference position of the start of the scope introducer is computed and line four (4) is indented from this reference position one indent unit. The indentation result is presented in Figure 2.

Indenting Modelica code is far from trivial when incomplete (possibly incorrect) code should be indented correctly. Most of the difficulty comes from Modelica scopes which are hard to recognize using just a scanner and some logic behind it. In languages like C/C++ and Java finding enclosing scopes is very easy as one character tokens are used for the scope opening and closing: `"{"` and `"}"`. In Modelica you need at least two tokens and much more case analysis to find where a scope starts and ends. Complications also arise when mixing if-statements with if-expressions (which was further complicated by the introduction of conditional declarations in the Modelica language). In this particular case we implemented a parser emulator that recognizes these constructs based on scanner tokens delivered backwards.



```
model MoonAndEarth
  import Modelica.Math.sin;
  import Modelica.Math.cos;
  Body earth(
    x(start=0),
    y(start=0),
    vx(start=0),
    vy(start=0),
    mass=5.9736*10^24);
  Body moon(
    x(start=384400000),
    y(start=0),
    vx(start=0),
    vy(start=1023.14),
    mass=7.347673*10^22);
  constant Real G=6.67300*10^(-11) "gravitational constant";
  Real force;
  Real xDistance;
  Real yDistance;
  Real alfa;
equation
  xDistance = moon.x - earth.x;
  yDistance = moon.y - earth.y;
  cos(alfa) = xDistance/sqrt((xDistance^2 + yDistance^2));
```

Figure 2. Example of code after automatic indentation.

The indenter works well in almost all cases, but there are cases in which is impossible to find the correct indentation. For example when the indentation of a line consisting of `"end Name;"` is requested and the scope introducer for `Name` is not found (that is identifier `Name` followed backwards by `class, model, package, block, record, connector` etc.) then the indenter fails and returns the indentation of the previous line.

9 Further Discussion

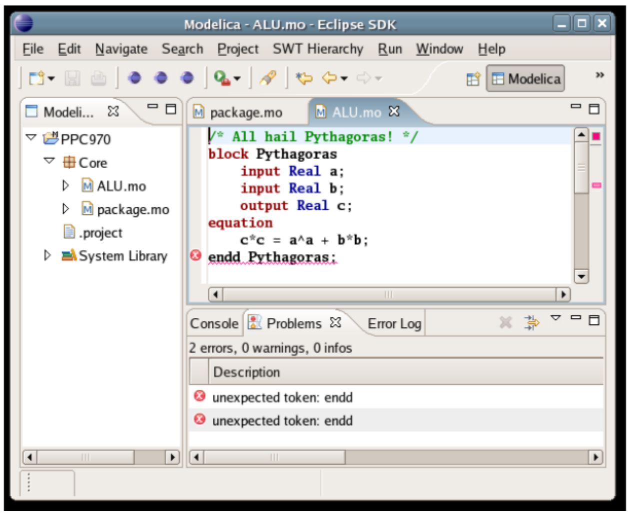
In this section we address some questions from the reviewers:

Question: “A question I have always had is whether there are any "mistakes" in the grammar that should be corrected with respect to these issues. Similarly, how is this handled with the Java tools in Eclipse?”

Answer: The answer to this question highly depends on the syntactic mistake the user made. For example if an **`"end if;"`** is missing at the end of an equation section, but is followed by **`"end`** `Model;"`, then such a mistake can be automatically corrected using a heuristic parser. However, if an opening scope is missing, i.e., **`model`** `Model` (or alternatively an ending scope) there is no way to know where it should be introduced. There are a lot of places that can be proposed:

- Just after the enclosing scope starts (after i.e., `package MyPack` introduction) if there exists such scope or the start of the file if no such scope exists.
- Just after the every existing ending scope of a model found by going backwards from the **end** `Model;`

Right now the Eclipse environment will call the OpenModelica compiler to parse the file each time the file is saved. The parsing errors are reported in the Eclipse environment as a list of errors, but also underlined where the error occurs as shown in Figure 3. Of course if the user selects an entire file and calls the automatic indentation routine, the indentation will work correctly if there are no *large large grammatical errors in the file*.



```
/* All hail Pythagoras! */
block Pythagoras
    input Real a;
    input Real b;
    output Real c;
equation
    c*c = a^a + b*b;
endd Pythagoras;
```

2 errors, 0 warnings, 0 infos

Description
unexpected token: endd
unexpected token: endd

Figure 3. Syntax checking.

Question: “Dymola’s pretty printing algorithm does not appear to be deterministic (it sometimes changes files for no reason just because they have been re-saved). Please discuss this deterministic issue and also what implications the algorithms will have for version control tools (i.e. avoiding complex or unnecessary changes since this will complicate "merge" operations).”

Answer: As exemplified in Sections 3.1 and 7 the disruption to the actual text is minimal so the code-versioning tools would have no problem with merging operations. This was one of our goals when designing and implementing the refactoring tools presented in the paper. The algorithms in this paper also apply to Modelica models constructed programmatically because these can also be viewed as refactorings. In general the construction of models programmatically is performed by a visual component diagram editor. The editor will give commands: `addModel(…)`, `addComponent(…)`, `addConnection(…)`, etc., to the internal handler of the textual model (that works on the AST and the positionTree) which in the case of a file with code formatting will minimally disrupt the existing code and add all the new code correctly indented at the end or in other appropriate places.

10 Related Work

The term refactoring and its use in a general and systematic sense was introduced by Martin Fowler et al [5], also based on earlier work, even though similar code transformation operations were previously available, e.g. in the InterLisp environment [11].

Early work in interactive integrated programming environments including unparsing/pretty printing supporting a specific language was done in the InterLisp system for the Lisp language [11], common principles and experience of early interactive Lisp environments are described in [16], a generic editor/unparser/parser generator used for Pascal (and later Ada) in the DICE system [9], [10], the integrated Mjölner environment with mullti-language editing and unparsing support [17]. None of these approaches preserve comments when unparsing, except the InterLisp environment where the comments were already part of the AST which was just pretty printed with a more readable indentation. However, also in the InterLisp case, all hand indentation and white space added by the user is lost, and text style comments (not part of the AST) are also lost.

Many parser generation systems, e.g. ANTLR [14], Eli [6], CoCo [15], also support unparsing from the generated AST, but do not support preservation of comments and hand-made indentation.

11 Conclusions

We have given a preliminary description of refactorings together with an approach for comment- and indentation preserving unparsing. This is currently ongoing work. Part of the unparser and the refactorings are implemented. A full prototype implementation is expected to be completed early spring 2008.

12 Acknowledgements

This work has been supported by the Swedish Foundation for Strategic Research (SSF) in the RISE and VISIMOD projects, by Vinnova in the Safe and Secure Modeling and Simulation project, and by the Swedish Research Council (VR).

References

[1] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. *Simulation News Europe*, 44/45, Dec 2005. http://ww.ida.liu.se/projects/OpenModelica

[2] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., Wiley-IEEE Press, 2004.

[3] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In *Proc. of the 4th International Modelica Conference*, Hamburg, Germany, March 7-8, 2005.

[4] The Modelica Association. *The Modelica Language Specification Version 3.0*, September 2007. http://www.modelica.org.

[5] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, June 1999.

[6] Uwe Kastens, William M. Waite, and Anthony M. Sloane,. *Generating Software from Specifications*. ISBN 0763741248. Jones and Bartlett Publishers. 2007.

[7] William W Pugh; Steven J Sinofsky. A new language-independent prettyprinting algorithm. Ithaca, NY : Dept. of Computer Science, Cornell University, 1987.

[8] Martin Mikelsons. Prettyprinting in an interactive programming environment. In *Proc. of ACM SIGPLAN SIGOA symposium on Text manipulation*. Portland, Oregon, 1981.

[9] Peter Fritzson. *Towards a Distributed Programming Environment based on Incremental Compilation*. 161 pages. PhD thesis no 109, Linköping University, April 13 1984.

[10] Peter Fritzson. Symbolic Debugging through Incremental Compilation in an Integrated Environment, *Journal of Systems and Software*, 3, pp. 285–294, 1983.

[11] Teitelman, Warren. *INTERLISP Reference Manual*. Xerox Palo Alto Research Center, Palo Alto, CA, 1974.

[12] Eclipse website. http://www.eclipse.org. Referenced Nov 2007.

[13] Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proceedings of the 5th International Modelica Conference (Modelica'2006)*, Vienna, Austria, Sept. 4-5, 2006.

[14] http://www.antlr.org. ANTLR. Accessed Nov 2007.

[15] Hanspeter Mössenböck, Markus Löberbauer, and Albrecht Wöß. The Compiler Generator Coco/R. http://www.ssw.uni-linz.ac.at/coco/. Accessed Nov 2007.

[16] Erik Sandewall. Programming in an Interactive Environment: The “LISP” Experience, *Computing Surveys*, 10:1, Mar. 1978.

[17] J. Lindskov, M. Knudsen, O. Löfgren, Ole Lehrmann-Madsen, and Boris Magnusson (Eds.). *Object-Oriented Environments - The Mjølner Approach*. Prentice Hall, 1993.

Appendix

Here we give (parts of) the generated position tree (posAST) for the code in the example section. The start and end are given in character offsets. The nodes that have -1 as start/end position do not actually exist in the text, but they appear in here to have 1-to-1 mapping to the AST definitions.

```
(Program, (Start: 1, End: 366, {
 (list<Class>, (Start: 23, End: 366, {
   (Class, (Start: 23, End: 366, { (Ident, (Start: 31, End: 35)
     (Boolean Partial, (Start: -1, End: -1) (Boolean Final, (Start: -1, End: -1)
     (Boolen Ecapsulated, (Start: -1, End: -1) (Restriction, (Start: 23, End: 30)
     (ClassDef, (Start: 35, End: 356, {
       (list<ClassPart>, (Start: 38, End: 356, {
         (ClassPart, (Start: 38, End: 356, {
           (list<ElementItem>, (Start: 38, End: 356, {
             (ElementItem, (Start: 38, End: 264, {
               (Element, (Start: 38, End: 264, {
                 (Boolean final, (Start: -1, End: -1)
                 (Option<RedeclareKeywords>, (Start: -1, End: -1)
                 (InnerOuter, (Start: -1, End: -1)
                 (Ident, (Start: -1, End: -1)
                 (ElementSpecEL5, (Start: 38, End: 264, {
                   (Boolean replaceable, (Start: -1, End: -1)
                   (Class, (Start: 53, End: 264, {
                     (Ident, (Start: 47, End: 53)
                     (Boolean Partial, (Start: -1, End: -1)
                     (Boolean Final, (Start: -1, End: -1)
                     (Boolen Ecapsulated, (Start: -1, End: -1)
                     (Restriction, (Start: 38, End: 46)
                     (ClassDef, (Start: 53, End: 264, {
                       (list<ClassPart>, (Start: 53, End: 264, {
                         (ClassPart, (Start: 80, End: 250, {
                           (list<ElementItem>, (Start: 80, End: 221, {
                             (ElementItem, (Start: 80, End: 100, {
                               (Element, (Start: 80, End: 100, {
                                 (Boolean final, (Start: -1, End: -1)
                                 (Option<RedeclareKeywords>,(Start: -1, End: -1)
                                 (InnerOuter, (Start: -1, End: -1)
                                 (Ident, (Start: 91, End: 92)
                                 (ElementSpecEL3, (Start: 91, End: 100, {
                                   (ElementAttributes, (Start: 80, End: 85, {
                                     (Boolean flow, (Start: -1, End: -1)
                                     (Variability, (Start: -1, End: -1)
                                     (Direction, (Start: 80, End: 85)
                                     (ArrayDim, (Start: -1, End: -1)
                                   })
                                   (TypeSpec, (Start: 86, End: 90, {
                                     (Path, (Start: 86, End: 90, {
                                       (Ident, (Start: 86, End: 90)
                                     })
                                     (Option<ArrayDim>, (Start: -1, End: -1)
                                   })
       ... // truncated text due to its large size
       }) (Option<String>, (Start: -1, End: -1)
     }) (Info, (Start: -1, End: -1)
   })
 })
 (Within, (Start: 1, End: 7,
  (Path, (Start: 8, End: 22, {(Ident, (Start: 8, End: 22)})
)
```

Here is another version of the example with character positions for end and start of a Modelica construct:

```
[001]within[007] [008]ParentPackage;[022]
[023]package[030] [031]pack[035]
[036]  [038]function[046] [047]addOne[053] [054]"function that adds 1"[076]
[077]   [080]input[085] [086]Real[090] [091]x[092] [093]=[094] [095]1.0;[099]
                             [100]// line comment[115]
[116]   [119]output[125] [126]Real[130] [131]y;[133]
                             [139]/*  multiple
                                      line
                                      comment */[221]
[222]  [224]algorithm[233]
[234]   [237]y[238] [239]:=[241] [242]x[243] [244]+[245] [246]1.0;[250]
[251]  [253]end[256] [257]addOne;[264]
[265]
[266]  [268]class[273] [274]myClass[281]
[282]    [286]Real[290] [291]y;[293]
[294]  [296]equation[304]
[305]    [309]y[310] [311]=[312] [313]addOne[319](5);[323] [324]// Call to addOne[341]
[342]  [344]end[347] [348]myClass;[356]
[357]end[360] [361]pack;[366]
```

Parts of the abstract syntax tree (AST) of the Example.mo in the example section is presented below. The AST has exactly the same structure as the position tree.

```
adrpo@KAFKA /c/home/adrpo/doc/projects/modelica2008/
$ omc +d=dump Example.mo
Absyn.PROGRAM([
 Absyn.CLASS(Absyn.IDENT("pack"),
  false, false, false, Absyn.R_PACKAGE,
  Absyn.PARTS(
   [Absyn.PUBLIC(
     [Absyn.ELEMENTITEM(
       Absyn.ELEMENT(false, _, Absyn.UNSPECIFIED , "function",
         Absyn.CLASSDEF(false,
          Absyn.CLASS(Absyn.IDENT("addOne"),
           false, false, false, Absyn.R_FUNCTION,
           Absyn.PARTS(
            [Absyn.PUBLIC(
             [Absyn.ELEMENTITEM(
               Absyn.ELEMENT(false, _, Absyn.UNSPECIFIED, "comp",
                Absyn.COMPONENTS(Absyn.ATTR(false, Absyn.VAR, Absyn.INPUT,[]),
                 Absyn.PATH(Absyn.IDENT("Real")),
                 [Absyn.COMPONENTITEM(
                   Absyn.COMPONENT(Absyn.IDENT("x"),[],
                    SOME(Absyn.CLASSMOD([], SOME(Absyn.REAL(1.0))))), NONE)]),
                Absyn.INFO("Example.mo", false, 4, 4, 4, 22)), NONE)),
              Absyn.ELEMENTITEM(
               Absyn.ELEMENT(false, _, Absyn.UNSPECIFIED , "component",
                Absyn.COMPONENTS(Absyn.ATTR(false, Absyn.VAR, Absyn.OUTPUT, []),
                 Absyn.PATH(Absyn.IDENT("Real")),
                 [Absyn.COMPONENTITEM(Absyn.COMPONENT("y",[],
                   NONE), NONE)]),
                Absyn.INFO("Example.mo", false, 5, 4, 5, 17)), NONE))]),
            Absyn.ALGORITHMS(
             ALGORITHMITEM(
              ALG_ASSIGN(
               Absyn.CREF(Absyn.CREF_IDENT("y", [])),
                Absyn.BINARY(
                 Absyn.CREF(Absyn.CREF_IDENT("x", [])),
                 Absyn.ADD,
                 Absyn.REAL(1.0)))))],
            SOME("function that adds 1")),
            Absyn.INFO("Example.mo", false, 3, 3, 10, 13))
  ... // truncated text due to its large size
 ], // end of Absyn.CLASS list
 Absyn.WITHIN(Absyn.IDENT("ParentPackage")
) // end Absyn.PROGRAM
```

Session 6b

Language, Tools and Algorithms

Sensitivity Analysis of Modelica Applications via Automatic Differentiation

Elsheikh, Atya[1] Noack, Stephan[2] Wiechert, Wolfgang[1]
[1] Siegen University, Department of Simulation, {elsheikh,wiechert}@simtec.mb.uni-siegen.de
[2] Research Center Jülich GmbH, Institute of Biotechnology 2, s.noack@fz-juelich.de

Abstract

Modeling and simulation of physical systems is, in general, a complex iterative process. Asserted models are necessarily based on simplifications, and in many cases are subject to improvement and optimization. In this context, a wide range of applications of sensitivity analysis can assist the modeling process, from parameter fitting and optimization through model validation to statistical analysis and experimental design. These common methods, among others, drew increasing attention to a research area of scientific computing, i.e. Automatic Differentiation (AD) of program code. The main objective of this work is to compute derivatives of variables in Modelica models using AD concepts to assist sensitivity analysis applications. It is shown how Open Modelica Compiler (OMC) and other tools simplify the implementation of ADModelica, a prototype of an AD-based tool for Modelica. As a proof of concept, an application in the field of biochemical networks is presented.

Keywords: Sensitivity Analysis, Automatic Differentiation, Open Modelica, Biochemical Networks

1 Introduction

AD is a methodology that refers to algorithmic techniques for semantic augmentation of numerical programs with additional code for derivative computations [6]. For many reasons, AD is a better choice over other ways for computing derivatives such as symbolic differentiation and finite difference methods. In contrast to symbolic differentiation tools, an AD tool does not generate the derivative formula explicitly, but it computes the numerical values of efficient derivative formulas expressed as a program. Nevertheless, the derivative values using AD are as accurate as the values of those generated by symbolic algebra packages up to machine precision. Furthermore, the results are not affected by any truncation errors, resulting from numerical differentiation using divided difference methods.

This work is concerned with AD of Modelica models. Modelica is essentially targeted towards modeling complex systems that can be described by differential algebraic equation (DAE) systems:

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

where $x \in \mathbb{R}^n$, $p \in \mathbb{R}^m$, $F : \mathbb{R}^{2\cdot n+m+1} \to \mathbb{R}^n$. Assuming that $\partial F/\partial x$ is non-singular for all $p \in \mathbb{R}^m$, and that $\partial x/\partial p$ is smooth enough, sensitivity analysis requires the sensitivities $\partial x/\partial p$ of solution variables with respect to perturbations in the parameters. These can be calculated by solving the original DAE system (1) and m sensitivity systems:

$$\begin{array}{c} \frac{\partial F}{\partial \dot{x}} \cdot \frac{\partial \dot{x}}{\partial p} + \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p} + \frac{\partial F}{\partial p} = 0, \\ \frac{\partial x}{\partial p}(0) = \frac{\partial}{\partial p}(x_0(p)) \end{array} \qquad (2)$$

obtained by explicit differentiation of (1) with respect to p [14]. Additionally, the sensitivities $\partial x_i/\partial x_j$ of certain variables x_i with respect to other specific variables x_j might be needed.

This paper presents first experiences with a prototype of a tool, ADModelica, that augments Modelica models with Modelica code for computing certain sensitivities, with minimal user efforts. Aiming at the full-support of Modelica language constructs, we implemented a first version, which supports most basic constructs of Modelica. The rest of the paper is structured as follows. Section 2 introduces basic terminologies and algorithmic aspects of AD. The Generalization of the introduced concepts into the Modelica framework is clarified in Sect. 3. Section 4 presents the ADModelica tool and briefly discusses some design and implementation issues. In Sect. 5, applications in the field of Biochemical Engineering using a special library is presented. Finally, conclusions are presented and future work is discussed in Sect. 6.

2 Introduction to Automatic Differentiation

Many techniques such as numerical differentiation or computer algebra methods are used to compute derivatives. However, AD has proved to be superior over other ways for obtaining derivatives in terms of computational efficiency, numerical precision and discretization parameters. ADIC [2] and ADIFOR [1] are examples of a wide range of AD tools for differentiating C and Fortran programs respectively. In this section, some basic terminologies of AD are introduced.

2.1 Basic Concepts

Formally, given a program P that computes a function:

$$f : x \in \mathbb{R}^n \rightarrow y \in \mathbb{R}^m$$

with n inputs and m outputs, a new code P' is sought to compute the Jacobian $f' = \partial y / \partial x$. The following terms are commonly used in the context of AD:

- *Independent variables* are program input variables with respect to which derivatives are sought.
- *Dependent variables* are output variables whose derivatives are desired.
- A *derivative object* represents some derivative information, such as a vector of partial derivatives $(\partial z/\partial x_1, ..., \partial z/\partial x_n)^T$ of a variable z with respect to a vector $x = (x_1, x_2, ..., x_n)^T$.
- Any program variable with which a derivative object is associated is called an *active variable*.

2.2 Algorithmic Aspects of AD

The key concept behind AD is that every computation, no matter how complex it is, is executed on a computer as a sequence of a limited set of elementary operations, such as addition and multiplication, and intrinsic functions, such as sine and cosine. The derivative of each of these elementary operations can be computed by applying the chain rule to combine the local partial derivatives of each executed operator. An AD tool operates by systematic application of the chain rule on the numerical code. For example, let $a(x)$ and $b(x)$ be intermediate values that depend on an independent variable x, and let $c := f(a, b)$. Then by using the chain rule, $\nabla_x c$ the derivative of the dependent variable c with respect to x is computed as:

$$\nabla_x c := \frac{\partial f}{\partial a} \cdot \nabla_x a + \frac{\partial f}{\partial b} \cdot \nabla_x b \qquad (3)$$

The chain rule is associative. If $y := f(g(x))$, $\partial y/\partial x$ can be computed by forwardly accumulating the derivatives (i.e. $\partial f/\partial g$ and $\partial g/\partial x$) in the computational path from the independent variable(s) (eg. x) to the dependent variable(s) (eg. y). By exploiting the associativity of the chain-rule, the augmented program is generated to evaluate $f(x)$ and the partial derivatives of f simultaneously.

2.3 Why AD for Modelica?

AD is naturally implemented by Modelica compilers to provide partial derivatives of functions for solving the DAE index problem [12]. A DAE system of high index is transformed into a solvable ODE system by differentiating some equations selected by Pantelides's algorithm [13]. Here, AD is chosen for the fundamentally different task of calculating sensitivities of solution variables, motivated by the following reasons:

- DAE systems are represented in Modelica by using components and connectors; internal formulas in components and models may be implemented with loops and many branches. Therefore, it makes sense to utilize existing tools and concepts of handling DAE systems, used by modelica compilers, for generating derivative formulas.
- For a Modelica model that computes a DAE System (1), a lot of common sub-expressions in F, $\partial F/\partial x$ and $\partial F/\partial p$ arise. In many cases, these common sub-expressions need not to be re-evaluated if these partial derivatives are computed using AD.
- Compiler techniques used for reducing the dimension of a generated DAE system, can be adopted by AD for reducing the number of equations needed to be differentiated , instead of blind differentiation of all equations, as the DAE system (2) suggests [4].

3 Differentiating DAE Systems

Assignments (eg. $x := f(y,z)$) are the main elementary units of procedural languages, whereas declarative equations (eg. $f(x(t),y(t),z(t)) = 0$) constitute the main building units of Modelica. While an assignment is a relation between inputs (a collection of values) and one output, an equation is a relation between several variables, that needs to be fulfilled concurrently. This conceptual difference has vital consequences on the way derivatives can be generated for DAE systems, namely, AD techniques for classical languages, such as C/FORTRAN, are not necessarily applicable for equation-based languages.

3.1 Example

Consider the DAE System

$$\begin{array}{ll} \dot{A} = -v, & A(0) = A_0 \\ \dot{B} = v, & B(0) = B_0 \\ v = v_{max} \cdot \frac{A}{A+k} \cdot \frac{I_k}{B+I_k} & \end{array} \tag{4}$$

describing the dynamics of a chemical reaction, in which a chemical substance with concentration $A = A(t)$ is converted to another chemical substance with concentration $B = B(t)$. $v = v(A,B,t)$ stands for reaction rate and v_{max}, k and I_k stand for enzymatic parameters. The first two ordinary differential equations represent balance equations, whereas the third equation describes the reaction rate using the well-known Michaelis-Menten Kinetics [7]. The sensitivities of $x = (A,B,v)^T$ w.r.t. parameters $p = (v_{max},k,I_k)^T$ can be computed as in (2) by adding the following equations:

$$\begin{array}{ll} \dot{A_p} = -v_p, & A_p(0) = 0 \\ \dot{B_p} = v_p, & B_p(0) = 0 \\ v_p = \frac{\partial}{\partial p} f(A,B,v_{max},k,I_k) & \end{array} \tag{5}$$

to (4), where

$$f(A,B,v_{max},k,I_k) = v_{max} \cdot \frac{A}{A+k} \cdot \frac{I_k}{B+I_k} \tag{6}$$

$$v_p = \nabla_p v = (\frac{\partial v}{\partial v_{max}}, \frac{\partial v}{\partial k}, \frac{\partial v}{\partial I_k})^T \tag{7}$$

and A_p, B_p are similar to v_p. Given that $J_p = I_3$ (Identity matrix of size 3), i.e.:

$$\begin{array}{ll} \nabla_p v_{max} & := (1,0,0)^T; \\ \nabla_p k & := (0,1,0)^T; \\ \nabla_p I_k & := (0,0,1)^T; \end{array} \tag{8}$$

(5) can be easily implemented in Modelica with the help of arrays. Notice that := stands for assignments.

3.2 Utilizing Common Sub-expressions

Given that the values of $A(t)$ and $B(t)$ are known for a time point t, $v(t)$ and $v_p(t)$ can be computed from the DAE systems (4) and (5). The third equation $v_p = \partial f / \partial p$ in the DAE system (5) consists of three equations of similar algebraic structure. Excessive re-evaluation of common sub-expressions arising in v and v_p can be avoided by dividing the equation $v = f$ in the DAE system (5) into a set of binary assignments using the Abstract Syntax Tree (AST) of v as shown in Fig. 1. The gradient of $v(t)$ is computed by forward accumulation of the gradients of the intermediate variables obtained by differentiating each assignment instead of direct differentiation of the algebraic formula. An implementation for the DAE systems (4) and (5) looks as follows:

$$\begin{array}{l} \dot{A} = -v \\ \quad \frac{\partial}{\partial t} \nabla_p A = -\nabla_p v \\ \dot{B} = v \\ \quad \frac{\partial}{\partial t} \nabla_p B = \nabla_p v \\ \\ u_1 := v_{max} \cdot A; \\ \quad \nabla_p u_1 := \nabla_p v_{max} \cdot A + v_{max} \cdot \nabla_p A; \\ u_2 := A + k; \\ \quad \nabla_p u_2 := \nabla_p A + \nabla_p k; \\ u_3 := u_1 \cdot u_2; \\ \quad \nabla_p u_3 := \nabla_p u_1 \cdot u_2 + u_1 \cdot \nabla_p u_2; \\ u_4 := B + I_k; \\ \quad \nabla_p u_4 := \nabla_p B + \nabla_p I_k; \\ u_5 := I_k / u_4; \\ \quad \nabla_p u_5 := (\nabla_p I_k \cdot u_4 - I_k \cdot \nabla_p u_4)/u_4^2; \\ v \ := u_3 \cdot u_5; \\ \quad \nabla_p v \ := \nabla_p u_3 \cdot u_5 + u_3 \cdot \nabla_p u_5; \end{array} \tag{9}$$

In this way, common sub-expressions are evaluated only once, and hence less arithmetic operations are needed. The assignments can be implemented in Modelica with the help of the *algorithm* construct.

3.3 Limitations

While optimizing common sub-expressions works well for AD of classical procedural languages, this may not be the case with equation-based languages. For example, in the DAE system (4), $v(0)$ can be computed by considering the available values of $A(0)$ and $B(0)$. Then, $v(0)$ is used to compute subsequent values of A and B, and hence forth. That is, at each iteration, $A(t)$ and $B(t)$ are used to compute $v(t)$. In other words, the values $v(t)$ *depends on* $A(t)$ and $B(t)$. By this way, computing $v(t)$ from $A(t)$ and $B(t)$

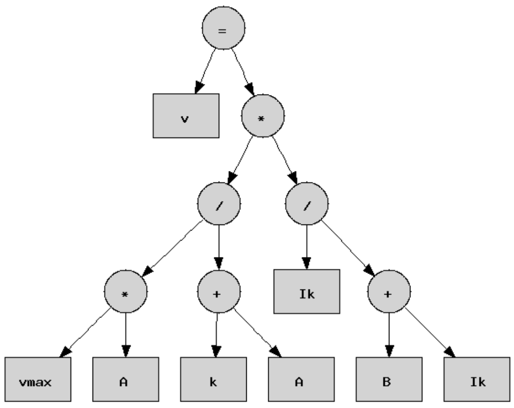


Figure 1: Abstract Syntax Tree (AST) of $v = f$

in (9) does not change the dependency of variables. However, in general, an equation can be divided into a set of binary operations if the output variable depends on the variables arising in the left hand side of all intermediate assignments.

Additionally, the dimension of the rewritten DAE system increases according to the way the Modelica compiler handles local variables. If intermediate results of local variables are always stored, this exhausts extra storage and computation time. Note that, the number of local variables can be reduced by reusing local variables. For example, there is no need to introduce new local variables u_4 and u_5 if u_1 and u_2 are used instead. Moreover, excessive use of the *algorithm* section may disable some optimization methods for reducing the dimension of a DAE system and hence worsen the performance. Finally, side effects implied by the enforced order of sub-expressions evaluation result in slightly different results for state variables.

4 Automatic Differentiation of Modelica Code

ADModelica is a prototype of a source-to-source AD tool that strives to support Modelica programs. The source-to-source approach employs a combination of classical- and equation-based compiler techniques to transform a program source code into a new source code that computes the derivatives. This section gives a quick overview of the implementation of ADModelica.

4.1 Possible Approaches

There are three levels, on which AD of (implicit) DAE systems can operate:

1. **Library level**: All library units (i.e. components and connectors) are differentiated independently to generate another library that additionally computes parameter sensitivities of variables. Each component is augmented with code for derivatives.

2. **Flat Model Level**: The source code is given as (or transformed into) pure equations, represented by elementary Modelica's constructs, rather than physical formulation with components and connectors. Sensitivity Equations are added in a new Modelica model.

3. **Generated C-code level**: The generated C-code is differentiated.

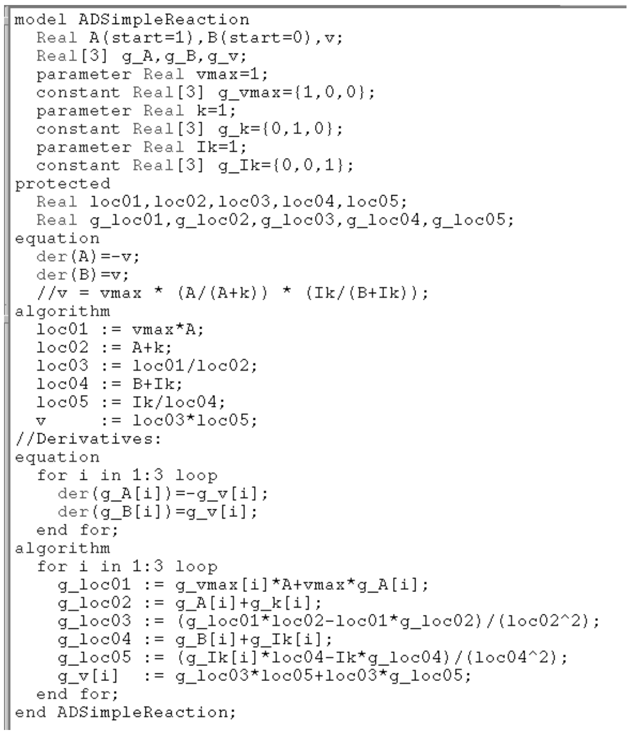
In [4], the above approaches are discussed in more details. The adopted approach is based on differentiation on the flat model level. The current supported input models, are namely those, which flattened models have pure mathematical formulation. Particularly, input models with components, connectors and arrays with equations expressed as *for*-loops are supported. However, some control constructs in Modelica, such as *if*, *while* and others, are not yet supported. As a remark, AD of such classical languages constructs is a well-know problem and has been successfully handled [6].

4.2 Overview of ADModelica

Figure 2 shows the corresponding Modelica implementation of the DAE system (9). The user specifies the independent variables. If not specified, all parameters are considered as independent variables. To every variable v of type Real an array representing the gradient of that variable *g_v* is associated. The array's size represents the number of independent variables. Each entry of the array represents the derivative of v with respect to an independent variable. To each active variable, a gradient is associated. ADModelica follows a conservative strategy that considers all variables and parameters active. In that case, non-interesting parameters have the zero gradients.

4.3 Design and Implementation

Implementing an AD tool from scratch, supporting a wide set of Modelica grammar, would be an ex-



```
model ADSimpleReaction
  Real A(start=1),B(start=0),v;
  Real[3] g_A,g_B,g_v;
  parameter Real vmax=1;
  constant Real[3] g_vmax={1,0,0};
  parameter Real k=1;
  constant Real[3] g_k={0,1,0};
  parameter Real Ik=1;
  constant Real[3] g_Ik={0,0,1};
protected
  Real loc01,loc02,loc03,loc04,loc05;
  Real g_loc01,g_loc02,g_loc03,g_loc04,g_loc05;
equation
  der(A)=-v;
  der(B)=v;
  //v = vmax * (A/(A+k)) * (Ik/(B+Ik));
algorithm
  loc01 := vmax*A;
  loc02 := A+k;
  loc03 := loc01/loc02;
  loc04 := B+Ik;
  loc05 := Ik/loc04;
  v     := loc03*loc05;
//Derivatives:
equation
  for i in 1:3 loop
    der(g_A[i])=-g_v[i];
    der(g_B[i])=g_v[i];
  end for;
algorithm
  for i in 1:3 loop
    g_loc01 := g_vmax[i]*A+vmax*g_A[i];
    g_loc02 := g_A[i]+g_k[i];
    g_loc03 := (g_loc01*loc02-loc01*g_loc02)/(loc02^2);
    g_loc04 := g_B[i]+g_Ik[i];
    g_loc05 := (g_Ik[i]*loc04-Ik*g_loc04)/(loc04^2);
    g_v[i]  := g_loc03*loc05+loc03*g_loc05;
  end for;
end ADSimpleReaction;
```

Figure 2: Implementation of the DAE system (4) and its Sensitivity Equations (5)

pensive and error-prone process. Therefore, existing tools and software are utilized by ADModelica, particularly OMC [5]. OMC allows communication with other tools through the CORBA interface. Figure 3 shows the main steps performed to generate a Modelica model that computes additional required derivatives. These steps are summarized as follows:

- **Flattening**: A high-level model is transformed to a model with pure mathematical equations, using the Open Modelica Compiler (OMC). ADModelica makes use of the CORBA interface, offered by OMC.
- **Transforming to intermediate format**: The ModelicaXML parser [15] parses an input model to an easy-to-handle format, in which the AST representation of the equations are implicitly inherited. The ASTs are extracted into intermediate format in Java classes.
- **Analyzing**: The dimension of the generated DAE system is reduced by removing alias equations (s.a. $x = y$ and $x + y = 0$) [9]. The computational path between variables is computed [3, 8].
- **Differentiating**: The ASTs of the derivatives are computed. A conservative strategy is to differentiate all equations. However, it is enough to

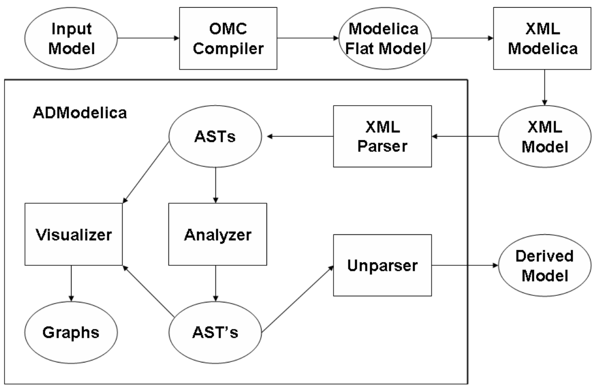


Figure 3: The Architecture of ADModelica

  differentiate all equations laying in all Strongly Connected Components (SCCs) of the computational path from the independent variable(s) to the dependent variable(s).
- **Unparsing**: The differentiated model is generated with additional code for derivatives.
- **Visualizing ASTs**: Producing graphs of the ASTs was proven to be useful during the course of development, for finding potential semantical mistakes.

5 Application

Modeling the dynamics of metabolic reaction networks has a wide spectrum of applications. Special attention has been paid to modeling biochemical systems with Modelica [11]. In general, the parameters expressing the characteristics of enzymatic reactions (eg. reaction rate, enzyme activation/inhibition constants, etc.) are one of the largest source of uncertainty in modeling metabolic networks, and are not necessarily known. Their values might be estimated by fitting them to measured data, resulted from stimulus-response experiments [16]. Estimating the correct values of parameters can reveal hidden information about the system. However, even in that case, the asserted model alone does not explain the underlying behavior.

Understanding the functions of enzymatic reactions within a metabolic network can be achieved by measuring changes to directed perturbations of certain parameters (eg. quantity of a certain enzyme). While

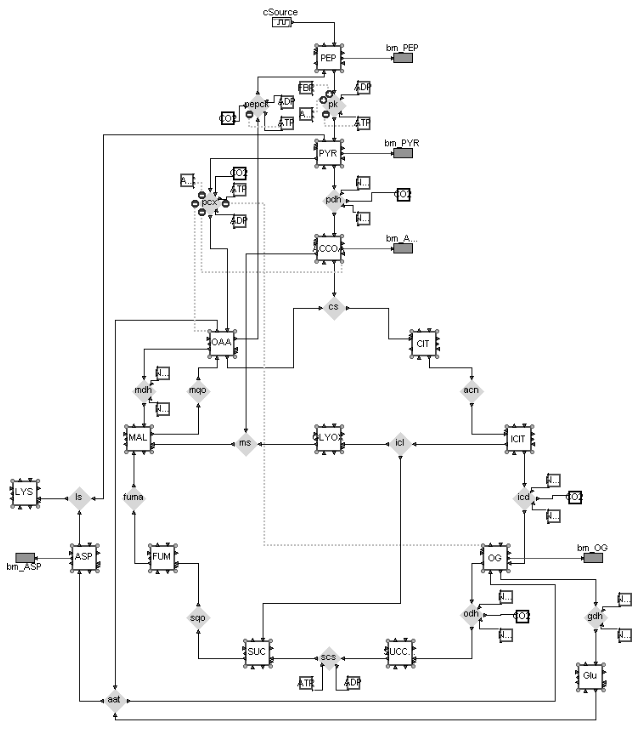


Figure 4: A dynamic Metabolic Network

this can be experimentally difficult or impractical, it is easier to quantify the effect of these changes using a validated model [17]. This can be achieved by computing the sensitivities of reaction rates and concentration to parameters $\partial r/\partial p$ and $\partial c/\partial p$, and the sensitivities of reaction rates to concentration of metabolites $\partial r/\partial c$. Using these sensitivities, the well known quantities of Metabolic Control Theory, i.e. the concentration and flux control coefficients C^M and C^F, can be calculated [10, 7].

In Figure 4, a dynamic metabolic network model including reactions of the tricarbon acid cycle is shown. The network has been implemented using a specialized library for biochemical networks, making use of many object-oriented features of the Modelica language. Various classes (e.g. Enzyme, Metabolites, Reactions) are the main common objects. Objects are connected via interfaces for potential variables (e.g. concentration c) and flux variables (e.g. reaction rate r). The dimension of the corresponding DAE system of the flattened model is 690. The number of non-trivial equations is 182. It takes few milli-seconds to simulate the network using Dymola (Dynasim AB, Sweden). The model was differentiated w.r.t. 64 independent variables, 49 of which are parameters corresponding to enzymatic characteristics and 15 concentrations variables. The dimension of the generated DAE is 12,270. It takes about 35 seconds to get the network and corresponding sensitivities simulated.

Investigations on the dynamics of metabolic network models mostly follow a system perturbation starting from a stationary state. In this example, the network is stimulated by a pulse of the input metabolite PEP. Results show that responses of following metabolite pools are very fast (e.g. PYR) or delayed (e.g. AC-COA). Especially in the case of the output metabolite LYS the concentration change is rather low in the given time frame. The results are used to identify some model parameters, which show a higher sensitivity in the instationary case directly after system perturbation, as well as others, which generally do not have any significant influence on the corresponding flux.

6 Summary and Future Work

This work shows that AD is a natural choice for computing sensitivities of solution variables for Modelica models. ADModelica is a prototype of a source-to-source AD tool for the Modelica language. It follows the flat model approach, as it is easy to implement because it does not consist of high-level language constructs. ADModelica utilizes OMC by using CORBA communication. Potential improvements of ADModelica can be achieved by making more use of OMC. OMC access a lot of facilities that can be utilized by ADModelica. Examples involve, but are not limited to:

- Symbolic manipulation of algebraic equations
- An intermediate format for computational graphs for DAE systems
- Utilizing the Dependency Flow Graph (DFG) of variables for decomposing a large resorted DAE system in Block Lower Triangular (BLT) format into smaller DAE systems

These facilities are used for optimizing common subexpressions, reducing the number of equations needed to be differentiated and computing sensitivities of variables w.r.t. other (non input) variables. Although, these are partially implemented by ADModelica, it is certainly better to rely on the reliable well-maintained and continuously growing OMC.

acknowledgment. This research is funded by German Ministry of Education and Research (BMBF) within the Sysmap project.

References

[1] C. H. Bischof, P. Khademi, A. Mauer, and A. Carle. *Adifor 2.0: Automatic Differentiation of Fortran 77 Programs*. *IEEE Computational Science & Engineering*, 3(3):18–32, Fall 1996.

[2] C. H. Bischof, L. Roh, and A. Mauer. *ADIC — An Extensible Automatic Differentiation Tool for ANSI-C*. *Software: Practice and Experience*, 27(12):1427–1456, 1997.

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

[4] A. Elsheikh and W. Wiechert. Automatic Sensitivity Analysis of DAE-Systems Generated from Equation-Based Modeling Languages. *submitted to 5th International Conference on Automatic Differentiation*, Bonn, Germany, 11-15 August 2008.

[5] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldami, H. Johansson, and A. Karström. The Open Source Modelica Project. In *Proceedings of The 2nd International Modelica Conference*, pages 297–306, Munich, Germany, March 2002.

[6] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.

[7] R. Heinrich and S. Schuster, editors. *The Regulation of Cellular Systems*. Springer, 1996.

[8] A. Leitold and K. M. Hangos. Structural Solvability Analysis of Dynamic Process Models. *Computers & Chemical Engineering*, 25:1633–1646, 2001.

[9] C. Maffezzoni, R. Girelli, and P. Lluka. Generating Efficient Computational Procedures from Declarative Models. *Simul. Pr. Theory*, 4(5):303–317, 1996.

[10] K. Mauch, S. Arnold, and M. Reuss. Dynamic Sensitivity Analysis for Metabolic Systems. *Chem. Eng. Sci.*, 52:2589–2598, 1997.

[11] E. L. Nilsson and P. Fritzson. A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems. In *Modelica2005, the 4th International Modelica Conference*, pages 85–93, Hamburg, Germany, March 2005.

[12] H. Olsson, H. Tummescheit, and H. Elmqvist. Using Automatic Differentiation for Partial Derivatives of Functions in Modelica. In *Modelica2005, the 4th International Modelica Conference*, pages 105–112, Hamburg, Germany, March 2005.

[13] C. C. Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, Mar. 1988.

[14] L. Petzold, S. T. Li, Y. Cao, and R. Serban. Sensitivity Analysis of Differential-Algebraic Equations and Partial Differential Equations. *Computers & Chemical Engineering*, 30:1553–1559, 2006.

[15] A. Pop and P. Fritzson. Modelicaxml: A Modelica XML Representation with Applications. In *Proceedings of The 3rd International Modelica Conference*, pages 419–429, Linköping, Sweden, November 2003.

[16] S. Wahl, M. Haunschild, M. Oldiges, and W. Wiechert. Unravelling the Regulatory Structure of Biochemical Networks Using Stimulus Response Experiments and Large-scale Model Selection. In *IEEE Proceedings Systems Biology*, volume 153, pages 275–285. IEEE Computer Society, 2006.

[17] W. Wiechert. Validation of Metabolic Models: Concepts, Tools, and Problems. In B. Kholodenko and H. Westerhoff, editors, *Metabolic Engineering in the Post Genomic Era*, chapter 11. Horizon Bioscience, 2004.

Synchronous and asynchronous events in Modelica: proposal for an improved hybrid model

Ramine Nikoukhah[1] Sébastien Furic[2]
[1] INRIA Rocquencourt BP 105, 78153 Le Chesnay Cedex, France
[2] LMS-Imagine, 7 place des Minimes, 42300 Roanne, France
ramine.nikoukhah@inria.fr furic@amesim.com

Abstract

The event synchronism in Modelica has been a subject of contradictory interpretations. An interpretation inspired by Scicos formalism [2] has been shown to provide desirable properties. In this interpretation, all independent events are assumed asynchronous; that includes events generated by the **sample** keywords. But in analogy with the way multi-rate systems are modeled in Simulink, it is desirable also to consider **sample** generated events as synchronous. In this paper, we propose a special treatment for the keyword **sample** to overcome this dilemma.

Keywords: Modelica, Scicos, synchronism, real-time code generation

1. Introduction

In [1], it is argued that all Modelica events should be considered asynchronous unless they are derived explicitly from a single event. This is in contrast to Dymola's implementation where all events are considered potentially synchronous, by default. In Dymola, simultaneity is interpreted as synchronism. In [1] it is shown that the asynchronous point of view not only leads to the generation of more efficient code but it can also allow for separate compilation of isolated modules.

It may be argued that with the asynchronous point of view, non-deterministic behavior becomes an issue. But the problem of non-determinism here is not worse than in the fully synchronous context because the reason for non-determinism in hybrid systems is the finite precision of the numerical solver. Indeed, it is not more nondeterministic to assume that two events: **time** > 3 and x < 2, where x is a continuous time variable, are asynchronous than assuming that they are potentially synchronous.

Asynchronous means that even if the two events occur (in theory) at exactly the same time, one is considered to occur just before or after. In the synchronous context, the formalism considers also the case of the two events happening simultaneously and treats it differently. However in practice, since events such as x < 2 are detected by the zero-crossing mechanism of the numerical solver, there is very little chance that two events be detected simultaneously even if theoretically they are simultaneous. So in the synchronous context, the non-determinism is even worse: not only there are three possible outcomes in the presence of two zero-crossing events but the user is lead to believe that it can count on simultaneous detection when in most cases the result is completely unpredictable.

The **sample** generated events, even though not produced by the zero-crossing mechanism during the simulation, should naturally be considered as independent and thus asynchronous as well. But this goes against the usual practices in Modelica where synchronism is often implicitly assumed. In this paper we propose a very special interpretation of the **sample** keyword which not only leads to models in accord with our asynchronous framework but assures synchronism among **sample** generated events.

2. Asynchronous framework

In the asynchronous interpretation of the Modelica specification, two events are considered synchronous only if they can be traced back to a single event source. For example in the following model:

```
when sample(0, 1) then
  d = pre(d) + 1;
end when;
when d > 3 then
  a = pre(a) + 1;
```

```
end when;
```

the event d > 3 is synchronous with the event **sample**(0, 1). The former is the source of the latter. But in

```
der(x) = x ;
when sample(0, 1) then
  d = pre(d) + 1;
end when;
when x > 3 then
  a = pre(a) + 1;
end when;
```

the two events are not synchronous. There is no unique source of activation at the origin of these events. So these events are considered asynchronous even if the two events are activated simultaneously; even if we can prove mathematically that they always occur simultaneously.

The basic assumption is that events detected by the zero-crossing mechanism of the numerical solver (or an equivalent mechanism used to improve performance) are always asynchronous. So even if they are detected simultaneously by the solver, by default they are treated sequentially in an arbitrary order.

3. Special case of sample construct

Under the asynchronous assumption, and by treating the **sample** keyword as a macro, the following program:

```
model M
  Boolean b;
  ...
equation
  b = sample(0, 1);
  when f(b) then
    ... g(b)...
  end when;
  ...
end M;
```

can be expanded as follows:

```
model M
  discrete time Integer k(start=0);
  Boolean b;
  ...
equation
  when time >= k then
    k = pre(k) + 1;
  end when;
  b = false;
  when change(k) and f(true) then
    ... g(true)...
  end when;
  ...
end M;
```

This means that we are lead to assume that different **sample** statements generate asynchronous events (we also lose periodicity information contained in the arguments of the **sample). For** example, in the model:

```
when sample(0, 1) then
  b = a;
end when;
when sample(0, 1) then
  a = b + 1;
end when;
```

the variables a and b are evaluated in an arbitrary order and no algebraic loop is detected..

Dymola on the other hand assumes that all events are synchronous. In particular it assumes that all the equations in both **when** clauses in this example may have to be satisfied simultaneously. That is why Dymola finds an algebraic loop in this example.

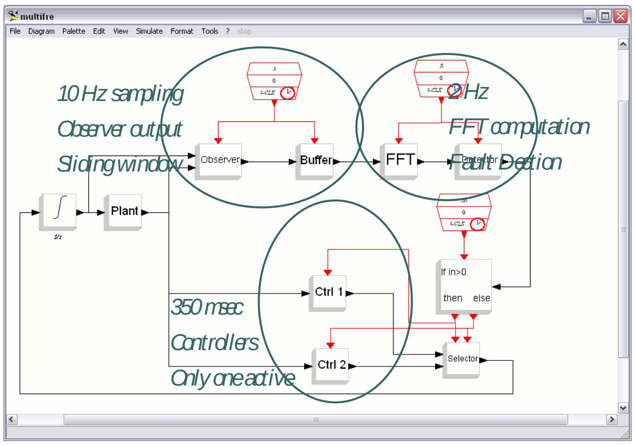
This seems reasonable; however Dymola also finds an algebraic loop in:

```
when sample(0, 1) then
  b = a;
end when;
when sample(0.5, 1) then
  a = b + 1;
end when;
```

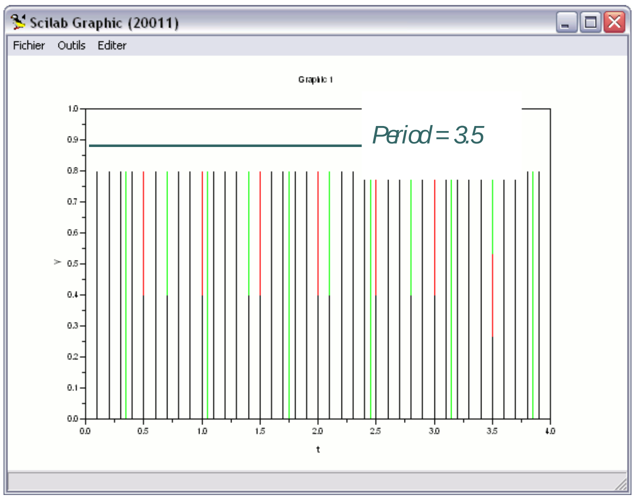
when clearly no algebraic loop exists in this model.

4. Periodicity information

The periodicity information may not be very useful for simulation but it is precious for real-time code generation. It is a lot easier to generate embedded code for a discrete-time system when the system is periodic and all the timing information is available during the code generation process. Consider for example the following system, which represents a continuous time plant with a discrete-time failure detector and a reconfigurable controller. Different components of the detector/controller mechanism run at different frequencies; we say then that the system is multi-rate.



Consider now the problem of hard real-time code generation for this mechanism, which contains three basic frequencies with periods 0.1, 0.5 and 0.35. The events corresponding to these three clocks are synchronized at different time instants. In general there could be 7 different situations for which static code generation must be performed but in this particular case only 5 situations come up. The important information to note here is that the system will function in a fully periodic way and the timing of all the situations can be computed in advance thanks to information on the periods (and offsets if any) of the clocks. It turns out that in this case, the overall period is 3.5; the timings of different event situations are illustrated below:



To model such a system in Modelica, it is common practice to assume synchronism of independent **sample** sources (this is done in particular, by developers of the Modelica Standard Libraries) and represent each clock by an independent **sample** statement.

But in the asynchronous point of view adopted by us, following the replacement of the **sample** macros with the corresponding Modelica code as presented previously, the clocks become asynchronous. In this framework, it is necessary to use a single clock and derive the other clocks by sub-sampling; otherwise the behavior of the system will not correspond to the desired behavior.

5. Synchronous sample

We have seen that on one hand it is desirable to consider all independent events to be asynchronous and on the other hand, it is convenient to force, depending on their arguments, sample generated events as synchronous.

The type of synchronism considered here has nothing to do with the way Dymola enforces synchronism but it is rather close to Simulink's way of handling multirate systems and Scicos' **SampleClk** blocks. The idea is to synthesize a basic clock at a precompilation phase so that all the synchronous clocks defined by **sample** statements can be obtained by sub-sampling the basic clock. The computation of the parameters of this basic clock is straightforward, see [3] for details. Here is a simple example:

```
when sample(0, 2) then
  <expr1>;
end when;
when sample(0, 3) then
  <expr2>;
end when;
```

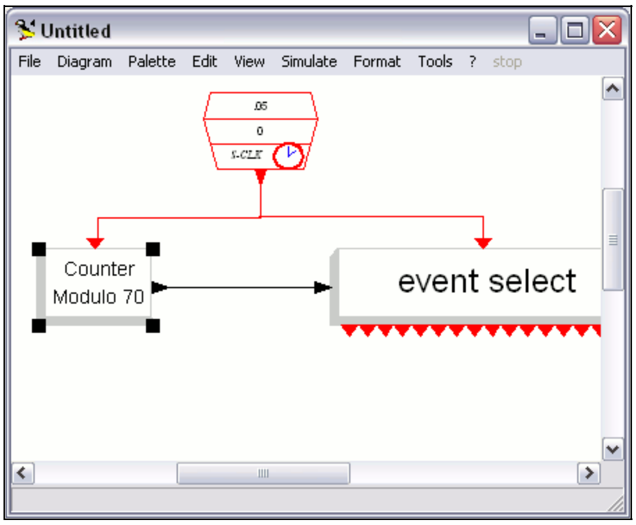
The periods involved in this case are 2 and 3; the period of the basic clock is obtained by computing the greatest common divisor of 2 and 3, which is 1. The overall period in this case is 6, so one way the precompiler could modify the code is as follows:

```
when sample(0, 1) then
  k = mod(pre(k) + 1, 6);
  if k == 0 then
    <expr1>;
    <expr2>;
  elseif k == 2 or k == 4 then
```

```
    <expr1>;
  elseif k == 3 then
    <expr2>;
  end if;
end when;
```

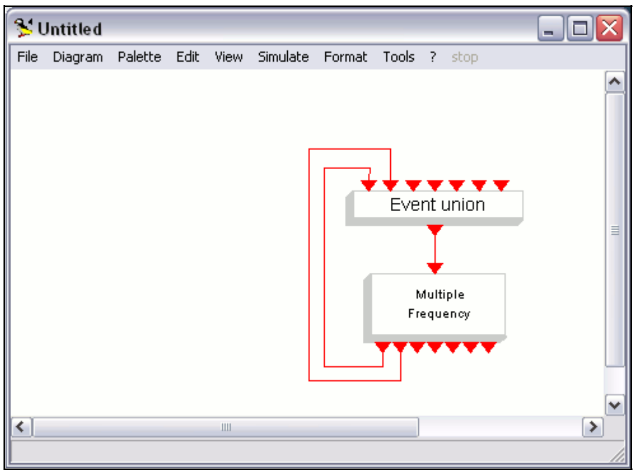
This way of sub-sampling clocks have already been introduced in the Modelica specification (see for example the fast sample, slow sample example on page 81 of [6]).

Going back to our code now, we see that it contains a single **sample** keyword so it is a synchronous code (assuming no **when** constructs are present in the rest of the model). The **sample** construct can now be expanded as previously described. This construction in Scicos is referred to as a periodic construction. For example going back to the detector/controller model from the previous section, the period of the basic clock would be 0.05 (the greatest common divisor of 0.1, 0.5 and 0.35) and the periodic solution would look like the following. Note that the modulo counter counts from 0 to 69 because the period is 3.5 and the basic clock's period is 0.05.



An alternative procedure consists of constructing a vector of time instants where events occur over a single period (in this case [0,2,3,4]) and generate events using independent event sources corresponding to time instances which, modulo 6, are mapped to the elements of this vector.

This construction can be more efficient for simulation but the periodic solution has the advantage of yielding a synchronous code. For the detector/controller example, the non-periodic (asynchronous) construction looks like the following. To keep the diagram simple we have only drawn two of the activation links out of possible 7 (actually 5 in this particular case).



Periodic solutions are also desirable for real-time code generation because the embedded code can be driven by a hardware fixed frequency clock.

6. Implications of the proposal

By admitting that the asynchronous assumption on independent event generators is the correct interpretation, if the special treatment proposed for the **sample** keyword is not used, most discrete-time models in use won't operate properly. The reason is that, despite some recommendations in the language specification, synchronism of independent **sample** sources is assumed by library developers (in particular, by developers of the Modelica Standard Libraries). This practice, mostly driven by analogy with other practices frequently encountered in Simulink-based modeling, conflicts with the asynchronous assumption made in our hybrid model.

To impose synchronism among various discrete-time models, instead of relying on the usage of identical **sample** keywords, synchronization signals should be used. This issue has been discussed in [5] where activation signals have been introduced.

Even though the use of activation signals is a powerful modeling mechanism that should be considered in future Modelica, for the special case of periodic event clocks, the treatment of the **sample** keyword as proposed in this paper avoids the need for there usage. Indeed, by assuming this treatment, backward compatibility for discrete-time models would be guaranteed. The precompilation phase makes the necessary modifications that assure the synchronization of isolated models that are related to each other simply because they include identical **sample** keywords. The backward compatibility is also assured in the case of multi-rate systems (when non identical **sample** keywords are present in the model)..

7. Conclusion

We have proposed to interpret the **sample** keyword in Modelica in a special manner in such a way as to assure synchronism between these keywords yet staying within the asynchronous framework proposed in [1].

The implementation consists of isolating the **sample** keywords in the flat Modelica model. If only one such keyword is present, then it is transformed as explained in the paper. If more than one **sample** is present in the model, the necessary clock computations are performed and all the **sample** constructs are replaced by conditional statements driven by a single **sample**, as illustrated on an example in the paper. This **sample** is then transformed as in the previous case.

Beside backward compatibility (no Modelica model needs to be altered), allowing the usage of independent **sample** keywords to model synchronous multi-rate systems provides valuable information for real-time code generation. However some issues remain to be solved with this approach, especially the way **sample** constructs are translated at the type level (they probably can not be abstracted away during type computation since the public information they carry may interfere with compatibility checks of models).

Using Modelica for real-time embedded code generation has great potentials. Unlike most code generation environments, in Modelica the execution semantics can be broken up into very fine grains and manipulated symbolically. In most cases this means that real-time code can be obtained without having to use preemption. This issue will be examined in a future paper.

References

[1] Nikoukhah, R., "Hybrid dynamics in Modelica: Should all events be considered synchronous", in Proc. EOOLT Workshop at ECOOP'07, Berlin, 2007.

[2] Campbell, S. L., Chancelier, J. Ph., and Nikoukhah, R., “Modeling and Simulation in Scilab/Scicos”, Springer, 2005.

[3] Caspi, P., Curic, A., Maignan, A., Sofronis, C. and Tripakis, S., "Translating Discrete-Time Simulink to Lustre”, in Embedded Software, Lecture Notes in Computer Science, Springer, 2003.

[4] Otter, M., Elmqvist, H. and Mattsson, S. E., “Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle”, CACSD’99, Aug; 1999, Hawaii.

[5] Nikoukhah, R., “Extensions to Modelica for efficient code generation and separate compilation”, in Proc. EOOLT Workshop at ECOOP’07, Berlin, 2007.

[6] Modelica Association, “Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, version 3.0”, 2007, available from **www.modelica.org**.

Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters

Farid Dshabarow
ABB Turbo Systems AG
Switzerland
Farid.Dshabarow@CH.ABB.Com

François E. Cellier, Dirk Zimmer
ETH Zürich
Switzerland
{FCellier,DZimmer}@Inf.ETHZ.CH

Abstract

This paper introduces a new Modelica library for modeling and simulation of systems with distributed parameters in one space dimension. The resulting partial differential equations of either the parabolic or hyperbolic types are being converted to sets of ordinary differential equations using either the method of lines or the finite volume approach. Some simple examples serve to document the utilization of the new library.

Keywords: *Distributed Parameter Systems, Numerical PDEs, Method of Lines, Finite Volume Method*

1 Introduction

1.1 History of General-purpose PDE Solvers

Lumped parameter systems have been successfully modeled and simulated using general-purpose simulation software for several decades. With the advent of Modelica, it has become unnecessary to model and simulate any physical systems with lumped parameters using either general programming languages, like C++, or special-purpose simulation languages, like Adams or Spice. Modelica is capable of converting any lumped parameter model of a physical system to executable code that is as efficient in its execution as the best manually coded spaghetti programs of the past. Modelica can also successfully compete with special-purpose simulation codes, like Spice or Adams, in the simulation of electronic circuits [5] and multi-body systems [15].

The modeling and simulation of distributed parameter systems using general-purpose simulation software has not been as successful. In the 70s and early 80s, a number of general-purpose simulation codes, like FORSIM VI [4], were developed for the purpose of modeling and simulating at least some classes of systems with distributed parameters. FORSIM VI, for example, was designed for simulating parabolic PDEs in one or two space dimensions. Hyperbolic PDEs could be simulated as well, but the resulting simulation code was not as efficient. Elliptic PDEs could sometimes be converted to equivalent parabolic problems using invariant embedding.

Around the same time frame, another program, ELLPACK [11], was developed that was designed for solving elliptic PDEs in two or three space dimensions. The ELLPACK project was very ambitious, and the code grew rapidly to a size that made the code difficult to use and maintain. In order to make ELLPACK easier to use, the designers of the code developed a preprocessor for translating an abstract model description down to a set of Fortran subroutine calls. Yet, as new algorithms were added constantly to the software, maintenance of the preprocessor became soon too difficult. Hence a compiler-compiler was developed that could be used to generate a new version of the preprocessor from an abstract description thereof. Yet in spite of all of those efforts, the resulting simulation programs were highly inefficient at run time.

Whereas one of the primary mantras of modeling and simulation environments is to be able to protect the user from having to fully understand the numerical properties of the underlying solver algorithms, this demand could never be fully satisfied when dealing with PDEs. The run-time efficiency of the resulting simulation code depends too heavily on the chosen discretization method, and no logic was found that could relieve the user from having to make hard choices manually.

Sometimes codes like ELLPACK have been used to quickly try out different combinations of algorithms and compare them with each other. In this way, the user could more quickly determine, which combination of algorithms might work best. However, once this decision has been reached, the final code nevertheless had to be hand-coded, because the real problems were not limited to the PDE solvers themselves, but more often than not were related to how the code dealt with complex geometries, i.e., how physical boundary conditions were converted to boundary conditions that the PDE solver could make use of [14].

For all of these reasons, the use of general-purpose simulation software for the simulation of distributed parameter systems became unfashionable again. The researchers dealing with these types of systems simply gave up, and most of today's simulation codes are specially designed codes for very small classes of problems only.

1.2 A Renaissance for General PDE Solvers

One technique that has proven to be more robust than other approaches is the finite element method [9]. The success of this technique is based on its ability for dealing effectively with complex geometries. Originally developed for simulating elliptic 2D and 3D problems, finite element methods have quickly also been adapted to the discretization of parabolic and hyperbolic PDEs [12].

FEMLAB is a general-purpose numerical PDE solver based on the finite element method. FEMLAB was developed in recent years for the simulation of multi-physics applications. The code is capable of simulating models involving multiple PDEs [13].

FEMLAB started out as a MATLAB toolbox. Yet, its developers learnt quickly the same truth that the ELLPACK developers had learnt before them: a general-purpose PDE solver becomes soon unmanageable without a preprocessor capable of interpreting an abstract model definition. They also learnt that they needed to offer CAD support for entering the device geometry.

FEMLAB was more successful than ELLPACK, in part, because the computers have meanwhile become faster, and in part, because they were less ambitious in the sense that they didn't insist on incorporating each and every algorithm that has ever been developed for the numerical solution of PDEs.

FEMLAB has recently changed its name to COMSOL. This software represents currently the gold standard of general-purpose numerical PDE solvers for multi-physics applications.

1.3 A Role for Modelica?

Modelica has become the de facto standard for modeling and simulation of physical systems with lumped parameters. Does it have a role to play in numerical PDEs also?

Modelica, or rather its implementations, such as Dymola, offer not much that is unique or special w.r.t. their simulation engine. The only feature worth mentioning in this respect is a fairly robust root solver (discontinuity handler). The true power of Modelica lies in its ability to deal with differential and algebraic equations (DAEs) in a very flexible and truly object-oriented manner.

Today's numerical PDE solvers, including COMSOL, offer numerically advanced algorithms, but are very primitive w.r.t. their user interface. The complexity and elaboration of the user interface is at approximately the same level that the Continuous System Simulation Languages (CSSLs) were prior to the advent of the CSSL standard [2].

Would a language like Modelica have made a big impact in the 1960s, had it been available? The answer to this question is no. The computers of those times were far too small and too slow to adequately host a language like Modelica. The researchers of those days dealt with much simpler models, models that could be handled by the tools available to them, not because they lacked a better understanding of physics, but simply, because their computers couldn't handle more complex models.

Are distributed parameter problems structurally simpler than lumped parameter problems? The answer to that question is also no. Physics in general deals with 3D fields, and lumped parameter models are simply abstractions of distributed parameter problems.

If we wish to bend a pipe, we first heat up the area where the pipe is to be bent. If we were to simulate the physics of bending a pipe, we would have to solve a 3D distributed parameter problem with one PDE describing the heat diffusion problem and another PDE describing the mechanical stresses and strains within the material. These PDEs would furthermore have to be solved in a geometry that changes over time as a function of the numerical solution of the two PDEs.

Researchers aren't currently simulating such processes, they aren't dealing with partial differential and algebraic equations (PDAEs) yet, because the computers of today are too small and too slow to adequately handle such problems.

Yet, it is not too early to ponder about the language constructs and numerical algorithms that will be needed in support of such endeavors, once the computers shall have advanced to a level, where they can deal with such models effectively and efficiently.

2 PDELib for 1D Numerical PDEs

Since the Standard Modelica Library doesn't offer any support yet for modeling distributed parameter systems, we decided to take a first, and very modest, step towards the much larger and more grandiose aim outlined in the introduction.

To this end, we revisited some of the programs of the past, in particular FORSIM VI, and decided to re-implement some of the algorithms and capabilities offered by FORSIM VI in a Modelica experimental library. The results of that effort are being presented in this paper.

In order to keep things simple, we decided to limit the tool to the numerical solution of parabolic and/or hyperbolic PDEs in a single space dimension, the class of 1D numerical PDEs.

Since 1D PDEs are solved on a straight line between point A and point B, the geometry plays no role yet in these problems. The spatial discretization is straightforward; finite elements aren't needed or even useful yet for the spatial discretization; and the resulting simulation code can still be simulated fairly efficiently and rapidly using almost any half-way suitable numerical algorithm.

The aim of the project was to create an experimental tool that can be used to study some properties of numerical PDEs that haven't received much coverage yet in the open literature.

One of the numerical problems to be studied is the propagation of discontinuities through a 1D hyperbolic PDE. Such discontinuities cause a new class of numerical problems. Once the discontinuity has reached the boundary condition of the PDE, it can no longer be isolated in time. At any moment in time, the discontinuity exists somewhere within the spatial domain covered by the PDE. Thus, traditional event handling cannot be used to deal with this type of discontinuities.

A structural problem to be studied concerns the numerical solution of 1D PDAEs. Can Modelica help in translating a 1D PDAE into a simulation code that can be simulated effectively and efficiently?

Two algorithms were implemented in the first official release of PDELib: the method of lines [6], and a dialect of the finite volumes approach [10].

3 Method of Lines

Given the 1D diffusion equation:

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \qquad (1)$$

The method of lines discretizes the spatial derivatives, while keeping the temporal derivatives continuous. In a first approximation, we may write:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \qquad (2)$$

Plugging Eq.(2) into Eq.(1), we find:

$$\frac{\partial u_i}{\partial t} \approx \sigma \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \qquad (3)$$

In this way, we have converted a PDE into a stiff set of ODEs that can now be simulated using any off-the-shelf stiff ODE solver, such as DASSL.

The method of lines is fairly easy to implement. The chosen approximation is third order accurate. If the user wishes to use a more accurate approximation formula, this can be done easily.

Care must be taken in a correct implementation of the boundary conditions. As the discretization approaches the boundary, biased discretization formulae in place of central formulae must be used in order not to make use of grid points outside the simulated domain.

The approach works fairly well, especially in the case of parabolic PDEs such as the diffusion equation. The spatial discretization of a parabolic PDE by means of the method of lines leads invariably to a stiff set of ODEs, but modern numerical ODE solvers are good at dealing with those.

This is the approach that FORSIM VI took. In order to relieve the user of having to remember different discretization formulae, FORSIM VI offered a set of Fortran subroutines for computing spatial derivatives both in the bulk and in the vicinity of the domain boundaries.

PDELib also hides the details of the discretization formulae from the user, but does so using a Graphical User Interface (GUI) as shown in Fig.1.

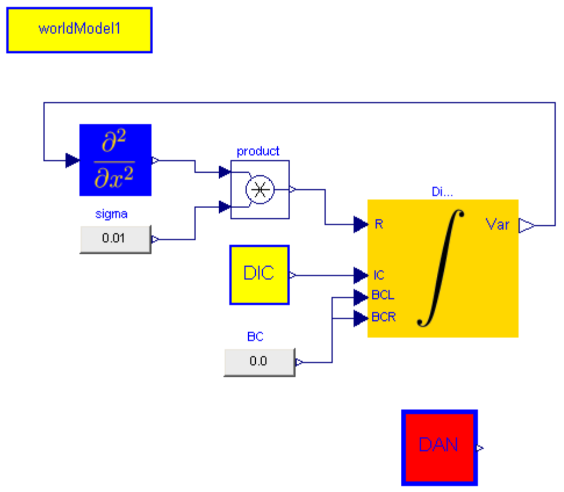


Figure 1: Model of 1D diffusion equation in PDELib

PDELib offers a method-of-lines (MOL) integrator block. This is a vector integrator block that integrates the vector of temporal derivatives, du_i/dt (marked as "R" on the integrator block), into the vector of states, u (marked as "Var"), while considering the vector of initial conditions (IC) as well as the left and right boundary conditions (BCL and BCR).

The blue box computes the spatial derivatives. In its parameter window, the user can select the approximation order to be used. Biased formulae of suitable approximation accuracy automatically replace the central formulae in the vicinity of the two domain boundaries.

The WorldModel box is used to provide general information, such as the grid width of the spatial discretization.

Since the diffusion equation with the chosen initial and boundary conditions has an analytical solution, that solution is also computed in the block DAN for comparison.

Simulation results are shown in Fig.2.

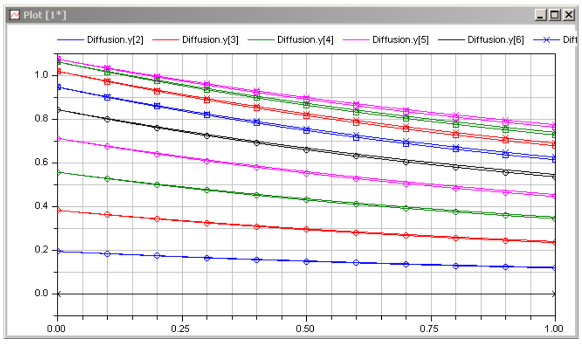


Figure 2: Diffusion equation simulation results

Since Dymola hasn't been designed for simulating PDEs, there is currently no support for 3D graphics in Dymola. The graph shows the temperature, u, at different space locations as a function of time.

The analytical results were superposed with the simulation results. In the simulation, the space was discretized into 40 segments of equal size. With 40 segments, the simulation results are still noticeably different from the analytical results.

The MOL approach is less well suited for dealing with hyperbolic PDEs, because their discretization leads to marginally stable ODE systems, rather than stiff ODE systems. Unfortunately, the numerical ODE solvers provided with Dymola and most other ODE simulators are not geared to accurately integrate marginally stable systems of ODEs.

The numerical condition of the model can sometimes be improved by using upwind discretization schemes [3]. In these schemes, the spatial derivatives are on purpose computed using biased formulae also in the bulk. FORSIM VI and PDELib offer optional upwind discretization schemes.

4 Finite Volume Method

Another discretization technique that has been successfully applied to numerically simulating hyperbolic PDEs is the Finite Volume Method (FVM). Just like the MOL technique, also the FVM approach comes in many different variants. Hence it may be useful to provide a toolkit that enables a user to compose a FVM from a set of component models.

In one space dimension, the FVM consists in subdividing the spatial domain into intervals, also called cells or finite volumes. The integral of the unknown function, u, is approximated over each of these cells at each time step. Let us denote the i^{th} cell by:

$$C_i = (x_{i-1/2}, x_{i+1/2}) \tag{4}$$

The average value of the function u over this cell is then:

$$U_i = \frac{1}{\Delta x} \int_{C_i} u(x,t)dx \tag{5}$$

How can we estimate the value of U_i? Considering the mass conservation law, we note that the average within the cell can only change due to fluxes at the boundaries, assuming that neither source nor sink is present in the cell. Mass conservation can be expressed mathematically in the following form:

$$\frac{d}{dt}\int_{C_i} u(x,t)dx = f(u(x_{i-1/2},t)) - f(u(x_{i+1/2},t)) \qquad (6)$$

where *f* denotes the flux function. The change of total mass inside the cell equals the flux entering the cell minus the flux leaving it.

Let us integrate Eq.(6) over time from *t* to *t*+*Δt* and divide the equation by *Δt* and *Δx*. We obtain:

$$\frac{1}{\Delta t\cdot\Delta x}\int_{C_i} u(x,t+\Delta t)dx - \frac{1}{\Delta t\cdot\Delta x}\int_{C_i} u(x,t)dx = \qquad (7)$$
$$\frac{1}{\Delta t\cdot\Delta x}\left(\int_t^{t+\Delta t} f(u(x_{i-1/2},t))dt - \int_t^{t+\Delta t} f(u(x_{i+1/2},t))dt\right)$$

Plugging Eq.(5) into Eq.(7), we obtain:

$$\frac{U_i(t+\Delta t)-U_i(t)}{\Delta t} = -\frac{1}{\Delta x}(F^t_{i+1/2} - F^t_{i-1/2}) \qquad (8)$$

where:

$$F^t_{i-1/2} = \frac{1}{\Delta t}\int_t^{t+\Delta t} f(u(x_{i-1/2},t))dt \qquad (9)$$

is the average flux over one time step.

We can reinterpret Eq.(8) as a discrete approximation of a differential equation:

$$\frac{d}{dt}U_i = -\frac{1}{\Delta x}(F^t_{i+1/2} - F^t_{i-1/2}) \qquad (10)$$

Using this simple trick, we have reduced also the FVM to a Continuous-Time/Discrete-Space (CTDS) method.

How do we approximate the average flux? Different approximations have been proposed. A simple approximation is the upwind flux:

$$F_{i-1/2} = \begin{cases} cU_{i-1}, & \text{if } c < 0; \\ cU_i, & \text{if } c > 0. \end{cases} \qquad (11)$$

i.e., the average flux across a border between cells during one integration step is proportional to the average value of *u* in the upwind cell.

5 Examples

In the following section of the paper, some of the models currently available as examples in PDELib are shown.

5.1 Linear Advection Equation

The advection equation is one of the simplest PDEs to be found. Given a constant speed, *c*, the linear advection equation can be written as:

$$u_t = -c\cdot u_x \qquad (12)$$

This problem was encoded using the MOL approach with the initial condition:

$$u(x,0) = cos(x) \qquad (13)$$

and with the boundary condition:

$$u(x_1,t) = -cos(-ct) \qquad (14)$$

applied at the left boundary of the domain. The MOL model is shown in Fig.3.

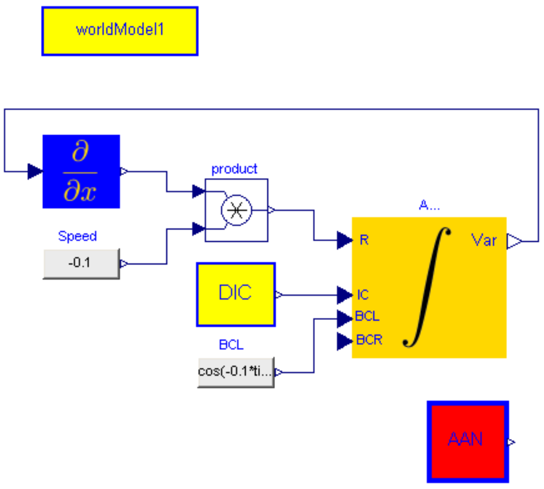


Figure 3: MOL model of linear advection equation

Some simulation results are shown in Fig.4.

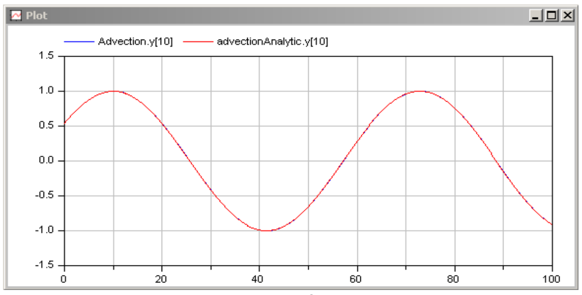


Figure 4: MOL simulation of linear advection equation

The same problem was also solved using the FVM technique with upwind flux computation. The model is shown in Fig.5.

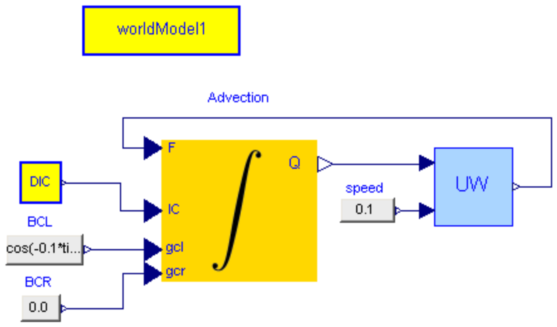


Figure 5: FVM model of linear advection equation

This model generates the simulation results shown in Fig.6:

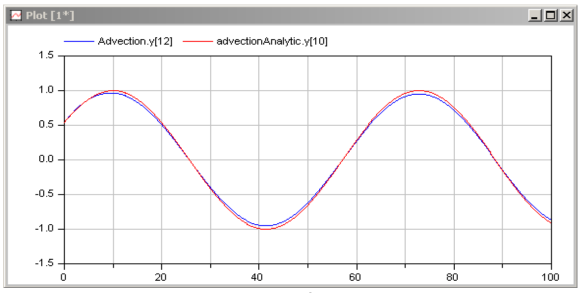


Figure 6: FVM simulation of linear advection equation

The index of the FVM solution is off by two segments due to the ghost cells used in this approach for computing the solution in the vicinity of the domain boundary [8].

5.2 Burger's Equation

The inviscid Burger´s equation is the non-linear PDE

$$u_t + (\frac{u^2}{2})_x = 0 \tag{15}$$

If we choose as initial condition:

$$u(x,0) = x \tag{16}$$

and as boundary conditions:

$$u(x_1,t) = 0 \tag{17}$$

$$u(x_n,t) = 0 \tag{18}$$

the problem has the analytical solution:

$$u(x,t) = \frac{x}{1+t} \tag{19}$$

The MOL implementation of Burger´s equation is shown in Fig.7.

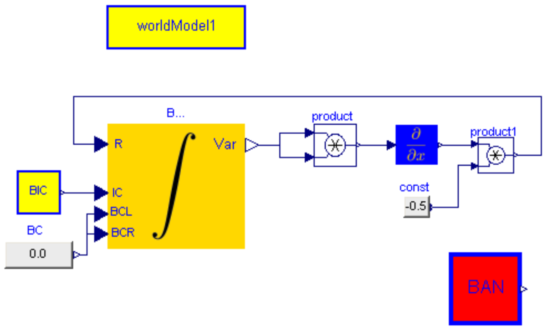


Figure 7: MOL model of Burger's equation

Some simulation results are shown in Fig.8.

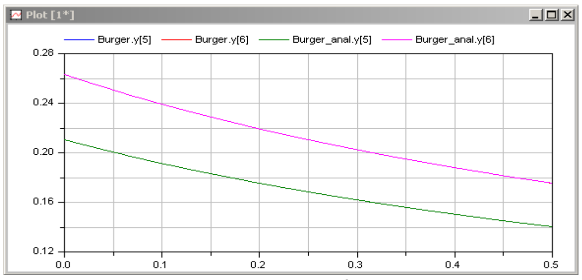


Figure 8: MOL simulation of Burger's equation

The results look excellent, but they are deceiving. The simulation here used 20 segments. Using 10 segments, the numerical results start deviating from the analytical results after only 0.2 seconds of simulated time. With 20 segments, the simulation results are more accurate, but the numerical simulation turns unstable after roughly 0.6 seconds. The more segments are being used, the faster the simulation becomes numerically unstable.

An FVM implementation of Burger´s equation is shown in Fig.9.

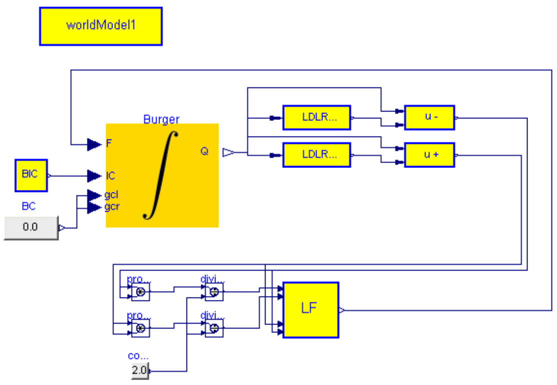


Figure 9: FVM model of Burger's equation

In this example, the FVM implementation uses a Lax-Friedrichs flux together with Local Double Logarithmic Reconstruction (LDLR) [1,7,10].

Some simulation results are shown in Fig.10.

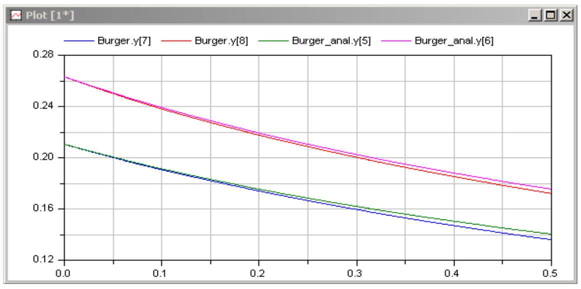


Figure 10: FVM simulation of Burger's equation

The FVM simulation remains numerically stable independent of the number of cells in use. Unfortunately, the results obtained are less accurate than using the MOL approach. The indices are again off by two because of the ghost cells.

6 Conclusions

What have we accomplished? We have been able to create an experimental library that enables experienced analysts to quickly try out different combinations of algorithms that can be used for the simulation of 1D parabolic and hyperbolic PDEs. Yet, we have failed in our aim to protect the user from having to understand the numerical properties of PDE solvers.

We chose a mathematical rather than a physical interface to our library, because it makes the tool more flexible and more general in its applicability. However, it was precisely that decision that made us fail in our endeavor of delivering a tool to the end user that can be applied blindly and reliably. This simply cannot be done at a mathematical level.

Yet, this is not a major problem. Modelica, due to its object-oriented philosophy, is good at information hiding. In the future, we shall be able to place a physical layer on top of the mathematical layer that offers solutions to particular subsets of PDEs, just as COMSOL does. Each physical module then decomposes its models internally into a combination of modules programmed at the mathematical layer.

We chose a blocks philosophy for our library. Each mathematical model is composed as a block diagram. In the long run, this decision will prove to have been a mistake. We shall need to learn to trust Modelica to make the right causality decisions for us. Otherwise, we shall never be able to solve PDAEs.

We had made the same mistake initially in the design of MultiBondLib [15], our multi-bond graph library. Initially, we formulated holonomic constraints between bodies using blocks from the Blocks library. If we use an adder:

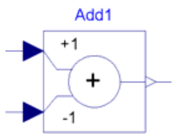


Figure 11: Adder of the Blocks library

from the Blocks library, we force Modelica to compute $y = u_1 - u_2$, but maybe the correct causality ought to be $u_2 = u_1 - y$. By using blocks from the Blocks library, we are tying Modelica's hands unnecessarily, which may lead to situations, where Modelica can no longer find a solution to the problem.

Yet for the time being, the decision to program PDELib using blocks rather than models helped us restrict the sources of errors. During the initial phase of the research, the phase of determining the most suitable numerical algorithms, the use of blocks may be a good thing.

Finally, Dymola doesn't offer any support yet for 3D graphics. Although it is possible to export simulation results to MATLAB and produce 3D graphics using that software, this is a hassle. Dynasim should develop a 3D graphics package that can be used to plot vectors of variables against time. The package should furthermore be tied to the 3D View Control window to give the users an opportunity to look at their 3D graphs from different angles.

References

[1] Artebrant, H., Schroll, J., Limiter-free Third Order Logarithmic Reconstruction. *SIAM Journal on Scientific Computation* 28 (2006) 359-381

[2] Augustin, D.C, Fineberg, M.S., Johnson, B.B., Linebarger, R.N., Sansom, F.J., Strauss, J.C.: The SCi Continuous System Simulation Lanugage (CSSL). *Simulation* 9 (1967) 281-303

[3] Carver, M.B., Hinds, H.W.: The Method of Lines and the Advective Equation. *Simulation* 31 (1978) 59-69

[4] Carver, M.B., Stewart, D.G., Blair, J.M., Selander, W.N.: *The FORSIM VI Simulation Pack-*

age for the Automated Solution of Arbitrarily Defined Partial and/or Ordinary Differential Equation Systems. Atomic Energy of Canada, Ltd., Chalk River, Ontario, 1978

[5] Cellier, F.E., Clauß, C., Urquía, A.: Electronic Circuit Modeling and Simulation in Modelica. In: *Proceedings of the Sixth Eurosim Congress on Modelling and Simulation*, Ljubljana, Slovenia (2007) Vol. 2, 1-10

[6] Cellier, F.E., Kofman E.: *Continuous System Simulation*. Springer-Verlag, New York, 2006

[7] Díaz López, J.: Shock Wave Modeling for Modelica.Fluid Library Using Oscillation-free Logarithmic Reconstruction. In: *Proceedings of the 5th International Modelica Conference*, Vienna, Austria (2006) Vol.2 641-649

[8] Dshabarow, F.: *Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters*. MS Thesis, ETH Zurich, Switzerland, 2007

[9] Hughes, T.J.R.: *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000

[10] LeVeque, R.J.: *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2002

[11] Rice, J.R., Boisvert, R.F.: *Solving Elliptic Problems Using ELLPACK*. Springer-Verlag, New York, 1984

[12] Thomée, V.: *Galerkin Finite Element Methods for Parabolic Problems, 2nd Edition*. Springer-Verlag, Berlin, 1997

[13] van Schijndel, A.W.M.: Modeling and Solving Building Physics Problems with FemLab. *Building and Environment* 38 (2003) 319-327

[14] Wu, Q.M., Cellier, F.E.: Simulation of Bipolar High-voltage Devices in the Neighborhood of Breakdown. *Mathematics and Computers in Simulation* 28 (1986) 271-284

[15] Zimmer, D., Cellier, F.E.: The Modelica Multi-bond Graph Library. In: *Proceedings of the 5th International Modelica Conference*, Vienna, Austria (2006) Vol. 2, 559-568

Farid Dshabarow received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2007. He is now working at ABB Turbo Systems, where he deals with gas dynamics simulations in turbochargers and software for calculating and visualizing turbocharger characteristics.

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.

Dirk Zimmer received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. He gained additional experience in Modelica and in the field of modeling mechanical systems during an internship at the German Aerospace Center (DLR) in 2005. Dirk Zimmer is currently pursuing a PhD degree with a dissertation related to computer simulation and modeling under the guidance of Profs. François E. Cellier and Walter Gander. His research interests focus on the simulation and modeling of physical systems with dynamically changing structure.

Session 6c

Thermodynamic Systems & Applications

Simulation of Peak Stresses and Bowing Phenomena during the Cool Down of a Cryogenic Transfer System

Hubertus Tummescheit[†], Kristian Tuszynski[†] , Philipp Arnold[‡]
Modelon AB[†,] Ideon Science Park, SE-22370 Lund, Sweden
Linde Kryotechik AG[‡], CH-8422 Pfungen, Switzerland
Hubertus.Tummescheit@modelon.se, Kristian.Tuszynski@modelon.se

Abstract

An extension for cryogenic systems to the AirConditioning Library by Modelon was used to analyze the cool down of a cryogenic transfer system where Linde Kryotechnologie in Pfungen, Switzerland was the main contractor. Simulation was used early in the design process to make sure that the system was well designed for a number of cool-down scenarios. Early detection of problematic parts of the system for some cool-down sequences lead to changes in the piping design. Simulation was also used to assess the maximum thermal stresses during cool down and determine suitable mass flow rates. Proper cool-down sequences were established iteratively with the help of a combined simulation of the cryogenic two-phase flow, the heat conduction in solid structures and the resulting thermal stresses. The two main problems to avoid during cool down are (1) excessive thermal stresses in thick-walled components, and (2) bowing of pipes with liquid cryogen in the lower part of a long, horizontal pipe with gaseous cryogen above. Two similar systems where considered, one for liquid hydrogen, the other for liquid oxygen. Dymola and Modelica were choosen for the project due to the good multi-domain and multi-physics capabilities, and the availability of model libraries that covered a large part of the problem.

Keywords: Cryogenics, two-phase flow, transient thermal stress simulation

1 Introduction

The Indian Space Research Organization, ISRO, is building and commissioning a new cryogenic engine test rig in their Liquid Propulsion Test Centre in Mahendragiri, Tamil Nadu. The system under investigation is the cryogenic transfer system for the cryogenic fluids hydrogen and oxygen, used to transfer cryogen from tankers into the run-tanks and from both tankers and run tanks to the test objects. The system is designed for a wide range of pressures and flow rates which leads to a rather complex overall structure of pipes, valves and measurement equipment. Simulations of the system cool down was used early in the design process to validate the design – here the main issue is to avoid bowing of dead-end pipes – and to find improvement potential from an operational point of view. Simulation was also used later on to establish suitable cool-down flow rates and valve sequences that fulfill the two main requirements: use as little cryogen as possible for cool down while not exceeding the maximum allowed thermal stresses.

Obtaining the desired mass flow rates in a transient two-phase flow system throughout the system is very difficult because of the enormous change in densities between gaseous and liquid cryogen: the density ratio can be up to 1:1000. During the filling of the system with liquid, deviations between local mass flow rates and controlled rates at a valve with one-phase inlet conditions can be large. In the situations when the control valve is inside the two-phase region, actual mass flow rates can not be controlled at all.

2 Modeling of thermal stress in cylindrical bodies

The model for thermal stress is based on a radial discretization of cylindrical geometries both for pipes and valves. For the bowing phenomenon, also a tangential discretization and, if necessary an axial one are added. The energy balance of a cylindrical slice of the pipe is based on the Fourier equation with a central difference approximation of the temperature gradient and takes the temperature dependence of the heat capacity and thermal conductivity into account. Stresses are computed separately for the stress introduced through temperature gradients and the mechanical stress due to the pressure inside the pipe. The stress vectors are summed to compute a total equivalent stress. The equivalent stress reaches its maximum value either on the inside of the cylinder or on the outside of the cylinder. The ratio of the maximum equivalent stress and the yield stress is the *stress ratio*.

The Fourier equation is given by [7],

$$\frac{\rho_i}{\lambda_i}\left(\frac{\partial T_i}{\partial t}Cp_i+\frac{\partial Cp_i}{\partial t}T_i\right) = A_i\cdot T_{i-1}+B_i\cdot T_i+C_i\cdot T_{i+1} \qquad (1)$$

Where *i = 2, 3.. Nr-1* and *Nr* is the discretization number of the material in radial direction. The two remaining elements are given by the boundary conditions.

The calculation of the Fourier coefficients, *A*, *B* and *C* for a radial discretization is shown in equation *(2)*.

$$A_i=\frac{r_i+r_{i-1}}{r_i(r_i-r_{i-1})(r_{i+1}-r_{i-1})}$$

$$B_i=\frac{r_{i+1}+r_i}{r_i(r_{i+1}-r_i)(r_{i+1}-r_{i-1})} \qquad (2)$$

$$C_i=-A_i-B_i$$

The axial heat conduction in the material is assumed to be negligible.

To obtain the thermal stress distribution, three stress components in tangential (Θ), radial (r) and axial (z) directions are calculated. The general stress equations are given by

$$\sigma_\theta = \frac{E\cdot\alpha}{(1-\nu)r^2}\cdot\left[\frac{r^2+r_i^2}{r_o^2+r_i^2}\int_{r_i}^{r_o}T(r)r\,dr+\int_{r_i}^{r}T(r)r\,dr-T(r)\cdot r^2\right]$$

$$\sigma_r = \frac{E\cdot\alpha}{(1-\nu)r^2}\cdot\left[\frac{r^2-r_i^2}{r_o^2+r_i^2}\int_{r_i}^{r_o}T(r)r\,dr-\int_{r_i}^{r}T(r)r\,dr\right]$$

$$\sigma_z = \frac{E\cdot\alpha}{(1-\nu)}\cdot\left[\frac{2}{r_o^2+r_i^2}\int_{r_i}^{r_o}T(r)r\,dr-T(r)\right]$$

(3)

where *E* is the Young modulus, *α* the linear expansion coefficient and *υ* the Poisson ratio.

By only calculating the thermal stress at the inner and outer points of the wall (the maximum stress of a pipe is always at one of these points) the equations can be simplified as:

$$\begin{aligned}
\sigma_\theta^i&=\frac{E_1\cdot\alpha_1}{(1-\nu_1)}\cdot[Tm-T(r_i)]\\
\sigma_\theta^o&=\frac{E_{Nr}\cdot\alpha_{Nr}}{(1-\nu_{Nr})}\cdot[Tm-T(r_o)]\\
\sigma_z^i&=\frac{E_1\cdot\alpha_1}{(1-\nu_1)}\cdot[Tm-T(r_i)]\\
\sigma_z^o&=\frac{E_{Nr}\cdot\alpha_{Nr}}{(1-\nu_{Nr})}\cdot[Tm-T(r_o)]\\
\sigma_r^i&=0, \qquad \sigma_r^o=0
\end{aligned} \qquad (4)$$

Where *Tm* is the mean temperature of the material and 1 and Nr refer to the innermost and outermost radial discretizations respectively.

The effective stress according to Von-Mises theory results in (from [7]):

$$\sigma_{eff}=\sqrt{\sigma_\theta^2+\sigma_z^2+\sigma_r^2-(\sigma_\theta\cdot\sigma_r+\sigma_\theta\cdot\sigma_z+\sigma_r\cdot\sigma_z)}$$

(5)

The stress-ratio is defined as the ratio of effective stress to yield stress of the material:

$$\sigma_{ratio}=\frac{\sigma_{eff}}{\sigma_Y} \qquad (6)$$

The two different problems analyzed later in the paper need different discretizations.

1. The thermal stress analysis from cool-down requires a two-dimensional model with radial and axial discretizations to capture the local thermal stresses along the pipe.
2. The bowing problem requires axial and tangential discretizations to capture the different deformations on the top and bottom of a pipe where the bottom is filled with boiling liquid and the top is filled with saturated gas.

Both cases were captured with a single model with all 3 discretizations, where the ones that were not needed were set to one element.

3 Flow modeling

For the two-phase flow in the pipes, a standard finite volume method assuming homogeneous equilibrium flow was used as described in [2] and [3]. Due to the partially violent transients, a dynamic momentum balance has been used for some of the simulations. Heat transfer needs to take into account the „sub-cooled boiling" regime, which is important towards the end of the cool down and is present during a large fraction of the overall cooldown time. Pressure drop models are from the standard literature like [4]. Properties for oxygen were implemented according to [1], hydrogen properties according to [8], and the results were compared to RefProp by NIST which contains the same property models.

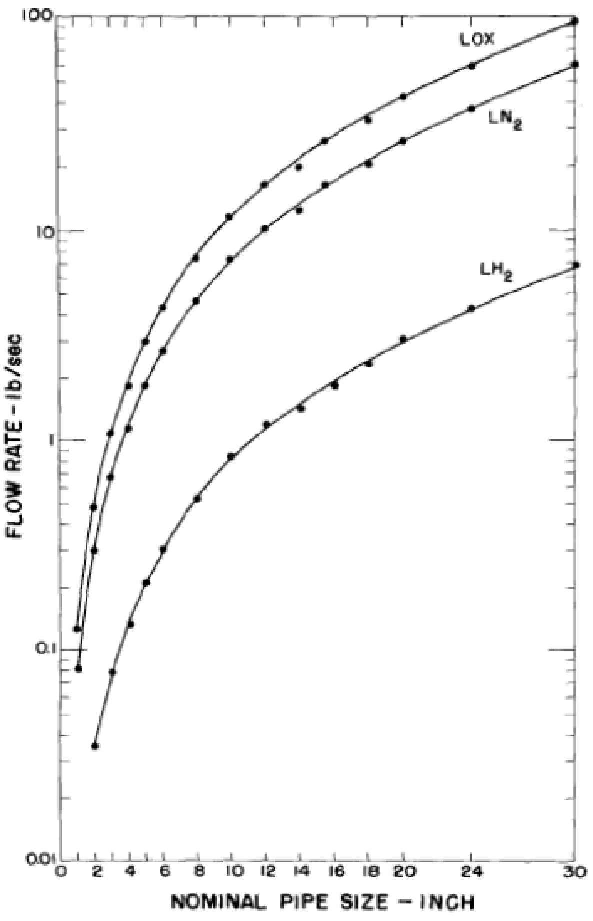


Figure 1: Flow rate which predicts non-stratified flow conditions for pipeline fluid qualities below95% (liquid and gas phase assumed saturated at boiling point), from [9].

The main trade-off that has to be taken into account is between minimal cryogen consumption for cool-down and a minimal cool down time. The mass flow is restricted by an upper limit, usually determined by the maximum allowable thermal stress, and a lower limit. The lower limit is defined by the „non-stratified flow" condition. A stable phase separation with liquid flow on the bottom of the pipe and gaseous flow above it results in differences in the heat transfer rate of about one order of magnitude. They may lead to faster cooldown on the bottom of the pipe, which may lead to bowing. The limit for stratified flow conditions for cryogens has been investigated in [9].

It could be argued that a homogeneous equilibrium model does not capture the physics of the cool down flow sufficiently accurate. For the main focus of the study, the thermal stress in the thick-walled components, it is not necessary to predict the flow and the flow-regime exactly (appart from avoiding stratified flow conditions), and therefore we do not believe that a non-homogeneous flow would improve the quality of the results in a way that would justify the much higher model complexity.

4 Low temperature properties

Both the thermal conductivity and the heat capacity of metal pipes go to 0 at 0 degrees Kelvin. This has a

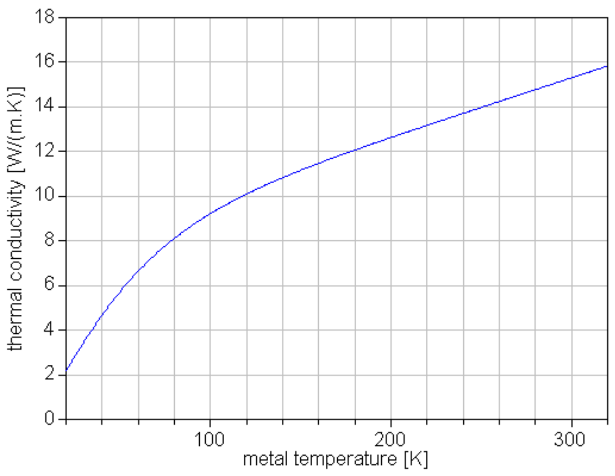


Figure 2: Thermal conductivity for steel 316L as a function of temperature.

number of surprising effects when the temperatures are approaching the lower limits (ca. 20 K for liquid hydrogen and ca. 80 K for liquid oxygen): the cold parts of metal pipes and valves almost insulate the remaining warmer parts from the cryogen, effectively slowing down the last part of the cool down.

Fortunately detailed data for metals used in cryogenic transfer systems is publicly available from NIST (National Institute for Standards and Technology) via their web-based database, see [8].

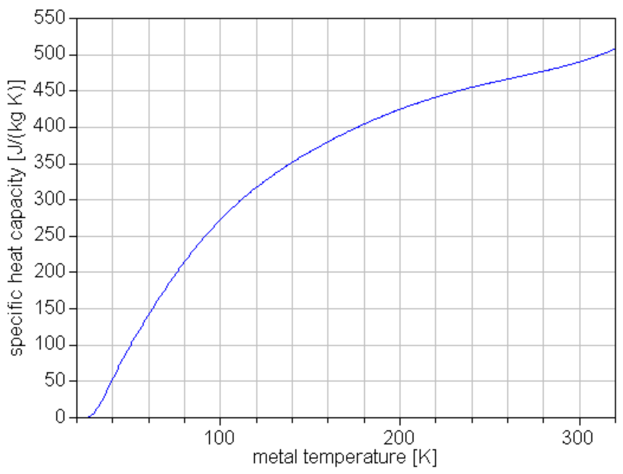


Figure 3: Specific heat capacity for steel 316L as a function of temperature.

5 Thermal stress results

For the evaluation of thermal stresses, a 10 m long pipe with ca 150 mm diameter (DN 150) and ca 8 mm wall thickness, material stainless steel 316 for oxygen and 304 for hydrogen, directly downstream of an open-close valve that opens completely in 2 seconds is investigated. The results for pipes give a good understanding for the situation of the complete system as they demonstrate well the differences caused by the different cryogen properties The upstream properties are:

1. Liquid oxygen of 0.5 MPa at 91 K.
2. Liquid hydrogen of 0.5 MPa at 21 K.

In both cases, the highest stress is not directly down

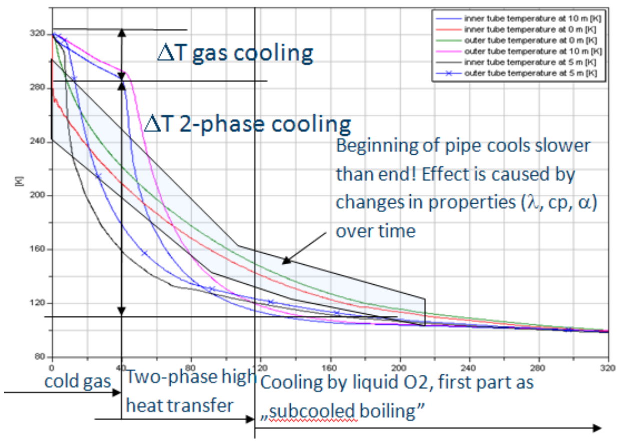


Figure 4: Temperature over time with different phases for a 10m high-pressure pipe during cool down with an oxygen mass flow of 1.7 kg/s at control valve.

stream of the valve but a short distance into the pipe, at a location where the combination of high heat transfer coefficient and large ΔT results in the combination with the highest heat flow. The longitudinal discretization is 20 segments, the radial discretization 10 segments for pipes.

Different phases of cooldown can be clearly distinguished from the temperature trajectories. The difference between hydrogen and oxygen cool down is also striking, but becomes understandable once the influences of the different thermophysical properties of the fluids and the metals are taken into consideration. Some of the results are not entirely intuitive, e.g. that the first part of the pipe has initially lower temperature than the downstream parts, but is the last part to be cooled down entirely.

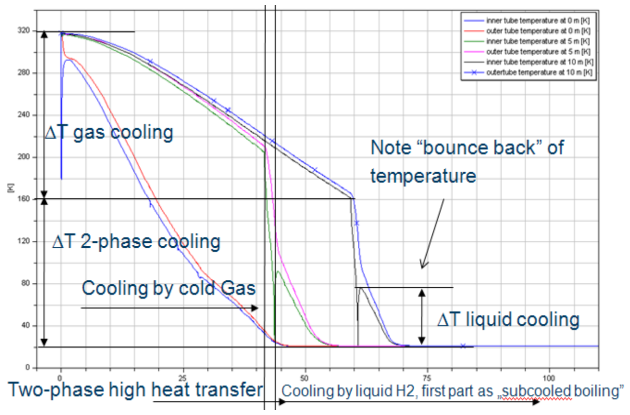


Figure 5: Temperature over time with different phases for a 10m low-pressure pipe during cool down with a hydrogen mass flow of 0.25 kg/s at control valve.

This is explainable from the change of the heat transfer coefficients over time/temperature: obviously the first part of the pipe is cooled down faster at the beginning, but the combination of a cold wall (metal conductivity decreases with temperature) and a low heat transfer coefficient (the beginning of the pipe is exposed to single phase liquid flow at very low Reynolds numbers). This fact, combined with further results omitted here, leads to the result that the cool down time is independent of the pipe length for pipe lengths less than 50 m. The differences between oxygen and hydrogen cool down become clear when looking in more detail at the required energy for the metal cool down and the available specific enthalpy differences for cooling in different phases, tabulated in Table 1. It is obvious that a much larger part of the cooldown is between gas phase and metal for hydrogen, both due to the larger energy content and the larger temperature difference. The gas phase cool down has a lower heat transfer coefficient which leads to lower stress peaks in the material. Secondly, the rapid cool-down with two-phase flow mostly happens after efficient pre-cooling with cold gas. Overall and against first intuition, cooling down with oxygen poses higher risks in spite of the lower temperature difference. Note also the temperature „bounce-back" of the metal layer in contact with the hydrogen after the hydrogen in the pipe changes from two-phase to liquid. This effect is caused by the drastic drop in heat transfer coefficent in the presence of much warmer outer layers in the pipe metal.

Material / phase	**Energy content for complete cooldown (from 318 K to 80 K for O_2, 20K for H_2)**
Steel 304	101.5 kJ/kg
Steel 316	94.6 kJ/kg
Total Δh H_2	4158 kJ/kg
Total Δh O_2	391.2 kJ/kg
Δh $H_{2\,evap}$	373 kJ/kg
Δh $H_{2\,gas}$	3785 kJ/kg
Δh $O_{2\,evap}$	191.2 kJ/kg
Δh $O_{2\,gas}$	200 kJ/kg

Table 1: Integrated energy content comparison

The largest source of uncertainty in the evaluation of the stress ratio is the occurrence of the „boiling crisis" in two-phase heat transfer at very high heat flow rates. Under such conditions, a thin layer of gas at the metal wall separates the boiling liquid from the metal by an insulating layer, thus drastically reducing the heat flow and the resulting thermal stresses. Correlations for the occurrence of the boiling crisis for cryogenic fluids are not very reliable, data only exists for non-cryogenic fluids. In addition, the boiling crisis condition for cryogenic cooling occurs at (almost) constant temperature of the hot side, which is different form the usual experiments with rapid heating and rising temperature on the hot side.

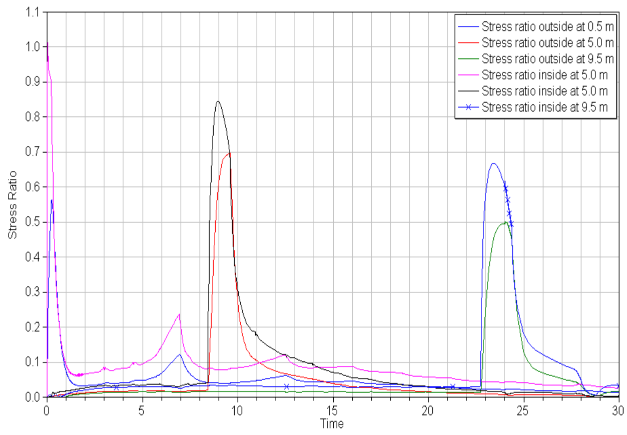


Figure 6: Stress ration along oxygen pipe directly after valve without any pre-cooling.

While this means that the exact heat transfer in the vicinity of the boiling crisis is difficult, the existing correlations can nonetheless be used to estimate the highest reasonable heat transfer coefficient and thus the worst case scenario for the thermal stresses in the metal wall. The results in Figure 6 for a high pressure pipe show that the combination of worst case assumptions (first segments of pipe that is subject to two-phase heat transfer from the start and high coefficinent of heat transfer) lead to stress ratios close to the permissible limit. The stress ratio plot in Figure 6 also shows that locations further downstream are subject to lower stress due to pre-cooling with cold gas. The stress peaks widen and the level decreases as the two-phase zone widens further downstream. A sensitivity study was conducted with repect to the most important parameters for the stress calculation, among others, the heat transfer coefficient, and the result was that the maximum heat transfer coefficient had a negligible effect on the stress ratio. For valves, due to the much thicker metal walls, the stress ratio exceeds 1.0 locally and for brief times. Cryogenic valves survive these conditions, but the high thermal stress leads to local deformations and „cold hardening", but is far from values that would cause complete material failure. While it is not possible to avoid these conditions everywhere in the system, the operation of the plant can be adapted to minimize the number of times and locations that are subject to the extreme conditions. It was, however, possible to avoid the severe thermal stress conditions for valves in the high pressure part of the system.

6 Pipe Bowing

The calculation of the pipe bending due to the temperature difference at the top and bottom of the pipe, when filling with cold liquid, is done with the following assumptions:

- Pipes are considered straight,
- The pipes are fixed at the lower end points with a gliding support at one end to compensate for longitudinal length change,
- Both radial and circumferential heat transfer is taken into consideration in the wall, axial heat transfer is neglected due to axial symmetry.

The liquid cross section area in the pipe is calculated according to *(1)*:

$$A_{liq} = acos\left(1 - \frac{L_{liq}}{r_{pipe}}\right) r_{pipe}^2 + (L_{liq} - r_{pipe}) \cdot \sqrt{2 \cdot r_{pipe} \cdot L_{liq} - L_{liq}^2} \quad (1)$$

The liquid volume is computed from the mass flow into the horizontal pipe, assumed to end at a closed valve. The mass flow into the pipe is taken from a prior cooldown simulation, at the position of the horizontally connected pipe.

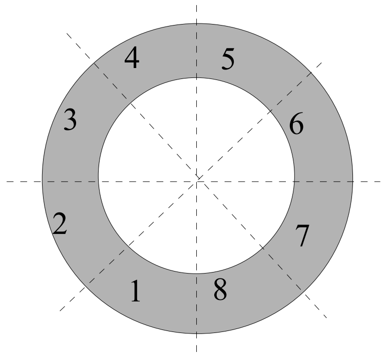


Figure 5: Circumferential discretization of the wall (defined by user)

Simulations use two heat transfer coefficients, one for the part of the wall that is in contact with liquid and the other for the part that is in contact with the gas. The gas temperature has very little influence on the overall result because of the low heat transfer coefficient between gas and pipe wall. Due to the boiling liquid underneath it will within short time after liquid cryogen is at the bottom reach the saturation temperature.

To calculate the heat transfer to the wall the actual liquid level is used to find the length of a discretization that is covered by liquid (if any) and the heat transfer is proportional to this value:

$$\dot{Q} = k_{liq} * (T_{wall} - T_{liq}) \cdot A_{HT} \cdot \frac{L_{liq} - L_{Dbottom}}{L_{Dtop} - L_{Dbottom}} + k_{gas} \cdot (T_{wall} - T_{gas}) \cdot A_{HT} \cdot \left(1 - \frac{L_{liq} - L_{Dbottom}}{L_{Dtop} - L_{Dbottom}}\right)$$

where A_{HT} is the heat transfer area, $L_{Dbottom}$ is the length from the bottom of the pipe to the lower boundary of a discretization, L_{Dtop} is the length from the bottom of the pipe to the top boundary of a discretization, k_{liq} is the heat transfer coefficient when in contact with liquid and k_{gas} is the heat transfer coefficient when in contact with the gas. Note that the weighted heat transfer area is a linearization of the inner pipe area fraction around the middle of a circumferential section and should thus only be used for relatively high discretization (16 were considered sufficient).

When calculating the pipe bending only the length change at the top (element 4 and 5 in Figure 5) and bottom (1 and 8 in Figure 5) of the pipe is taken into consideration. The length change is calculated through:

$$\Delta L = L \cdot \alpha (T_{wall}) \quad (4)$$

where α is the linear expansion coefficient of the material.

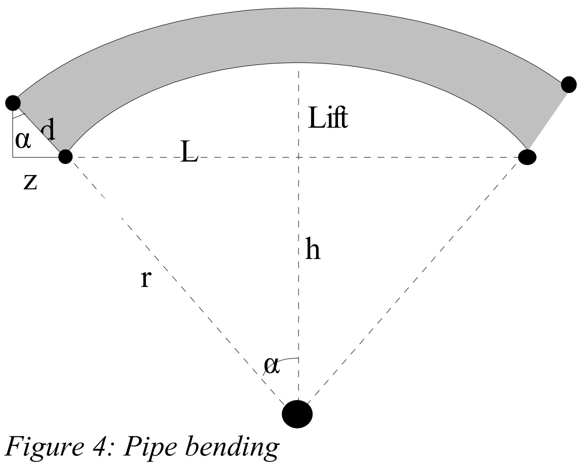


Figure 4: Pipe bending

$$Lift = r - h \quad (5)$$

$$r = \frac{L/2}{\sin(\alpha)} \quad \text{and} \quad \sin(\alpha) = \frac{z}{d} \quad (6)$$

$$h = \frac{L/2}{\tan(\alpha)} \tag{7}$$

(5), *(6)* and *(7)* yields,

$$Lift = \frac{L}{2}\left(\frac{d}{z} - \frac{1}{\tan(\alpha)}\right) \tag{8}$$

If the lift would reach high values of several centimeters, the influence of the lift on the local level and heat transfer would have to be taken into account, but such values are outside of the permissible range anyways.

The worst case encountered in the final modified version of the plant diagram was for a dead end of slighly less than 4 m length and a filling time form empty to full of about 11 minutes. The worst lift was 2.8 cm, a tolerable amount, and the worst case stress ratio using an equivalent stress from the full three-dimensional stress tensor was around 0.45. The length of the pipe has the worst effect on bowing as it effects both the geometry and the exposure time, and dead end pipes longer than 4 m would quickly cause inacceptable bowing.

7 Computational effort

For cooldown scenarios of the larger plant segments, the computational effort was very high: for the most complex segments of the plant cooldown, CPU-times of 3-4 days were necessary for each simulation case, and most of the work is spent during the first few seconds of simulation time. Dymola's version of the dassl solver only managed to survive the initial time without error when the option „equidistant output grid" was switched off. This in turn lead to result files of around 1GB that could not be handled by Dymola and made postprocessing very tedious. Overall, for system simulations of the level of complexity encountered during the cool down simulations, We see the following tool requirements for large scale system simulations with short periods of very sharp gradients:

1. A fine grained control over how many variables are stored and how often they are stored that does not influence the step-size control algorithm. Dymola's Dassl is a bad example of a solver that takes the storage interval into account in a way that lets simulations fail for a small step size to storage interval ratio.
2. Means to influence step size control during extreme gradients under short time, or set a minimum step size and get warnings in the log when the requested accuracy was not achieved.
3. Improved numerical debugging facilities. Dymola's current debugging facilities for numerical problems in large models are insufficient.

8 Conclusions

Modelica is not primarily known for its strength for modeling partial differential equations, but due to its suitability for system level simulations, there are situations in which Modelica and Dymola are an excellent tool even for models that require a full 3-dimensional PDE discretization, under the contraint that this only works for simple geometries. In particular the heat conduction equation with its simple structure can be combined with 1-dimensional two phase flow for thermal stress calculations. The key advantage is that it is possible to capture the most critical thermal stress situation within a complex plant without the need to resort to co-simulation, or difficult to assess assumptions.

This simulation study regarding cool down of a cryogenic transfer system was able to achieve a number of goals, in part because simulation was used already early in the design process:

1. It was possible to establish design guidelines regarding dead pipe ends at closed branches of the network to avoid pipe bowing. The guidelines were incorporated in later revisions of the design.
2. Flow rates were optimized with respect to the contradictory goals of minimum cryogen consumption and avoidance of stratified flow conditions.
3. Simulation results allowed to devise cool down sequences that substantially decreased the thermal stress for all parts in the plant except the parts closest to the tanker used for filling.

There are situations in which there is no possibility to validate simulations against measurements. In spite of that shortcoming, simulation gives important insight into system behaviour and even allows to improve both system design and system operation. Even quantitative analysis is possible to a certain degree when important parameters are well understood and a careful sensitivity analysis is conducted with respect to such parameters.

Cryogenic plant simulations, even under the violent transients that occur during cool down of transfer lines, can be modeled easily with the cryogenic option of the AirConditioning Library.

References

[1] **Span, R.,** Multiparameter Equations of State. An Accurate Source of Thermodynamic Property Data, Springer, Berlin, 2000.

[2] **Tummescheit, H.:** *Design and Implementation of Object-Oriented Model Libraries using Modelica*, Dissertation, TFRT-1063-SE, Department of Automatic Control, Lund Inst. of Technology, Lund, Schweden, 2002.

[3] **Tummescheit, H., Eborn, J.** und **Prölß, K.:** AirConditioning – a Modelica Library for Dynamic Simulation of AC Systems, in P*roceedings of the 4th International Modelica Conference*, Hamburg, pp. 185 – 192, 2005.

[4] **VDI-Gesellschaft Verfahrenstechnik und Chemiewesen** (Editor), VDI-Wärmeatlas, 9th Edition, Springer, Berlin, 2002

[5] **Versteeg, H. K. and Malalasekera, W**., An Introduction to Computational Fluid Dynamics – The Finite Volume Method, Prentice Hall, 1995.

[6] **Lemmon, E.**, The RefProp User manual, version 7.1, Personal communication See also information on version 7.0 at http://www.nist.gov/srd/nist23.htm (accessed 2005-11-15).

[7] **Fauple J.H.**, **Fisher F.E.**, Engineering Design-A Synthesis of Stress Analysis and Material Engineering, Wiley, New York, 1981.

[8] http://Cryogenics.nist.gov. Accessed October 2005.

[8] **Younglove, B.A.**, Thermophysical Properties of Fluids. I. Argon, Ethylene, Parahydrogen, Nitrogen, Nitrogen Trifluoride, and Oxygen, J. Phys. Chem. Ref. Data, Vol. 11, Suppl. 1, pp. 1-11, 1982.

[9] **D.H. Liebenberg, J.K. Novak, F.J. Edeskuty**: Cooldown of Cryogenic Transfer Systems, AIAA Paper No. 67-475.

Enhancement of a Modelica Model of a Desiccant Wheel

Andreas Joos* Gerhard Schmitz† Wilson Casas
Hamburg University of Technology
Institute of Thermo-Fluid Dynamics,‡ Applied Thermodynamics
21071 Hamburg, Germany

Abstract

This paper presents a MODELICA model for a desiccant wheel. Desiccant wheels are used in new concepts for air conditioning systems, which can save primary energy in contrast to conventional systems. This model is based on a model, which was presented at the MODELICA Conference 2005 [2], however in this study the model is improved with a new modeling approach to represent the wheels rotation. This structural change made the model faster and able to produce continuous output in contrast to the one of *Casas et al.*, [1, 2]. This was an essential step to enhance long term simulations of desiccant systems and control strategies. These simulations are necessary to optimize such systems and to evaluate their primary energy consumption.

Keywords: Modelica; Simulation; Desiccant Wheel; Air Conditioning; Sorption

1 Introduction

In desiccant air conditioning systems, moist air is dehumidified by means of a desiccant wheel, see figure 1. Water vapor is absorbed by desiccant material as humid air passes through the wheel. Using this technology, considerable energy savings can be obtained compared to conventional air conditioning systems. In [1] a model library has been developed to evaluate the performance of the desiccant assisted air conditioning process, so that different configurations and system concepts can be easily realized. Because it is necessary to simulate a period of a year to evaluate an air-conditioning concept, fast, dynamic models with a good accuracy are required. All these requirements argue for MODELICA as modeling language. The main and most complex component of this library is the model of the desiccant wheel.

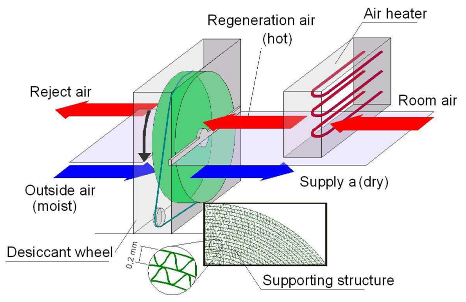


Figure 1: Example of an air-conditioning using a desiccant wheel

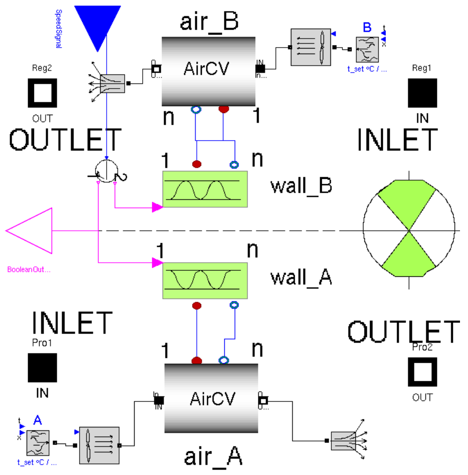


Figure 2: Schema of the old modeling approach [2]

Early approaches for numerical models can be found in [3, 7, 8]. These model formulations have the disadvantage in that they can not handle desiccant materials with discontinuities in their sorption isotherm (e.g. lithium chloride, LiCl). In [2] a MODELICA model is introduced to overcome those limitations, see figure 2 for an overview.

This model is discretized in such a way, that a system of ordinary differential and algebraic equations is generated, which can be easily re-configured for different set-up's. Also new relations for further sorption

*email: andreas.joos@tu-harburg.de, Tel:+49 40 42878 3079
†email: schmitz@tu-harburg.de, Tel:+49 40 42878 3144
‡www.tt.tu-harburg.de

isotherms can be provided without much effort. But due to the modeling approach of the rotation movement of the desiccant wheel through the two airflows, every half revolution time a state event in the numerical solver is caused. Thereby the maximal step size of the solver is restricted. This approach is also only valid, if the change of the boundary conditions of the desiccant wheel is insignificant in half a revolution cycle. Considering the field of application, this is a rather academic aspect. Although the user should be aware of this fact. Another disadvantage is that the model produces discrete output variables from a continuous process. To overcome these handicaps a new model approach of the coated wheel's movement through the airflow was developed, implemented and tested. This approach was developed during the work on [4].

2 MODELICA Model of *Casas et al.*

The model of *Casas et al.* is described in detail in [2]. This section will only give a short overview of the model and highlight the structural criteria that were changed in this work. As shown in figure 2 the implementation in MODELICA is based on control volumes for air (AirCV) and for the desiccant material (wall_A/B), which can exchange heat and moisture.

MODELICA can only handle ordinary differential equations with respect to time. Therefore the basic idea of the first approach was a variable transformation to express the position of the rotating wheel with respect to the airflows in terms of time instead of angular position. Among the assumptions made in [2], three are elementary in this approach:

1. The states are not a function of the wheel's radius: $\vartheta, x \neq f_{(R)}$.
2. The variation of boundary conditions during half a rotation is negligible: $\left.\frac{\partial BC}{\partial t}\right|_{\frac{T}{2}} \approx 0$.
3. The angular velocity of the wheel ω is constant during half a period: $\left.\frac{\partial \omega}{\partial t}\right|_{\frac{T}{2}} \approx 0$.

Equation (1a) gives the average outlet temperature of one airstream. To calculate the integral, the tangential outlet temperature distribution must be known. This leads to a tangential discretization of the wheel and a modeling of the the motion of the discrete pieces through one airflow into the other. To display this movement the variable transformation from the angle

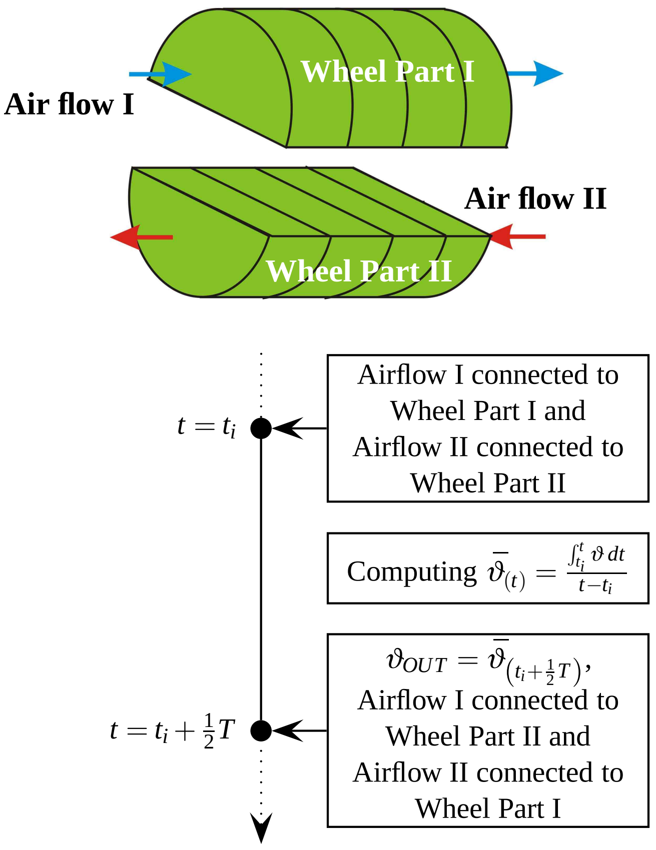

Air flow I
Wheel Part I
Air flow II
Wheel Part II
$t = t_i$
$t = t_i + \frac{1}{2}T$
Airflow I connected to Wheel Part I and Airflow II connected to Wheel Part II
Computing $\bar{\vartheta}_{(t)} = \frac{\int_{t_i}^{t} \vartheta\, dt}{t - t_i}$
$\vartheta_{OUT} = \bar{\vartheta}_{(t_i + \frac{1}{2}T)}$, Airflow I connected to Wheel Part II and Airflow II connected to Wheel Part I

Figure 3: Function principle of the model from *Casas et al.*

ϕ to the time t is introduced, which leads to equation (1b).

$$\bar{\vartheta}_{\text{Air,out}} = \frac{1}{\pi} \int_0^{\pi} \vartheta_{\left(\phi, t, BC_{(t)}\right)}\, d\phi \tag{1a}$$

with

$$t = \frac{\phi}{\omega} = \frac{\phi \cdot T}{2\pi} \text{ and } dt = d\phi$$

leads to

$$\bar{\vartheta}_{\text{Air,out}} = \frac{2}{T} \int_{t_0}^{t_0 + \frac{T}{2}} \vartheta_{\left(t, BC_{(t)}\right)}\, dt \tag{1b}$$

The advantage of this formulation is, that the wheel has only to be split in two halves; one for each air flow. At the end of half a rotation period, the boundary conditions of the two pieces are switched; the wheel has performed half a revolution. Figure 3 illustrate this behavior.

Another more process engineering oriented point of view is that the continuous process of the turning wheel is represented by batch processes, which each last half a revolution period.

Among the application restrictions mentioned, this model has two other drawbacks. First it produces discrete output variables from continuous input values in contrast to the physical process. And second, as will be pointed out in section 4.2 the models computing time is quite large, because it causes every $\frac{T}{2}$ a state event.

3 Structure of the new MODELICA Model

Based on the restriction of applicability and the large computing times mentioned in section 2, a new modeling structure has been developed. The basic idea in this approach is not to perform the variable transformation, but to use equation (1a). To reach this aim, another way to describe the motion of the wheel through the air flows had to be introduced.

The construct of the air and desiccant material control volumes is no longer virtually moved through the air flow by switching the air connectors every half rotation period. Instead the control volume are locally fixed and a *desiccant fluid* was introduced, which flows through the *desiccant CV*'s in cross flow to air flow direction.

Therefore the existing control volumes for air and the desiccant material were used to build a wheel model with a discretization in axial and tangential direction. This modeling approach is sketched in figure 4(a). The black lines on the wheel should hint to the discretization. In contrast to figure 3 the air connectors of the *Air CV* are attached to the in- and outlet connectors of the wheel model. Also the desiccant material models are connected in series to model the rotation by keeping the *desiccant fluid* in a continuously circulating flow.

Figure 4(b) illustrates how the *air* and the *desiccant CV*'s interact by exchanging heat and moisture for modeling the (de-)humidifying the air by the sorbens. These flows are sketched by the double headed arrows between the two CV's. Air flows along the cylinder axis while the solid passes its CV's in tangential direction. These two streams are indicated in figure 4(b) by the arrows near the two CV's.

For these purposes a *desiccant fluid* flow had to be introduced in the *desiccant CV* and set in relation to the revolution speed. This mass flow is computed by equation (2d), which results of the wheel's mean circumferential velocity (2a), the conservation of mass (2b) and the radial passage area (2c). The area is obtained by dividing the longitudinal half section $R \cdot L$ by the axial discretization n.

$$v_{DF} = \frac{R}{2} \cdot \omega \tag{2a}$$

with

$$\dot{m} = A \cdot \rho \cdot v \tag{2b}$$

and

$$A = \frac{L \cdot R}{n} \tag{2c}$$

leads to

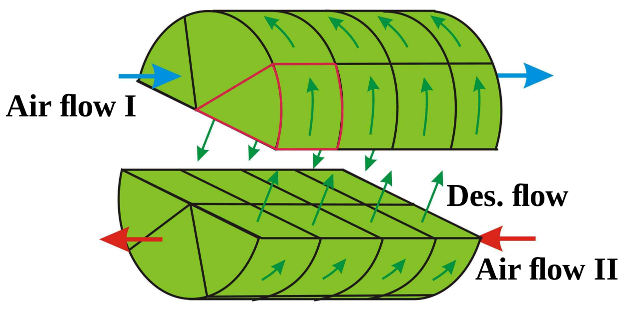
$$\dot{m}_{DF,i} = \frac{L \cdot R^2 \cdot \omega \cdot \rho_{DF}}{2 \cdot n} \tag{2d}$$

The density ρ_{DF} represents the mass of the carrier material for the sorbent divided by the volume of the wheel, thus including its porosity. This definition was chosen, because the phase equilibrium calculation uses the loading of the carrier material with the sorbens and the loading of the sorbens with water.

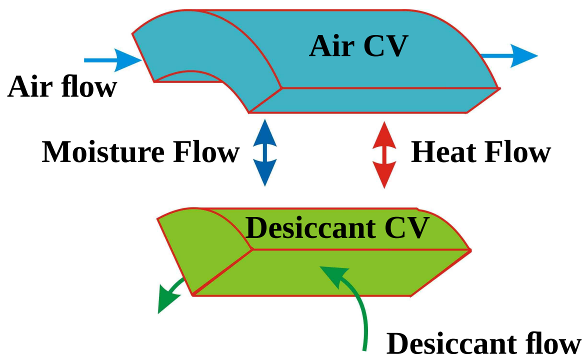
As mentioned before, the new wheel model was constructed by control volumes for air and desiccant material. *Casas et al.* used for their model *Air CV*'s with an axial discretization n in flow direction. To include as much of the existing code as possible m of the *air CV*'s are put side by side to get a control volume, which is discretized in two dimensions. Each stream tube in this construct can not directly interact with its neighbors. Listing 1 gives some code snippets to illustrate the implementation in MODELICA.

There are m instances of the model `AirCV`, which are n times discretized *Air CV*. Each `AirCV` has an `HeatConnector` and `HumidityConnector` in order to couple it with the *Desiccant CV*. These connectors are united in the two models `Heat1Dto2D` and `Humidity1Dto2D`. The function of these two models is to provide a $n \times m$ matrix of heat and humidity flows respectively, so that an Air2D model can easily be connected to a two dimensional desiccant material model to form half a desiccant wheel. Figure 5 shows the two *Air CV*'s connected to two *Desiccant CV*'s, which form a closed loop with a circular flow of the *desiccant fluid*.

The *Desiccant CV* is constructed using an analog method. Its model name in the library is `SMCV_2D`.



(a) Complete Wheel



(b) Detail scheme of one element from the wheel in fig. 4(a)

Figure 4: Scheme of the new modeling approach

It has *n* stream tubes, which are discretized *m* times. The mass flow of the virtual fluid is computed in both SMCV_2D models by equation (2d). The connectors between the *Desiccant CV* only contain the temperature and the water loading of the *desiccant fluid*. The reason for this is to avoid initialization problems with a circular incompressible flow. The disadvantage of this approach is, that the wheel's rotation direction is fixed. But this restriction is also true for the real desiccant wheel, which was used for the experimental part. Another simplification of this model is that the entrainment of air from one air flow to the other is not modeled. So the simulation of fast rotating wheels will lead to errors. But in their application the wheel's circumferential speed is small in comparison to the velocity of the air flows.

Listing 1: Excerpt from the Air CV 2D Matrix model

```
model Air2D
  parameter Integer n = 1 "Axial
     Discretization";
  parameter Integer m = 1 "Tangential
     Discretization";
    ⋮
  AirCV[m] Air(
    each n=n,
     ⋮
  );
  MeanValues Mean(m=m);
  Heat1Dto2D HeatMatrix(n=n,m=m);
  Humidity1Dto2D HumidityMatrix(n=n,m=m
     );
  AirSplit Split(m=m);
equation
  for j in 1:m loop
    connect(Split.Outlet[j], Air[j]
       .Inlet);
    connect(Air[j].Outlet, Mittel.Inlet
       [j]);
    connect(Air[j].HeatConnector,
       HeatMatrix.Heat1D[j]);
    connect(Air[j].HumidityConnector,
       HumidityMatrix.Hum1D[j]);
  end for;
   ⋮
end Air2D;
```

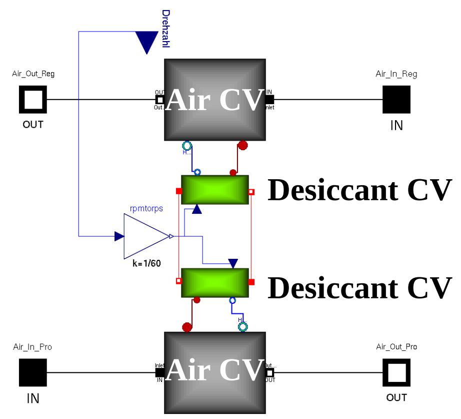


Figure 5: Dymola representation of the new modeling approach

4 Comparison of the two Models

4.1 Results

The model of *Casas et al.* contains sorption isotherms in the medium model for LiCl, which where validated with measured values. The steady state results of the whole wheel were checked against the manufacturer's data and the transient simulations against the model of *Rau et al.* [5]. These isotherms were also implemented in the *desiccant material CV* of the new model. To validate the implementation the test model shown in figure 6 was used. This model consists of a *desiccant material CV* which is connected to an *Air CV*. The des-

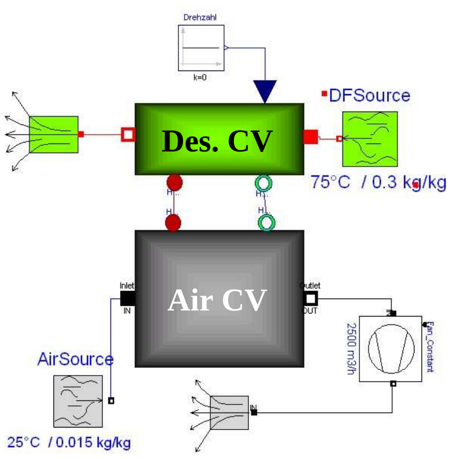


Figure 6: Test model for the comparison with the isotherm data of *Rau et al.*

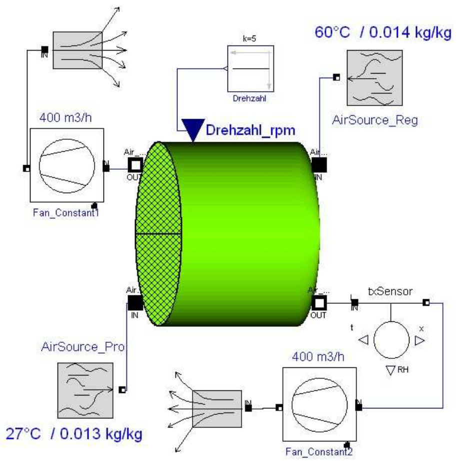


Figure 8: Test model for comparisons of the whole wheel model, here with the model form this work

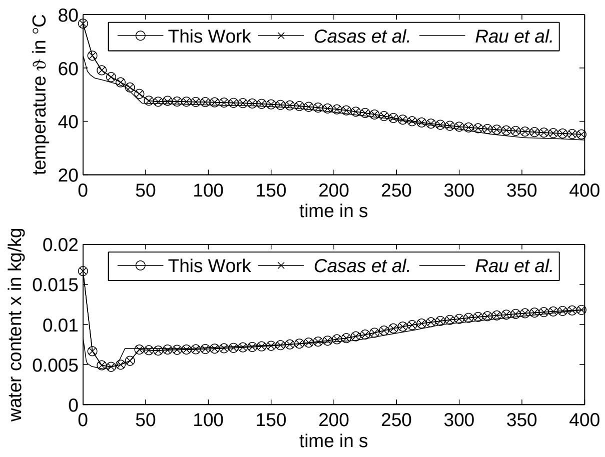


Figure 7: Comparison of a single element of the model from *Casas et al.* and the new one

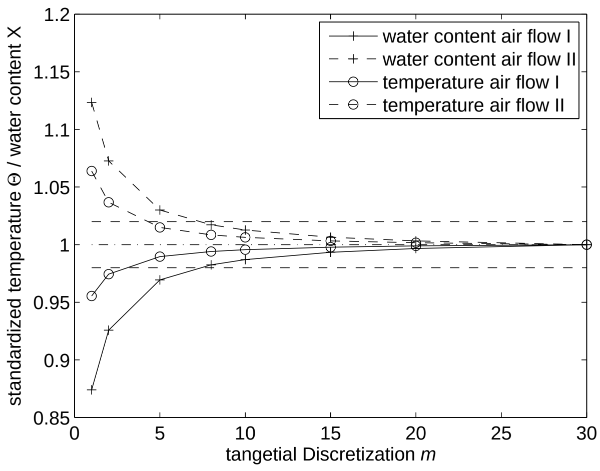


Figure 9: Influence of the tangential discretization to the accuracy

iccant fluid flow is set to zero in this model, so that the results can be compared to the single blow simulations done by *Rau et al.* [5] and *Casas et al.* [1, 2]. Figure 7 shows the results of the three models. It is obvious that the sorption isotherms implementation in the new model is equivalent to the model of *Casas et al.*. In this aspect it is adequate, because the mathematical implementation of those isotherms was already adapted to the use in MODELICA.

For all results, which are discussed below, models were used, which contain an instance of a whole desiccant wheel model and sources and sinks for the air flows. One of those test configurations is presented in figure 8, in this case with the new model.

The accuracy of the new model depends on the tangential discretization m. The behavior of the *desiccant material CV* approaches that of the real wheel as the computational grid is refined, viz. the more discrete control volumes are instantiated in flow direction of the *desiccant fluid*. Figure 9 shows standardized outlet temperatures Θ and water contents X of the two air flow through the desiccant wheel model. They are plotted against the tangential discretization m. The reference value is the corresponding simulation result from a calculation with $m = 50$. The values were computed from the steady state results from a step response after 5000 s. It can be seen, that at $m = 8$ the relative error is smaller $\pm 2\%$. Later the consequences on the CPU time will be discussed, but it can be seem from table 1, that depending on the required accuracy m should be chosen as 5 or 8.

Figure 10 shows a detail view on the step response of an outlet temperature of three test models, one with the desiccant wheel form *Casas et al.* and two with wheel models from this work with different numbers of tangential CV's. The complete simulation time was

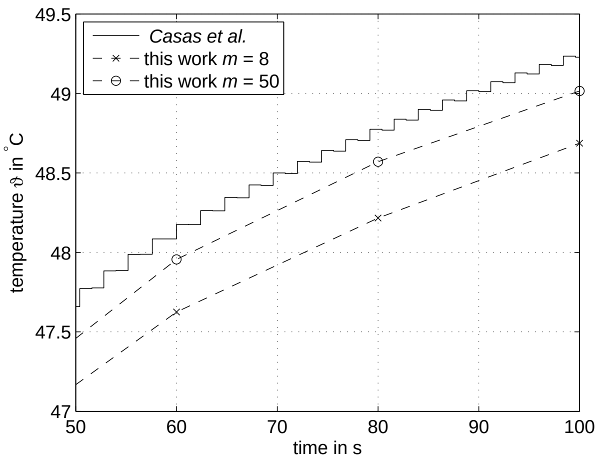


Figure 10: Comparison of complete the model from *Casas et al.* and the new one

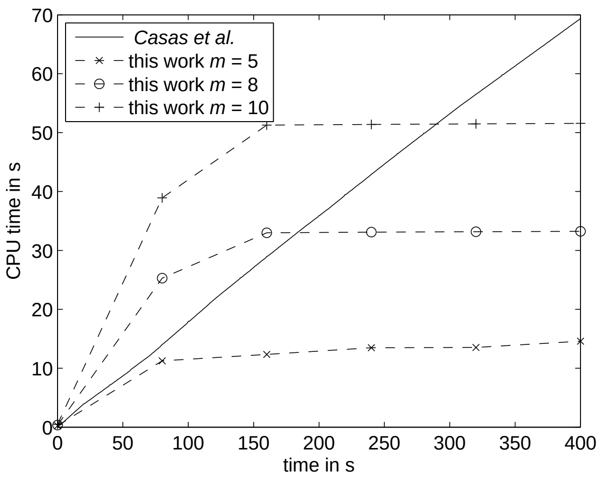


Figure 11: Impact of tangential discretization m on computing time, compared with the model from [2]

5000 s. The new model produces continuous output variables in respect to the solver step-size in contrast to the discrete output of the old one. The model with the tangential discretization of $m = 8$ has a slightly larger deviation from the one with $m = 50$ than the model of *Casas et al.*. But as will be shown in the next section, it has a remarkable advantage concerning computing times. And compared with measurement errors, the accuracy is sufficient.

4.2 Computing Time

The computing times discussed in this section correspond to calculations on one core of an Genuine Intel(R) CPU T2300 @ 1.66 GHz on a laptop with 1 GB RAM.

Figure 11 shows a comparison in computing time between the previous approach and the new model. The old model has no tangential discretization, but has to modify the `connect` statement between the air flows an the wheel's control volumes every half period. Whereas the new one needs to be divided in at least five to eight parts (m in fig. 11) per control volume to produce good output values. This leads to the behavior presented in the plot. The old model produces at every half revolution time an event while switching the sides, which wastes computing time while reinitialization of the equation system. This leads to the nearly linear characteristic consisting of numerous small steps. The new model contains a multiple ($\sim m$ times) of equations compared the the old one, so the computing time for one step is much higher, but due to the model structure time steps larger than half the revolution time are possible. In highly dynamic regions, like the beginning of the plot in figure 11, the computational effort is high, but due to the large time steps in regions with small gradients, the overall computing time is lower for the new model with a tangential discretization of m = 5,8 and 10 for simulation times larger than 300 seconds in this example.

This behavior is expected to lead to a large decrease in computing time, especially at long time simulations of air-conditioning systems.

Table 1 gives an overview of the equations, which are created by the different test models, so the number of equations is a little larger than in the stand alone wheel model. The table also shows the computational time of the test model for a simulation time of 5000 s. Due to the above mentioned effect of the state events during the simulations of *Casas et al.*'s model, the new model with $m = 5$ is nearly 20 times faster even though it consists of about 10 times the number of equations. In this case with the old wheel's test model 4166 state events occurred during simulation time.

5 Summary and Outlook

Because of the enhancement of the desiccant wheel model from *Casas et al.* a MODELICA model could be created, which combines good accuracy with acceptable computing times. It was successfully used in further work of *Applied Thermodynamics*, like [4, 6], as the heart of a library for desiccant systems. Several simulation of complete climate periods were performed as well as studies concerning different control strategies of those systems. For the analysis of control strategies the models were exported to Matlab/Simulink to find and optimize control parameters.

Table 1: Number of equations and CPU time of the test model against the tangential discretization m

m	No. of equations	CPU time[a] in s
1	1681	~ 1
2	3043	~ 4
5	7129	20
8	11215	48
10	13939	78
15	20749	186
20	27559	289
30	41179	810
50	68419	2154
Casas et al.	1184	827

[a]For a simulation time of 5000 s in Dymola using Dassl

Especially for this part it was very helpful, that the new model no longer produces discrete output. It was also possible to adapt the model parameters in such a way that data from existing air conditioning systems could be recomputed.

Nomenclature

Latin Symbols	
A	Area
L	Length
m	Tangential discretization
$\dot{m}$	Mass flow
n	Axial discretization
R	Radius
t	Time
T	Period
v	Velocity
x	Water content
X	Standardized water content
Greek Symbols	
ϑ	Temperature
Θ	Standardized temperature
ρ	Density
ϕ	Angle
ω	Angular Velocity
Abbreviations and Subscripts	
BC	Boundary condition
CV	Control Volume
DF	Index for *desiccant fluid*
i	Index for i-th element

References

[1] Casas, Wilson: *Untersuchung und Optimierung sorptionsgestützter Klimatisierungsprozesse*. PhD thesis, TU Hamburg-Harburg, 2005.

[2] Casas, Wilson, Katrin Proelss, and Gerhard Schmitz: *Modeling of desiccant assisted air conditioning systems*. In *Proceedings of the 4th International Modelica Conference*, volume 2, pages 487–496. Modelica Association, 2005.

[3] Casas, Wilson and Gerhard Schmitz: *Numerische Untersuchungen an einer sorptionsgestützen Klimaanlage*. In *VDI Fortschrittliche Energiewandlung- und Anwendung*, volume 1594 of *VDI-Berichte*, 2001.

[4] Joos, Andreas: *Untersuchung und Optimierung eines solargestützten Heiz- und Klimatisierungssystems für ein Einfamilienhaus*. Master's thesis, TU Hamburg-Harburg, Institut für Thermofluiddynamik, 2006.

[5] Rau, J. J., S. A. Klein, and J. W. Mitchell: *Characteristics of lithium chloride in rotary heat and mass exchangers*. Int. Journal of Heat and Mass Transfer, 34(11):2703–2713, 1991.

[6] Schmitz, Gerhard, Wilson Casas, and Andreas Joos: *Entwicklung eines thermisch betriebenen Klimatisierungssystems für Ein- und Zweifamilienhäuser*, December 2006. Institute of Thermo-Fluid Dynamics, Hamburg University of Technology.

[7] Simonson, C.J. and Robert W. Besant: *Heat and Moisture Transfer in Desiccant Coated Rotary Energy Exchangers: Part I. Numerical Model*. HVAC&R Research, 3(4):325–340, 1997.

[8] Zheng, W. and W.M. Worek: *Numerical simulation of combined heat and mass transfer processes in a rotary dehumifier*. Numerical Heat Transfer, 23:211–232, 1993.

Real-Time HWIL Simulation of Liquid Food Process Lines

Magnus Gäfvert[a] Tomas Skoglund[b]
Hubertus Tummescheit[a] Johan Windahl[a]
Hans Wikander[c] Philip Reuterswärd[a]

[a]Modelon AB
Ideon Science Park, SE-223 70 Lund, Sweden
magnus.gafvert@modelon.se

[b]Tetra Pak Processing Systems
Ruben Rausings gata, SE-221 86 Lund, Sweden
tomas.skoglund@tetrapak.com

[c]Avensia Innovation AB
Gasverksgatan 1, SE-222 29 Lund, Sweden

Abstract

This paper describes a newly developed Modelica and Dymola based solution for hardware-in-the-loop (HWIL) simulation in the food processing industry. The solution has been evaluated for potential larger scale deployment into the operational processes of Tetra Pak Processing Systems. The solution consists of a real-time enabled model library for liquid food processing, which is compiled into a process simulator using Dymola, and custom developed software for communication between the process simulator and a production PLC control system using industry standard OPC protocols.

Keywords: physical modeling and simulation; hardware-in-the-loop; liquid food processing; process simulation; real-time simulation

1 Introduction

Dynamic simulation of liquid food process lines, e.g. pasteurization lines in dairies, see Figure 1, has already been practiced in a systematic way by means of the FoodProcessing library (FP), see Figure 2. This Modelica [1] and Dymola [2] based dynamic model library developed for in-house use has previously been reported in [3] (Skoglund, 2003), [4] (Skoglund and Dejmek, 2006) and [5] (Skoglund, 2007). Besides the fundamental laws of conservation, e.g. mass and energy, the model library addressed particular characteristics of liquid food process lines. For example dynamic propagation of fluid properties was considered due to the need of simulating start-up and shut-down with fluid changes, which are occurring frequently in the addressed applications.

Within the operations of Tetra Pak Processing Systems, the FoodProcessing library was used to simulate many processes with their control system as a tool for development or improvement. Simulation was also used as a means for trouble shooting.

Figure 1. A typical dairy process-line for pasteurization.

In the regular delivery process of Tetra Pak's order handling, food processing units are functionally tested by running them with water before they are shipped to the customers. This is carried out to secure high quality of the equipment. The test cannot be carried out before the machine is manufactured, which leads to the need of extra time before delivery. The test itself also requires costly test places with water, steam, electricity, compressed air and drain available. Also, water has different properties compared to the liquid food that will eventually be processed by the unit which means that the test result may deviate from real plant performance.

To enable shorter delivery time at a lower cost, alternatives to this functional test were investigated. One of the alternatives is to run real-time hardware-in-the-loop (HWIL) simulation where the real PLC (Programmable Logic Controller) control system is connected and run with the process model. Since the process model enables simulation with not just water, but real fluid models the HWIL simulation may also, in some cases, be more realistic. Furthermore, often the normal water test does not include special equipment (centrifugal separators or equipment upstream/downstream) due to practical problems. For simulation, this limitation does seldom exist. In simulation it is also possible to monitor virtually any dynamic variable in the system without the need for sensors, which may be of great help to quickly understand and resolve issues.

Furthermore HWIL simulation enables other possibilities, e.g. as a test, validation, and verification tool in PLC software development, and operator training [6] (De Prada et al., 2003) and [7] (Bäckman and Edwall, 2005).

This article describes:

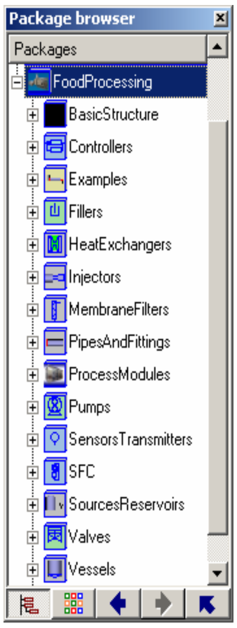
- How the model library was adapted for real-time simulation
- How a communication program was developed as a link between the PLC and the simulator.

The work was carried out as a project with Tetra Pak Processing Systems and Modelon.

2 The "FoodProcessing" library

Since the start of the development of the "FoodProcessing" (FP) library [3] (Skoglund, 2003) much more work was spent to address characteristics of liquid food process lines. Thus Skoglund et al. (2006) [8] described a way to handle fluid transitions in heat exchangers that leads to thermal transients. A model for axial-dispersed plug flow (ADPF) was also described [9] (Skoglund and Dejmek, 2007) and extended to model first-order reaction kinetics [10] (Skoglund and Dejmek, 2007). Figure 2 shows the FoodProcessing library in the Modelica tool Dymola.

The library has since been used to configure many process lines and to investigate various performance issues, e.g. product losses. Thus the development of a mixing zone was simulated for product filling and emptying in a commercial UHT line for milk sterilization [11] (Skoglund & Dejmek, 2007). The result was compared with measured data.



FoodProcessing
- BasicStructure
- Controllers
- Examples
- Fillers
- HeatExchangers
- Injectors
- MembraneFilters
- PipesAndFittings
- ProcessModules
- Pumps
- SensorsTransmitters
- SFC
- SourcesReservoirs
- Valves
- Vessels

Figure 2. The FoodProcessing library

The library was also used for trouble shooting and testing new design ideas, both concerning process design and control algorithm.

3 Real-Time Aspects

The original FoodProcessing library was designed for high-fidelity desktop simulation with variable-step high-order solvers and many models were not suited for real-time simulation. It was decided to translate and adapt the library into a real-time Food-Processing library (FPRT). The models in FPRT are made to simulate robustly with fixed-step solvers with a computational load that avoids computation over-runs when executed in real time on standard PC:s.

One major difficulty is the nonlinear equation systems that appear from the pressure dynamics for pipe networks with incompressible fluids. A related difficulty is the effective structural change that valve closing and opening implies on the equation systems. The overall system contains stiff modes and requires implicit methods for numerically stable integration. The inline integration feature of Dymola is used to take advantage of the symbolic reduction and transformation. To reduce the sizes of the resulting equation systems an explicit integration routine was introduced by inlining Modelica code in some components in the library. (The mixed implicit/explicit inline integration in Dymola does not handle the present type of models.)

Several numerical tweaks were introduced to increase robustness [12]. Several models have also been simplified and discretization grids have been made smaller. The number of dynamic states has been reduced.

4 HWIL Setup

HWIL simulation is often performed on dedicated computers with real-time operating systems and extensive I/O possibilities. In the present application the solution should be able to run on a standard PC with Microsoft Windows operating system with Ethernet or automation-bus communication with the PLC hardware. This means that hard real-time cannot be ensured. The sampling rate of feedback control-loops in the food processing applications are in the range of 100 ms or longer, and sufficient performance can be met with soft real-time.

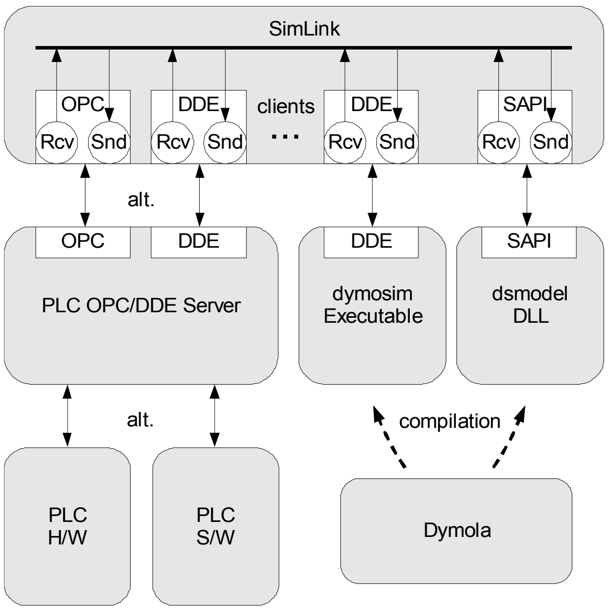


Figure 3. Overview of HWIL setup with SimLink signal routing application, PLC communication server, PLC hardware or emulated controller, and dymosim executable or dsmodel DLL process simulator.

The HWIL setup consists of a PLC system with control algorithms and programs, the process simulator, and a software to route signals between the PLC and the simulator. See Figure 3.

4.1 PLC System

PLC systems are digital computers for automation with extensive support for I/O arrangements and bus communication. The PLC computers host control programs for sequence control and sampled data feedback control typically expressed with IEC 61131-3 languages such as ladder diagrams or function block diagrams.

Tetra Pak work with several suppliers of PLC systems, such as Rockwell Automation, see Figure 4, and Siemens, and the HWIL solution must support them all. Most major systems support the DDE (Dynamic Data Exchange) and OPC (OLE for process control) technologies for interoperability [13,14].

Figure 4. Allen Bradley PLC Controller from Rockwell Automation.

DDE is an old technology for communication between multiple applications under Microsoft Windows. OPC is a standard protocol for open connectivity in industrial automation. DDE suffers from scalability and performance issues, and is more or less being superseded by newer technology. Therefore, the communication between the simulator and the PLC was decided to build mainly on OPC with DDE as fallback.

OPC is originally designed for communication with HMI (Human-Machine Interface) units, operator panels, and enterprise systems with moderate to low requirements on data bandwidth. The standard has then evolved to cover a wider class of communication tasks in industrial automation. OPC supports synchronous and asynchronous communication and is highly flexible and scalable. OPC is not primarily intended for feedback control or communication with high-bandwidth hard real-time requirements. With a soft real-time performance of about 400 items at a rate of about 20-30 ms, or 2000 items at about 100 ms it is still deemed sufficient for the present HWIL application.

In HWIL simulation the I/O signals in the PLC are re-routed from the physical I/O card to memory addresses associated with OPC items. This means that the PLC programs must be extended with a simulation mode to support simulated I/O.

Some vendors, such as Rockwell Automation, also offer emulators for their PLC computers. This makes it possible to also work with SWIL (software-in-the-loop) with the same setup.

4.2 Process Simulator

Dymola supports, via the dymosim executable, stand-alone real-time simulation with DDE communication. All variables in the model are then available as DDE items for subscription. The performance and scalability issues with DDE mean that alternative solutions have also been investigated.

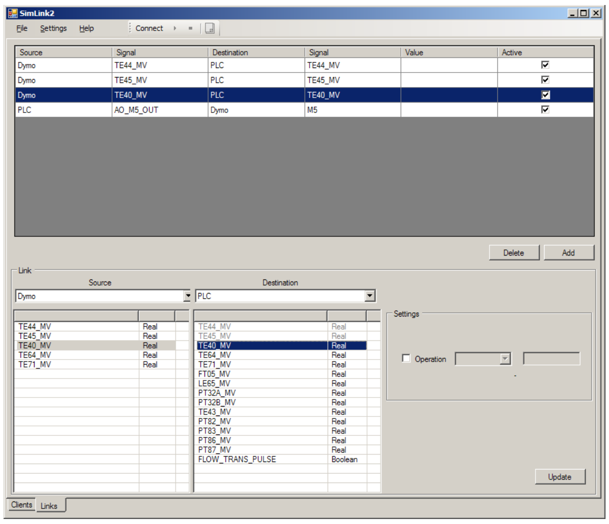
One attractive solution is to use a model DLL (Dynamically Linked Library) similar to that used in the DymolaBlock that enables Dymola models to be used in MATLAB/Simulink. This means that an external integration routine is used and the model derivatives and outputs are returned by direct function calls. An SAPI (Simulation Application Programming Interface) for calling the simulation model as a function was therefore developed together with build scripts to produce the model DLL. For real-time simulation the integration routine is a fixed-step explicit Euler with event management. The direct function calls means that virtually all communication overhead is eliminated.

4.3 Signal Routing with SimLink

The number of signals in a HWIL setup may be in the range of a few ten for small process modules, to several hundred for large processes. The signals represent all sensor and actuator values that logically connects the process with the PLC, but may also include values from "virtual sensors" that are not available on the real process. The signals may also represent alarms and warnings from the model components, for example to alert the user of operating points outside the range of validity.

A core component in the HWIL setup is the organization and synchronization of the signal routing. The SimLink software described in the following was developed for this purpose.

SimLink can be viewed as a coupling panel where input and output signals from different clients are connected or linked via a graphical user interface. SimLink is a Windows application and builds on the Microsoft .NET Framework and is based on OPC Core Components and OPC .NET API 2.0. SimLink is configured by specifying a set of clients, defining their signals, and then connect the signals by introducing links, see Figures 5 and 6. The configuration of clients, signals and links can be saved to a configuration file, that later can be loaded into the application. This makes it easy to maintain different configuration setups. The configuration file contains all information of the setup, and is stored in a human readable XML format.

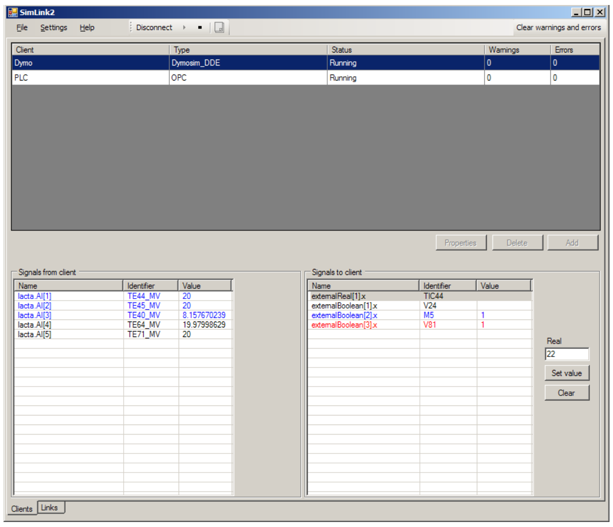


Source	Signal	Destination	Signal	Value	Active
Dymo	TE44_MV	PLC	TE44_MV		☑
Dymo	TE45_MV	PLC	TE45_MV		☑
Dymo	TE40_MV	PLC	TE40_MV		☑
PLC	AO_M5_OUT	Dymo	M5		☑

TE44_MV	Real
TE45_MV	Real
TE40_MV	Real
TE64_MV	Real
TE71_MV	Real

TE44_MV	Real
TE45_MV	Real
TE40_MV	Real
TE64_MV	Real
TE71_MV	Real
FT05_MV	Real
LE65_MV	Real
PT32A_MV	Real
PT32B_MV	Real
TE43_MV	Real
PT82_MV	Real
PT83_MV	Real
PT86_MV	Real
PT87_MV	Real
FLOW_TRANS_PULSE	Boolean

Figure 5. The Links view displays details on signal links and offers a convenient user interface for connecting clients.



Client	Type	Status	Warnings	Errors
Dymo	Dymosim_DDE	Running	0	0
PLC	OPC	Running	0	0

Signals from client

Name	Identifier	Value
lacta.AI[1]	TE44_MV	20
lacta.AI[2]	TE45_MV	20
lacta.AI[3]	TE40_MV	8.157670239
lacta.AI[4]	TE64_MV	19.97998629
lacta.AI[5]	TE71_MV	20

Signals to client

Name	Identifier	Value
externalReal[1].x	TIC44	
externalBoolean[1].x	V24	
externalBoolean[2].x	M5	1
externalBoolean[3].x	V81	1

Real: 22

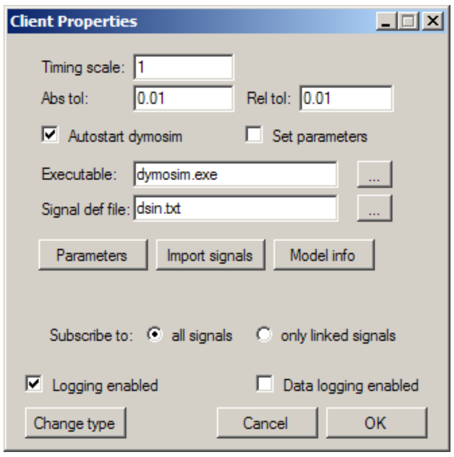
Figure 6. The Clients view gives an overview of configured clients and signals. Signals can be monitored and manipulated in run mode.

When the setup is finished, the next step is to connect to the clients. The application then makes sure that it has valid connections to all clients and that they are ready to send and receive signals. After connecting, the application is ready to go into run mode, and this is done by clicking the *play*-button. In run mode, SimLink is listening to all signals sent to the clients from external applications and internally routes them through the links to clients connected to the receiving applications. Run mode is ended by clicking the *stop*-button.

SimLink currently supports the client types listed in Table 1. Figures 7 – 9 shows the properties dialogs for the DDE, OPC, and SAPI client types.

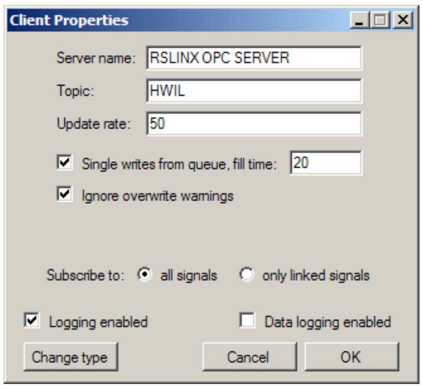
Table 1. SimLink client types.

Client type	***Description***
DDE	Connects to programs that support Windows DDE.
OPC	Connects to programs with an OPC server
SAPI DLL	The simulation model resides in a DLL that is loaded into the client. The client contains the simulation algorithms and communicates with the model through a direct function API (SAPI).
Internal	Signal sink for testing purposes.
Trigger	Signal source that generates output signals at a specific rate.


Client Properties

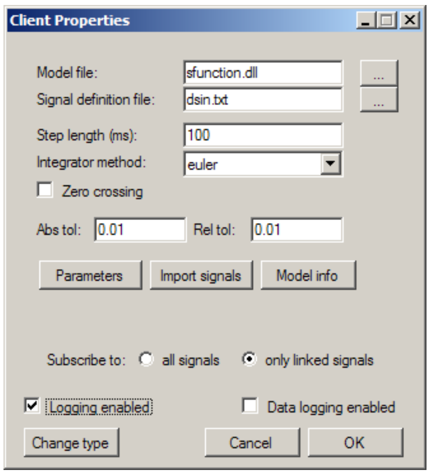
Timing scale: 1
Abs tol: 0.01 Rel tol: 0.01
[✓ Autostart dymosim] Set parameters
Executable: dymosim.exe
Signal def file: dsin.txt
Subscribe to: [✓ all signals] only linked signals
[✓ Logging enabled] Data logging enabled

Figure 7. Properties for DDE client.


Client Properties

Server name: RSLINX OPC SERVER
Topic: HWIL
Update rate: 50
[✓ Single writes from queue, fill time:] 20
[✓ Ignore overwrite warnings]
Subscribe to: [✓ all signals] only linked signals
[✓ Logging enabled] Data logging enabled

Figure 8. Properties for OPC client.


Client Properties

Model file: sfunction.dll
Signal definition file: dsin.txt
Step length (ms): 100
Integrator method: euler
Zero crossing
Abs tol: 0.01 Rel tol: 0.01
Subscribe to: all signals [✓ only linked signals]
[✓ Logging enabled] Data logging enabled

Figure 9. Properties for SAPI client.

The SimLink OPC client is to date verified to support PLC systems from Rockwell Automation and Siemens.

5 Process Examples

The process line that was chosen for the evaluation of the real-time HWIL simulation was a custom designed commercial processing module for dairy pasteurization[1], see also Figure 1. Figure 10 shows the top-level model diagram (flow chart) as configured by using the library FPRT.

The process consists of a balance tank, a plate heat exchanger for pre-heating and pasteurization, a deaerator, a homogenizer, holding cell, steam-powered hot-water unit, and pumps, valves, and sensors being monitored and controlled by the PLC system. The process supports a number of operating modes, e.g., start-up, production, cleaning, and hibernation, which have different flow configurations.

[1] Tetra Therm Lacta, designed and manufactured by Tetra Pak Processing Systems.

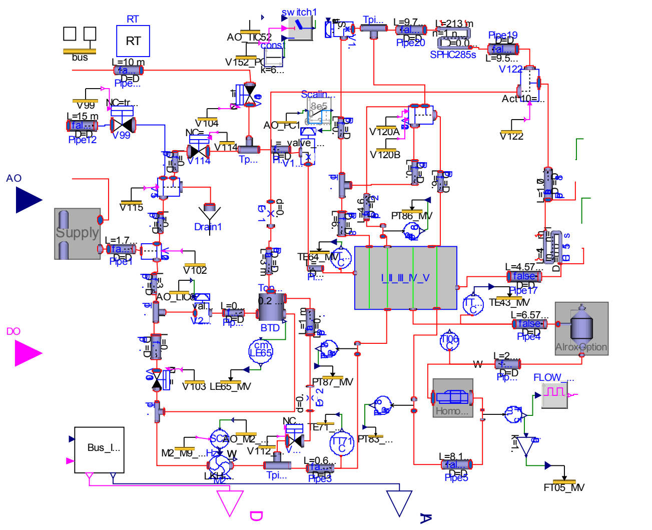


Figure 10. Flow diagram of the process line used to evaluate real-time HWIL simulation. The model is built in a hierarchy. The figure shows the top level view.

Figures 11 and 12 show the PLC operator panel that is used to control and monitor the process.

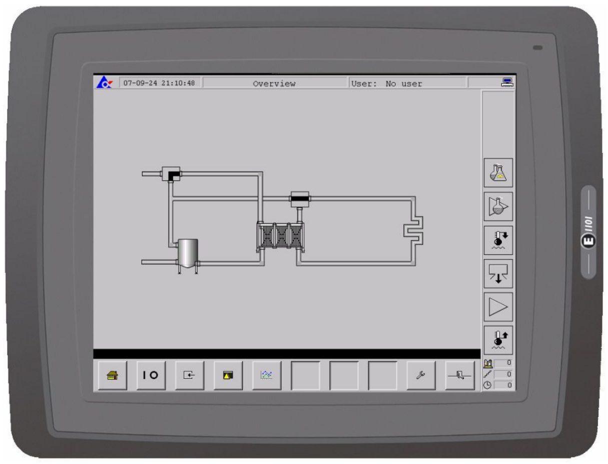


Figure 11. An overview picture of the PLC operator panel.

6 Results

The described HWIL solution is being evaluated both from a business perspective and a technical perspective. Technically, the presented solution seems to fulfil all given requirements. There have been a number of minor issues that have been resolved, but the core solution design and architecture has shown to be sound, scalable, and extensible.

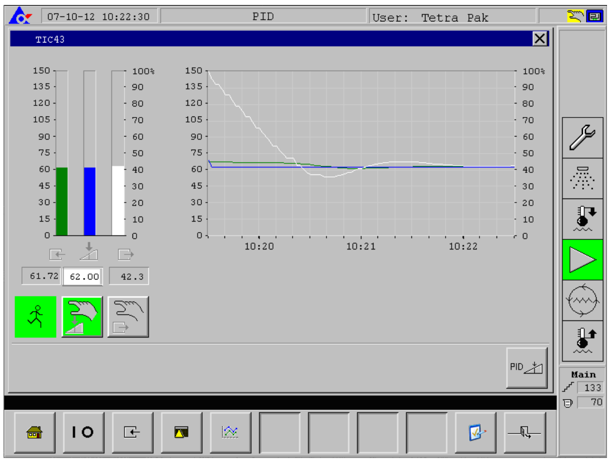


Figure 12. PLC operator panel showing the performance of a PID controller regulating a simulated process.

A HWIL simulation setup was prepared for the process module described in Section 5 with its production PLC system from Rockwell Automation (hardware and software). A number of evaluations were arranged where the HWIL testing procedure was compared with the traditional water testing described in Section 1. The tests were built on cases from water testing of actual units delivered to customers and now in production. The models and PLC programs were rigged with a set of faults from a database and testing personnel then performed an HWIL testing to see if these faults would be identified. The protocols from this HWIL testing reported all or most problems also found in water testing, and then some additional that was not found in water testing. The results so far indicate that HWIL simulation may indeed replace water testing and also result in better test coverage.

One comment from the de-briefing of the test personnel was that the simulated process was lacking the noises, sounds, vibrations, and other sensory information that is used by humans to monitor the process and detect deviations. Still, the overall impression was that the simulator had advantages over water testing. The SimLink software was extended with alarm/alert functionality to support emulated sensory information from the simulation model. A component can, for example, trigger a boolean signal to indicate that the fluid media is boiling. On the real process this might have been detected by noise. In SimLink this triggers an alarm that alerts the user of the abnormal situation.

The HWIL setup was also in parallel used for testing, validation, and verification in development of new PLC control software. The HWIL setup has proven to be very useful to find and identify issues and bugs at an early stage.

There have been some problems with robustness of the simulation at complex mode switches. Possibly, this will lead to some re-design of the fundamental models of the fluid dynamics. For example, the current models are designed for uni-directional flow, even though back-flow may occur in transients and mode switches.

7 Conclusions

Experiences from the presented solution indicate that the Modelica based HWIL technology may contribute significantly and in a wide range of operations in a business organization like Tetra Pak Processing Systems. Parts of a larger evaluation effort have been performed and indicate that expensive and time-consuming water testing may be replaced by simulation. Application of the HWIL solution for software debugging has also been done successfully.

Large parts of the FP library have been adapted for real-time simulation with the evaluated process module. Remaining parts will be adapted as HWIL simulation is introduced for other process modules.

The SimLink program was designed and developed for general HWIL simulation with PLC systems in the process industry. It has been continuously improved during the work described in this paper and has now become a stable and feature complete core component in the HWIL simulation environment.

References

[1] Modelica Association, http://www.modelica.org

[2] Dynasim AB, http://www.dynasim.se

[3] Skoglund, T. Simulation of Liquid Food Processes in Modelica. Proceedings of the 3rd International Modelica Conference 2003, 51-58. Linköping, Sweden, November 3-4, 2003, Organized by the Modelica Association and Linköping University, Sweden. Available at http://www.modelica.org.

[4] Skoglund, T. and Dejmek P. A model library for dynamic simulation of liquid food process lines. Proceedings of FOODSIM 2006, 5-12, Naples, Italy, June 15-17, 2006, Organized by EUROSIS.

[5] Skoglund, T. Dynamic Modelling and Simulation of Liquid Food Process Lines. Lund, Sweden: Doctoral thesis, Department of Food Technology, Engineering and Nutrition, Faculty of Engineering, LTH, Lund University, 2007.

[6] De Prada, C., Merino, A., Pelayo, F., Acebes, F., Alves, R. A simulator of Sugar Factories for Operator Training, AfoT 2003, II International Workshop on Information Technologies and Computing Techniques for the Agro-Food Sector, Barcelona, Spain, November 27-28, 2003, Monograph CIMNE No-86.

[7] Bäckman, J., Edvall, M. Using Modelica and Control Systems for Real-time Simulations in the Pulp. Proceedings of the 4th International Modelica Conference, March 7-8, 2005, pp. 579-583, Hamburg University of Technology (TUHH), Germany. Organized by the Modelica Association and TUHH. Available at www.modelica.org

[8] Skoglund, T., Årzén, K-E. and Dejmek, P. Dynamic object-oriented heat-exchanger models for simulation of fluid property transitions. International Journal of Heat and Mass Transfer, 2006, 49, pp. 2291-2303.

[9] Skoglund, T. and Dejmek, P. A dynamic object-oriented model for efficient simulation of fluid dispersion in turbulent flow with varying fluid properties. Chemical Engineering Science, 2007, 62, pp. 2168-2178

[10] Skoglund, T. and Dejmek, P. A dynamic object-oriented model for efficient simulation of microbial reduction in dispersed turbulent flow. Journal of Food Engineering, 2008, 86, pp. 358–369.

[11] Skoglund, T. and Dejmek, P. Fuzzy traceability – A process simulation derived extension of the traceability concept in continuous food processing. Trans IChemE Part C, Food and Bio products Processing, 2007, 85 (C4) pp. 354-359.

[12] Tummescheit, H. Design and Implementation of Object-Oriented Model Libraries using Modelica, Doctoral thesis, Lund University, 2002.

[13] Iwanitz, F., Lange, J., OPC Fundamentals, Implementation, and Application, 3rd Ed., Hüthig Verlag Heidelberg, 2006.

[14] The OPC Foundation. http://www.opcfoundation.org

Session 6d

Mechanical Systems & Applications

Automatic Model Conversion to Modelica for Dymola-based Mechatronic Simulation

Tamás Juhász, M. Sc. and Ulrich Schmucker, Dr. Sc. techn.
Fraunhofer Institute for Factory Operation and Automation, Magdeburg, Germany
Tamas.Juhasz@iff.fraunhofer.de and Ulrich.Schmucker@iff.fraunhofer.de

Abstract

Virtual product development allows us to recognize and evaluate the characteristics of a new product on the basis of simulation at an early stage without having to build a physical model. Currently the most, widely spread commercial CAE systems do not offer direct support to external dynamic simulation applications. Conversely, dynamic simulation of a detailed model is required to maintain good correlation between the behaviour of the real product and its virtual counterpart. In this paper it will be presented, that using a partially automated workflow a convenient Modelica model translation can be achieved from the output of a mechanical CAD system, allowing the final model to be extended independently with new elements from other simulation domains, considering Dymola-based multi-domain simulation. A .NET-based integrated tool for mechatronic model editing and online / offline visualization support using advanced 3D (and stereo) techniques will also be emphasized in this article.

Keywords: Pro/Engineer, Mechatronics, Collision, Modelica, Dymola Simulation, Stereo, 3D Visualization

1 Introduction

Virtual engineering offers a completely new aspect of product development, as thereby all sections of the product life cycle can be independently analyzed and in parallel continuously optimized in the virtual world. Simulation makes the practical verification of the desired behaviour possible in early development stages.

It is very cumbersome to manually create a parametric simulation model representing a complex product that has been designed in a CAD system. Additionally it is often the case that in machine production a family of component parts with varying parameters has to be designed repeatedly. Nevertheless the product planning is usually an iterative practice: some internal model parameters must be fine-tuned, according to model assessment or verification processes. This implies that an automated model conversion is highly demanded in order to accomplish a good workflow. The designing engineer can inspect the behaviour of the given virtual product by utilizing a dynamic simulation of that. For a convenient iterative workflow a solution have to be provided to automate the conversion between the standard output format of the source CAD system and the input format of the target simulator.

In this article it will be presented that using *Robot-Max*, our .NET-based mechatronic model authoring and visualization application mechanical CAD data from the widely-spread Pro/Engineer CAD environment can be imported, new mechatronic components can be added, thus multi-domain Modelica models can be generated and the results of the Dymola-based multi-domain simulation can be visualized in a convenient way, even in 3D stereo using various 3D technologies, for example by exploiting autostereo monitors, polarizer- or liquid crystal shutter glasses.

2 Translation from CAD data to Modelica models

We have interposed an own developed tool into the design workflow to achieve automated conversion to Modelica models from a Pro/E CAD model assembly (e.g. for mechanics: geometry, mass / inertia parameters, joints).

Similar work has been done in [1], but using the AutoCAD Mechanical Desktop system, and a different, shallower structure of Modelica models. Our approach allows a 3rd party to extend the mechanical model with additional elements from other engineering domains in such a way that a designer can still change / fine tune parameters in the CAD environment (and re-export the mechanical model), without sacrificing the extra work that another expert might have already done within the other model domains, where there might be connections to the previous mechanical model.

2.1 Basic steps of the translation process

There is a large amount of commercial and non-commercial applications (e.g.: "3D_Evolution" or "TransMagic") available on the market offering native conversion between common standard (STEP, IGES) and other well-known (AutoCAD, CATIA, Inventor, Pro/Engineer, SolidWorks, Unigraphics, etc.) CAD data formats. Thus without subsequent restrictions it is assumed our source data is available in the format that our CAD system (Pro/Engineer) is able to import.

The translation from a CAD source file to Modelica description needs the following basic steps:

- Assuming the CAD model has already been imported into *Pro/Engineer*, you can export the hierarchy and geometry information of the actual model to VRML files simply through a click over the File menu "Save as…" command. Note that in a general case you get a main hierarchy file and the geometries of the subsequent parts in separate .WRL files. Geometry information is essential if you want to model collision between the parts during the simulation (see section 3.1 for further information).
- *SimMechanics* is a single-domain extension of the Simulink environment developed by MathWorks, and can be used for modelling and simulation of mechanical systems. Under Pro/Engineer environment the freely available *Pro/E-to-SimMechanics* plug-in [2] lets you export the given CAD assembly to a single (so called "Physical Modeling XML") descriptor file, which is invented to ease the generation of SimMechanics models out of Pro/E data in an automated way. The result XML file contains global hierarchy-, constraint- and physical parameter information (inertia-tensors, masses, etc.), but no geometries at all.
- We developed an application (it is called *RobotMax*) the core logic of which processes the aforementioned XML descriptor file matching with VRML hierarchy/geometry files, thus generating an internal multibody model out of the CAD information. In *RobotMax* the internal (original) model can be extended interactively with various electromechanical elements (e.g.: with parametrical motors from a model library: see section 4.1) to form a more complex mechatronic (multi-domain) model. Finally, our tool is able to export its final mechatronic model to Modelica models using the built-in conversion module, and on demand by the same time it propagates the geometry to DXF format mesh files, in order to use those as visualizing shapes in Dymola environment.

2.2 Building a draft hierarchy out of XML information

A single "Physical Modelling XML" file enumerates all parts (= XML bodies) in the original root CAD assembly (= XML subsystem) from which it was created. The special *RootGround* part represents a fixed point in the environment. Each normal XML rigid body entry contains information about the physical parameters (mass, inertia, surface area, volume, etc.) of the given Pro/E part and has at least two coordinate frames in *World* space: the one that defines the location of the centre of gravity (*CG*) of the rigid body, the other that shows the origin transformation of the body's geometry (*CS1*). XML bodies can carry any number of additional frames (*CS2, CS3* …), which all have a unique integer ID: these unique numbers are used by us to find the corresponding parts between joints.

As it was mentioned before, the XML file also contains information about joints, which represent the constraints of the original CAD assembly. Each XML joint (J_i) has two integer IDs that are uniquely referencing two different frames (these are named "*Base*" and "*Follower*" in a SimMechanics model).

The special *weld* joints are used to mount two rigid parts together with no degrees of freedom left between those. There can also be a series of primitive joints between two frames, representing various degrees of rotational / translational freedom between those parts. In the hierarchy this always implies the following sequence: "*Follower*" → "J_1" → … → "J_N" → "*Base*", where "a → b" shows that "a" is the child of "b" in the hierarchy (i.e. it inherits all transformation from that). Using the XML Joints' frame references you could build a skeleton (a draft hierarchy) of XML bodies. Unfortunately this does not imply automatically that the final hierarchical model is also ready: the geometries of the possibly colliding (but point-sized so far) bodies are still missing at this point.

In CAD systems it is quite often the case that more parts in an assembly share the same name (you can imagine a "template" part that has been used many times as a building element). On the contrary, in case of the target language Modelica, the variable names must be unique inside each model. Via translating the mechanical CAD information, a single, pure mechanic Modelica model has to be generated first. This initial model contains only the parametrical bodies and the mechanical joints, which are connected by "connect" Modelica clauses. All exported bodies and joints must have an individual, unique instance name.

As the auto-generation of VRML and XML files are independently done, the partially auto-generated names inside the result files (e.g.: "*Obj01*", "*Obj02*" vs. "*Obj*", "*Obj-1*") will neither be globally unique nor match each other. In order to find the corresponding entities both in VRML and in XML domains, you have to follow a sophisticated procedure. This is in the most cases inevitable, because there are usually less XML bodies than actual VRML geometries. You must know which geometries form together a single rigid body, if you want to have a consistent collision handling during the simulation.

2.3 Matching hierarchies in XML and VRML domains

All VRML geometry nodes have a homogenous transformation matrix (which can arise derived from their respective parents, recursively), from which you can derive their global pose (position and orientation) in the 3D world. This derived 4x4 matrix is also used to transform the local vertices of a given VRML shape into the global (*World*) coordinate system during rendering, for example. Fortunately the same pose information is also included in XML file with *CS1* frame of the XML bodies.

First you have to search for matching the position of all *CS1* frames (extracted from XML) with an origin frame location from VRML geometries being just imported. If there are more bodies having the same CS1 frame position, you continue filtering the "candidates" by differences in *CS1* frame orientation. Assuming there are still more than one parts with the same global pose in *CS1* (which is blissfully a rare case), you can compare the names of the XML bodies and VRML shapes (namely just their prefixes: e.g.: "*Obj01*" will match with "*Obj*" or "*Obj-1*") to find the highly demanded single positive match. It is hardly imaginable that there are more parts in the CAD assembly with exactly the same pose and name. This should indicate that there is an error in the source CAD plan.

It is often the case, that there are subsequent levels in the VRML geometry hierarchy: in this case these child shapes are to be merged into the same higher level geometry.

After assigning the VRML geometry to the corresponding XML bodies, the final, pure mechanic multibody model can be finally generated. For this sake, the necessary physical parameters (masses, locations of *CG* frames, inertia tensors) have to be substituted into the final Modelica actors' parameters.

3 Expanding the standard Modelica library

The Modelica Standard Library is a standardized and free package that is developed together with the Modelica language by the international Modelica Association [8]. The Mechanics Multibody Library (MML) is a package in the main library providing 3-dimensional mechanical components to model mechanical systems in a convenient way.

The MML does not include support for rigid body collision handling. Handling contacts between mechanical objects can be very important in many disciplines of mechatronic simulation (e.g.: robot manipulating tasks).

3.1 Collision Handling

We extended the MML library with support for collision handling using a spring and damper material model, suggested by the article [3], but based on more robust Bullet collision library in our recent implementation. We discussed the details of our implementation in [4]. In this section it will be presented what sort of new Modelica components have been developed for this purpose.

The basic *World* model in MML represents a global coordinate system fixed in 3D space origin. The behaviour of the basic World model has been extended via inheritance: from the original base model a *Collision Manager* (CM) subclass has been inherited that is responsible for collision handling in simulation of multibody systems.

The standard Modelica implementation of rigid bodies (see *BodyShape* component in MML) needed also to be extended to handle collision (via communication with the CM). Our *Actor* class encapsulates the physical kinematic- (pose, velocity and acceleration), dynamic- (mass, location of centre of gravity and inertia tensor) and material- (stiffness and restitution) parameters (also initial values of those) of a rigid body. Note that actors don't have any geometry information.

The *Shape* class extends the *Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape* class, offering 3D visualization possibilities in Modelica environment. Each *Shape* instance must connect to a single actor with a respective 4x4 transform of local origin of the geometry. These objects represent the geometry of the rigid body they connect to. The *Collider* class is the subclass of Shape, which can serve the collision geometry of that part. In order to ease the export to Modelica, these classifications of new shape classes make handling of geometry orbicular from both the aspect of Modelica and *RobotMax*, our CAD translator application.

For online, real time visualization support (see section 5.3) each *Shape* instance has a 7 component pose vector (3D position + a quaternion orientation) simply assigned by their local origin frame's pose. There is a pre- allocated ***P*** pose matrix (dimensions: 7 by N) reserved for the N shape objects, stored in a shared memory. The shared memory is implemented in a C++ class, is compiled to a DLL and it offers C interfaces to Modelica. The columns of ***P*** are updated every simulation step by the respective shapes' poses using the external C function invocation *setPose( )* from the Shape instances' Modelica source.

The singleton *Collision Manager* instance stores information about the positions, orientations, angular- and linear velocities of all *Colliders* existing in the global collision set. At the initial phase of the simulation each Collider instance reports the CM its geometry, which cannot change during the simulation. The CM updates the external collision forces on each colliding shape in each simulation step. These shape instances propagate the external collision force through their connectors to the respective actor instances.

Unfortunately the Modelica language specification being used at the development time (it was version 2.2.1) did not allow having a collection containing polymorphic references to the instances of a user defined class (i.e. abstract models) themselves: our Modelica arrays can contain only basic data types. This introduces a little performance loss: the CM has to store duplicated information in separate arrays about the positions, orientations, angular- and linear velocities of all shapes existing in the global collision set.

Some shapes can be *individually* excluded from collision handling via disabling their collision flags (for example in draft motion tests). On the other hand, sometimes it is desired (usually for simplified models) to allow *also pairs* of bodies to constantly interpenetrate each other during the simulation, without any internal tension or force between them. For this purpose the user of the extended library can assign a matrix to the CM containing the IDs of unwanted collider pairs.

There is a permanent bidirectional communication between the colliders and the collision manager. The external collision response forces and -torques that are calculated and responded by the CM, act together on the given actors automatically as it was told before. This is due to the behaviour of bidirectional Modelica "*connect*" equations.

The Modelica standard has a well-designed interface to external software modules [5] (e.g.: Fortan or ANSI C: sometimes allowing more powerful algorithm implementation). Accordingly, we were not confined to implement the whole collision manager class in pure Modelica. For the algorithmic core functionality of collision detection and -response calculation the C interface could be used:

For each supported collider shape type a C++ class had to be implemented, having parameters similar to their Modelica counterparts. These classes are instantiated at the initial phase of simulation: as soon as a Modelica *Collider* is initiated, the corresponding C++ constructor is invoked from Modelica code, through our C interface wrapper.

In each simulation step the C++ part of the CM updates the pose of all C++ shapes via their Modelica counterparts' pose, and invokes the main method to query the actual collision forces and -torques for all active geometry in the scene. In the background the free Bullet library [7] is being used to query collision information among our rigid bodies (these are being treated as independent ones, no joint-constraints are introduced here). The penetration checking functionality of Bullet is done the following way:

For each pair of shape types, a certain collision algorithm is assigned, by using an internal dispatcher. The collision detection library part of Bullet can retrieve contact points between any triangular geometry types (for some concave-convex case the primitive geometry types – such as sphere or cylinder – need to be tessellated to triangles). The used algorithms are a modified version of the GJK algorithm [6] with the EPA - Expanding Polythope Algorithm for convex-convex cases, and GIMPACT for the cases involving concave geometry.

Our pair-wise collision response calculation method (spring and damper technique: dependent on penetration velocity, relative motion of colliding parts, material stiffness- and restitution parameters) is discussed in [4]. A single invoke on the external C++ library can solve the collision response for the whole system at once, thus the external forces on the Modelica colliders can be updated in each simulation step.

3.2 Abstract joint models allowing domain independency

Our purpose is to simulate articulated multibody mechatronic systems having multiple rigid bodies connected by joints. The original test CAD models, which we seized to test our conversion process, have either no motor information, or this information is not accessible from the outside (i.e. cannot be exported from) the CAD system. This implies that in *RobotMax* all XML joints will be converted first to abstract ones by default (prismatic, revolute and spherical joints, or serial combination of those are supported in the entire system). Note that spherical joints (allowing 3 rotational degrees of freedom in their coupling centre point) are always passive: they cannot be actuated in the original manner.

We implemented abstract joint models in Modelica for prismatic and revolute joints, which are exactly the pure mechanical constraints, representing the allowed single degree of translational or rotational freedom between their 3D frame connectors. These abstract joint models have a one dimensional *'Drive'* flange, as you can see on Figure 1:

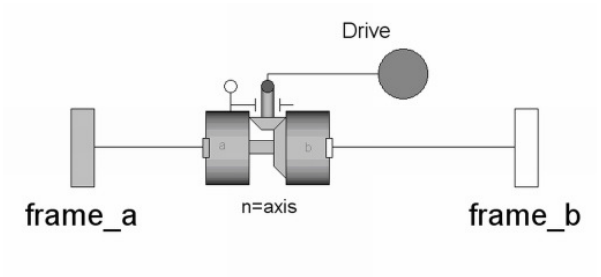


Figure 1: Abstract model of a revolute joint

If the *'Drive'* flange connector is not connected from the outside in the container Modelica model, an abstract joint will be equivalent to an ideal, free joint. On the contrary, connecting a motor's drive part to the drive flange of these joints makes actuated joints in the final mechatronic model. For the details please refer to section 4.2.

Using this abstraction we could decouple the pure mechanical model from other electromechanical components: these can be exported to a separate top-level Modelica model.

4 Adding electromechanical components to the internal model in RobotMax

Our goal is to support the simulation of the dynamic behaviour of the product being designed in the source CAD system. Assuming you have a CAD model of an industrial robot having a few joints that should be actuated by motors, you could easily ask what kind of motors should be applied in order to achieve a pre-defined speed along the desired path of the tool centre point, or to stay below the maximal allowed positioning error.

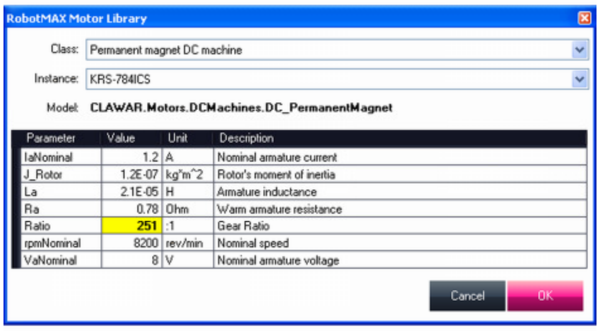
Unfortunately, we can't seize so far any description of the possibly occurring electromechanical components from the Pro/E CAD system, which we could embed automatically into the final mechatronic model at the end of the conversion process.

You can say that the requirement of having motors in an articulated multibody system is more than desirable. Without such elements you could not simulate / verify the active dynamic behaviour of a moving virtual structure.

4.1 The Motor Library in RobotMax

We developed an XML-based extensible Object Library that can contain parametric components of any modelling domain. The special modelling domain of electromechanical components (motors) will be emphasized in this section. For example the Motor Database inside the Object Library contains motor classes (e.g.: DC motors or induction machines) as entries.

Every class in the library has an absolute path reference to the Modelica implementation of the model represented by it. These classes enumerate their parameters, which all must have a unique name (referring to their respective variables in the Modelica model). Each parameter must also have a type (Float, Integer, String, etc.) and a Boolean flag indicating whether its actual value is editable by the user. For example changing the gear ratio parameter in a final motor instance is still allowed. A general parameter can also have a physical unit (like 'Ohm' or 'kg·m^2': one should use SI standard units, unless it is not specified here differently), minimum / maximum limits and a descriptive comment optionally.



Class: Permanent magnet DC machine

Instance: KRS-784ICS

Model: CLAWAR.Motors.DCMachines.DC_PermanentMagnet

Parameter	Value	Unit	Description
IaNominal	1.2	A	Nominal armature current
J_Rotor	1.2E-07	kg*m^2	Rotor's moment of inertia
La	2.1E-05	H	Armature inductance
Ra	0.78	Ohm	Warm armature resistance
Ratio	251	:1	Gear Ratio
rpmNominal	8200	rev/min	Nominal speed
VaNominal	8	V	Nominal armature voltage

Figure 2: An example entry in our motor library

Figure 2 shows a screenshot of our library's browser dialog displaying the parameters of our DC permanent magnet motor class. The user can also edit here the highlighted gear ratio, before it will be inserted to the internal model in *RobotMax*.

The instances of a class are enumerated in the library after each class declaration, defining the actual / initial values for each parameter in all occurring instances. The instances must have a global unique ID (a primary key along that column), which is always required in a relational database (e.g.: during searching).

4.2 Our actuated joint models in Modelica

For the most mechatronic simulation purposes one has to set continuous reference values of the active joints in the system (defining position, velocity or acceleration parameters of those) in order to make the parts follow a pre-defined trajectory. A well-designed controller should be introduced that minimizes the error between the actual and the reference values of each joint in every single moment.

We implemented 1-1 parametrical, translation- and rotation based drive model in Modelica (for prismatic- and revolute joints, respectively), which contain a separated control- and actuator part, and is decoupled from the given joints' mechanical part. Figure 3 shows our general model for an actuated joint:

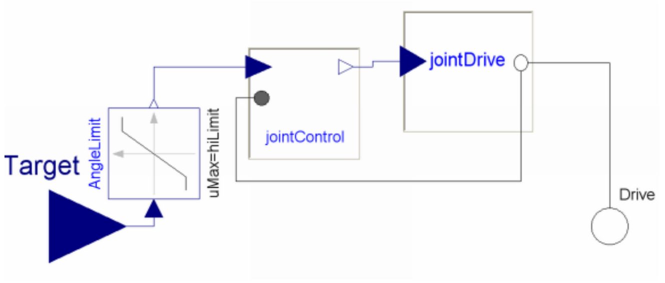


Figure 3: The schema of our joint drive subsystem

The general *"jointDrive"* actuator model has a replaceable motor and gear component. If a component is declared *replaceable* in Modelica it means that one can transparently exchange the implementation of this part with another model, unless the given external connector interfaces are kept intact.

As it was told before, the pure mechanical model was exported to a separate Modelica text file. As long as the names of the joint entries are not changing, we will find the way to connect the respective '*Drive*' connectors in both models. Thus the user can experiment with fine-tuning the motor parameters and simulate the new model without the need to redo the CAD / XML conversion process from the beginning again.

5 Simulation and visualization

The workflow presented so far had been finally extended with a motion planning task, which can be carried out right before the Modelica export step, in order to define a continuous-time function in a convenient way for each joint's path.

5.1 Defining motions and simulating the model

RobotMax – our .NET-based CAD translator / environment editor application – offers keyframe-based motion planning and has built-in support for inverse kinematics that was used in the following example to model a palette manipulating motion with an industrial robot model. The user can also fine-tune the motion by interactively adjusting the values of the selected servos.

The following image sequence shows the three basic steps of the CAD to SIM process (in Pro/E → RobotMax → Dymola order), presented so far:

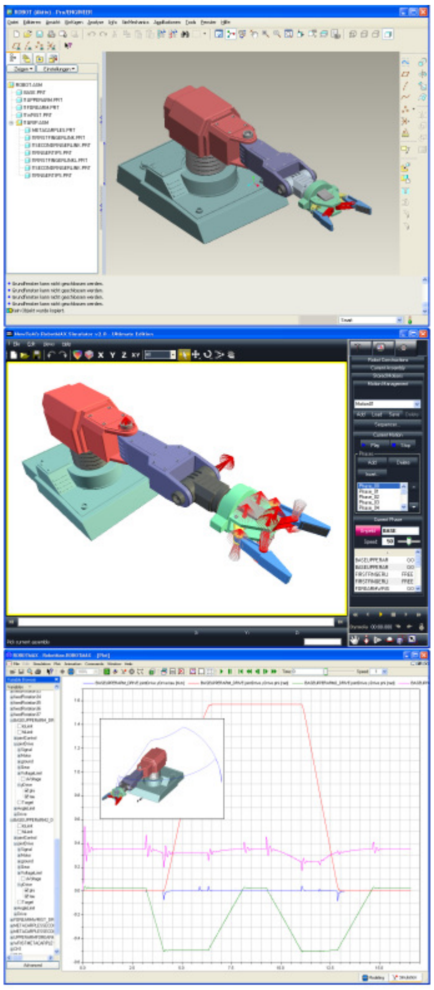


Figure 4: The Dymola simulation of an industrial robot-arm designed in Pro/Engineer and converted by RobotMax using the presented workflow

5.2 Creating test scenarios in RobotMax

The *Scene Editor* in *RobotMax* can be used to create various static / dynamic scenarios, allowing testing the interaction between the actual virtual product (that is being designed in CAD) and its environment, which is usually modelled separately or is sometimes simply neglected. For example a bumpy road can be added to the 3D world in case of testing a new car suspension assembly. VRML geometry can be imported, or the user can create new static / dynamic objects from primitives with the interactive tools in *RobotMax*. The parameters (materials, dimensions, positions, etc.) are interactively changeable (in case better precision is needed, can be set also manually) and the modifications can be undone, thus allowing an iterative approach of testing with various scenarios. Multiple viewports and various alignment tools are helping you to make the test setup as precise (and as informative) as possible.

5.3 Problem of online visualization

The Dymola simulator [9] being used in our project has a 3D viewer (animation) support for multibody models containing 3D geometry, but has a limited functionality and is not user friendly enough.

If you want to visualize the simulation results from a 3rd party application while Dymola is running in the background, the poses of the various geometries have to be gathered and transmitted to the viewer application online. Although Dymola stores the output of a simulation in a file (in Matlab® format), this file is exclusively locked: thus no other application can read from that file until the simulation finishes. Another solution had to be found to access pose information during the simulation.

5.4 Visualization in RobotMax

Our idea to transfer data to a viewer was to query it from the shared memory containing the ***P*** matrix of actual shape-poses (see section 3.1). On a viewer side there is usually no need to update the pose of the objects after each solver step (e.g.: a step size of 1 ms would lead to 1000 frames/second required refresh rate). The problem can be turned around: you can retrieve (poll) the actual pose of any shape from the shared memory at a desired, smaller frequency (e.g.: 50 Hz).

The *RobotMax* has all functionalities a modern 3D viewer application requires, with multiple orthogonal or perspective viewports, interactive camera setup and built-in support for advanced 3D visualization techniques – including real 3D methods such as autostereo (for monitors with lenticular lens layer), time-interleaved (for liquid crystal shutter goggles), spectral-interleaved (for red-cyan anaglyph spectacles) or dual output (for two projectors and polarizer glasses) – representing the actual internal model in 3D. For a broader overview of these techniques please refer to the article [10].

If the user switches *RobotMax* to online visualization mode, it polls the pose information for each shape continuously and updates the viewports with the preset frequency only.

In order to be able to inspect the simulation results multiple times, there is a support for offline visualization, of course. A simulation output file can be parsed by an external application only after it has been completely written and released by Dymola. In offline visualization mode RobotMax invokes Dymola with the generated Modelica models (according to the process described in Section 2) and waits for the lock of the result file to be released.

The sequences of samples of each simulation signal are stored in this file, including input / output variables, state variables and their derivatives. In *RobotMax* after parsing all the exported signals into memory, the signals belonging to the world transformation matrices are used to setup a keyframe animation, which can be sought and played back from a desired position at the desired speed.

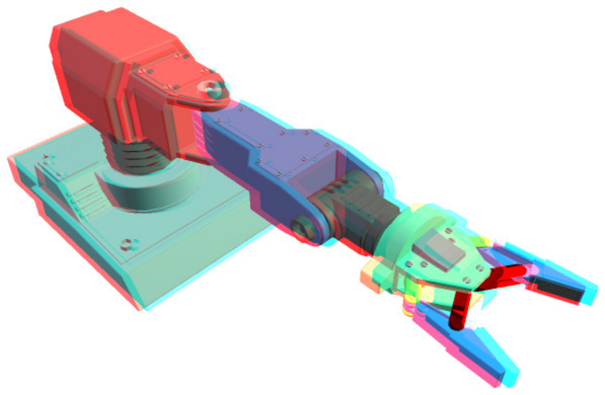


Figure 5: anaglyph mode 3D visualization, screenshot

On Figure 5 a screenshot can be seen that was taken in *RobotMax* showing the red-cyan spectral-separated anaglyph stereo image of the previous industrial robot-arm example at initial pose at the very beginning of the simulation.

6 Conclusion and future work

A highly automated, convenient conversion workflow from Pro/Engineer CAD data to multi-domain Modelica simulation models has been presented in this paper. The relevant online / offline visualization methods – with advanced 3D techniques within the same integrated tool used for model translation – were also discussed here. For a schematic overview of the presented workflow see Figure 6.

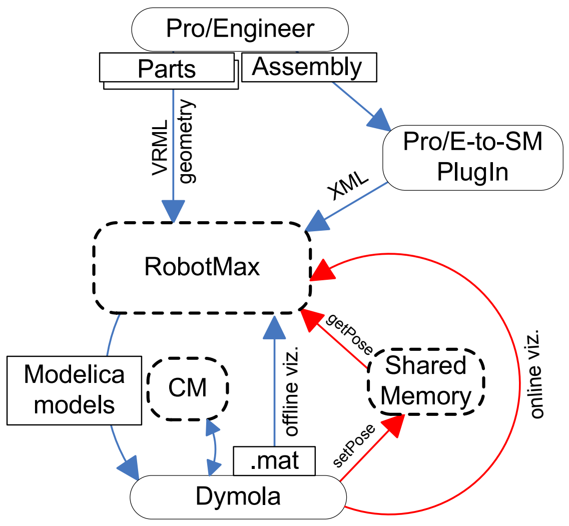


Figure 6: schematic process overview

In our future implementation we will use the Pro/ToolKit API to directly access data of other, newly added components in Pro/Engineer WildFire 3, such as springs, dampers and motors, and translate these elements also into the final Modelica models. Using this interface the functionality of the Pro/E-to-SimMechanics plug-in can be completely replaced later by our implementation.

There is a free Modelica library – called BondLib, available at [11] – for bond-graph represented analog electronic circuits, including a full implementation of Spice models. The presented CAD conversion process can be generalized to introduce complex models of electrics / electronics domain, to be converted to Modelica. We will investigate the possibilities to introduce *OrCAD* layout plans and *P-Spice* models into our virtual mechatronic workflow.

In the next version of our *RobotMax* tool we will implement a 2D Plot functionality to be able to inspect simulation signals as 2D curves in a given viewport. Our collision response calculation in tangential space (which is currently very simplified) has to be improved to achieve more realistic friction forces and –torques between contacting bodies.

References

[1] Engelson, V.; Bunus, P.; Popescu, L.; Fritzson, P.: “Mechanical CAD with Multibody Dynamic Analysis Based on Modelica Simulation”; In Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS-2003), September 18-19, 2003, Västerås, Sweden

[2] MathWorks: SimMechanics Translators: http://www.mathworks.com/products/simmechanics/description5.html

[3] Otter, M.; Elmqvist, H.; Díaz López, J.: ”Collision Handling for the Modelica Multi-Body Library”; In Proceedings of the 4th International Modelica Conference, March 7-8, 2005, Hamburg, pp. 45-53

[4] Juhasz, T.; Konyev, M.; Rusin, V.; Schmucker, U.: ”Contact Processing in the Simulation of CLAWAR”; In Proceedings of 10th CLAWAR International Conference, 16-18 July 2007, Singapore, pp. 583-590.

[5] Fritzson, P.: “Principles of Object-Oriented Modeling and Simulation with Modelica 2.1”, Wiley Press 2004, ISBN 0-471-471631, pp. 311-322.

[6] Gilbert, E. G.; Johnson, D. W.; Keerthi, S. S.: ”A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space”; In IEEE Trans. Robotics and Automation 4 (Vol2), April 1988, pp. 193-203.

[7] Bullet 3D Collision Detection Library: http://www.bulletphysics.com/Bullet

[8] Modelica Association – http://www.modelica.org

[9] Dynasim AB: Dymola 6 – http://www.dynasim.com/index.htm

[10] Juhasz, T.; Vajta, L.: “The Role of 3D Simulation in the Advanced Robotic Design, Test and Control”, International Journal of Advanced Robotic Systems – Cutting Edge Robotics 2005, ISBN 3-86611-038-3; pp. 47-61.

[11] Cellier, F.: BondLib – Modelica library: http://www.modelica.org/libraries/BondLib

Modelica Implementation of the Skateboard Dynamics

Ivan I. Kosenko[1], Alexander S. Kuleshov[2]
[1] Moscow State University of Tourism and Service, Department of Engineering Mechanics,
Glavnaya str., 99, Cherkizovo-1, Moscow reg., 141221, Russia
[2] Lomonosov Moscow State University, Department of Mechanics and Mathematics,
Leninskie Gory, Main Building of MSU, Moscow, 119991, Russia

Abstract

In the present paper analysis and simulation are performed for a simplest model of a skateboard. We suppose that the rider control is absent during the motion. Equations of motion of the model are presented and their stability analysed in brief.
Modelica implementation of the skateboard dynamics is described as well. Its main featured outlined, and the verification procedures explained. It is pointed out the skateboard can behave in dynamical sense likewise the known example of the rattleback.
Keywords: skateboard; nonholonomic constraints; normal form; contact models, dynamical verification

1 Introduction

Nowadays the skateboarding, the art of riding on a skateboard, is one of the most popular sports. Nevertheless serious researches concerning dynamics and stability of a skateboard are almost absent. At the late 70th – early 80th of the last century Mont Hubbard [1, 2] proposed two mathematical models describing the motion of a skateboard with the rider. To derive equations of motion of the models he used the principal theorems of dynamics. In our paper we give the further development of the models proposed by Hubbard to have an additional possibility to verify the engineering solutions for this type of a vehicle.
Simultaneously to give the further move in field of the sportswear appliances development we created and verified a dynamical model of the skateboard. The model was developed on Modelica, and it is easy to improve it in different directions to be able to investigate the regular riding technique or the interesting tricks performed by the experts while the skateboarding shows.
The skateboard typically consists of a board, two trucks and four wheels, see Figure 1. The modern boards are usually from 78 to 83 cm long, 17 to 21 cm wide and 1 to 2 cm thick. The most essential elements of a skateboard are the trucks, connecting the axles to the board. Angular motion of both the front and rear axles is constrained to be about their respective non-horizontal pivot axes, thus causing a steering angle of the wheels whenever the axles are not parallel to the plane of the board, see Figure 2. The vehicle is steered by making use of this kinematic relationship between steering angles and tilt of the board. In addition, there is a torsional spring, which exerts a restoring torque between the wheelset and the board proportional to the tilt of the board with respect to the wheelset, Figure 3. We denote the stiffness of this spring by k_1.

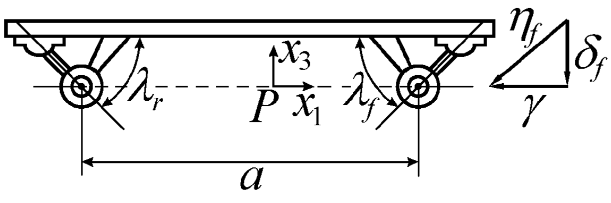


Figure 1: The Skateboard Side View

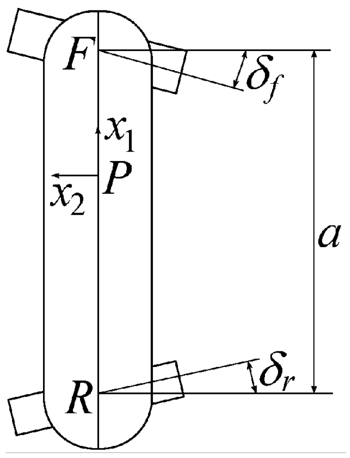


Figure 2: The Skateboard Top View

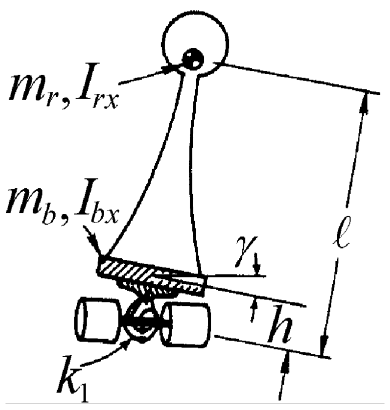


Figure 3: The Skateboard Front/Rear View

2 The Problem Formulation. Equations of Motion.

We assume that the rider, modeled as a rigid body, remains perpendicular with respect to the board. Therefore, when the board tilts through γ, the rider tilts through the same angle relative to the vertical. Let us introduce an inertial coordinate system $OXYZ$ in the ground plane. Let $FR = a$ is a distance between two axle centers F and R of a skateboard. The position of a line FR with respect to the $OXYZ$-system is defined by X and Y coordinates of its center and by the angle θ between this line and the OX-axis, see Figure 4.

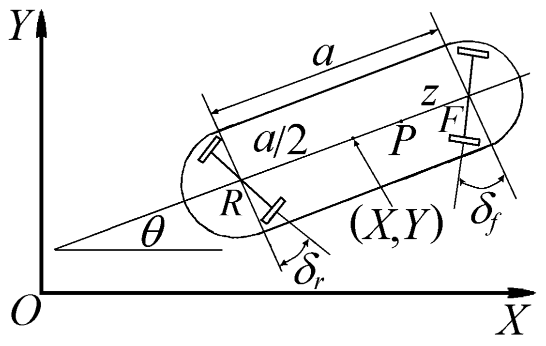


Figure 4: The Basic Coordinate Systems.

The tilt of the board causes the rotation of front wheels clockwise through δ_f and the rotation of rear wheels anticlockwise through δ_r, Figures 2, 4. The wheels of a skateboard are assumed to roll without lateral sliding. This condition is modeled by constraints, which may be shown to be nonholonomic

$$\begin{aligned} &\dot{Y}\cos(\theta-\delta_f)-\dot{X}\sin(\theta-\delta_f)+\frac{1}{2}a\dot{\theta}\cos\delta_f = 0, \\ &\dot{Y}\cos(\theta+\delta_r)-\dot{X}\sin(\theta+\delta_r)-\frac{1}{2}a\dot{\theta}\cos\delta_r = 0. \end{aligned} \tag{1}$$

Under these conditions velocities of a points F and R will be directed horizontally and perpendicularly to the axles of wheels and there is a point P on the line FR which has zero lateral velocity. Its forward velocity we denote by u. It may be shown, that (see e. g. [1] – [6])

$$u = -\frac{a\dot{\theta}\cos\delta_f\cos\delta_r}{\sin(\delta_f+\delta_r)},$$

$$FP = \frac{a\sin\delta_f\cos\delta_r}{\sin(\delta_f+\delta_r)}, \quad \dot{\theta} = -\frac{u\sin(\delta_f+\delta_r)}{a\cos\delta_f\cos\delta_r}. \tag{2}$$

Using results obtained in [5, 6] we conclude that the steering angles δ_f and δ_r are related to the tilt of the board by the following equations

$$\tan\delta_f = \tan\lambda_f\sin\gamma, \quad \tan\delta_f = \tan\lambda_f\sin\gamma, \tag{3}$$

where λ_f and λ_r are the fixed angles which the front and rear axes make with the horizontal, Figure 1. Using constraints (3) we can rewrite equations (1) as follows

$$\begin{aligned} \dot{X} &= u\cos\theta + \frac{(\tan\lambda_f-\tan\lambda_r)}{2}u\sin\gamma\sin\theta, \\ \dot{Y} &= u\sin\theta - \frac{(\tan\lambda_f-\tan\lambda_r)}{2}u\sin\gamma\cos\theta. \end{aligned} \tag{4}$$

Expressions (2) become

$$FP = \frac{a\tan\lambda_f}{\tan\lambda_f+\tan\lambda_r}, \quad \dot{\theta} = -\frac{(\tan\lambda_f+\tan\lambda_r)}{a}u\sin\gamma. \tag{5}$$

Suppose that the board of the skateboard is located on the distance h above the line FR. The length of the board is also equal to a. The board center of mass is located in its center. As to the rider we suppose that the rider center of mass is not located above the board center of mass, but it is located over the central line of the board on a distance d from the front truck. Let l be the height of the rider center of mass above the point P. Other parameters for the problem are: m_b is the mass of the board, m_r is the mass of the rider; I_{bx}, I_{by}, I_{bz} are the principal central moments of inertia of the board; I_{rx}, I_{ry}, I_{rz} are the principal central moments of inertia of the rider. We introduce also the following parameters:

$$I_x = I_{bx} + I_{rx}, \quad I_y = I_{by} + I_{ry}, \quad I_z = I_{bz} + I_{rz}.$$

It can be proved, see [5], that the variables u and γ satisfy the following differential equations

$$\begin{aligned} &\left(A + (C-2D)\sin^2\gamma + K\sin^4\gamma\right)\dot{u} + \\ &\left(C - 3D + 3K\sin^2\gamma\right)u\dot{\gamma}\sin\gamma\cos\gamma + \\ &B\left(\ddot{\gamma}\cos\gamma - \dot{\gamma}^2\sin\gamma\right)\sin\gamma = 0, \\ &E\ddot{\gamma} + \left(D - K\sin^2\gamma\right)u^2\sin\gamma\cos\gamma + \\ &k_1\gamma - (m_b h + m_r l)g\sin\gamma + \\ &B\left(\dot{u}\sin\gamma + u\dot{\gamma}\cos\gamma\right)\cos\gamma = 0. \end{aligned} \tag{6}$$

Here A, …, E, K are functions of the parameters, namely

$$\begin{aligned}
A &= m_b + m_r, \\
E &= I_x + m_b h^2 + m_r l^2, \\
B &= \frac{m_b h}{2}\left(\tan\lambda_f - \tan\lambda_r\right) + \\
&\quad \frac{m_r l}{a}\left((a-d)\tan\lambda_f - d\tan\lambda_r\right), \\
C &= \frac{m_b}{4}\left(\tan\lambda_f - \tan\lambda_r\right)^2 + \\
&\quad \frac{I_z}{a^2}\left(\tan\lambda_f + \tan\lambda_r\right)^2 + \\
&\quad \frac{m_r}{a^2}\left((a-d)\tan\lambda_f - d\tan\lambda_r\right)^2, \\
D &= \frac{\left(\tan\lambda_f + \tan\lambda_r\right)}{a}\left(m_b h + m_r l\right), \\
K &= \frac{\left(\tan\lambda_f + \tan\lambda_r\right)^2}{a^2}\left(I_y + m_b h^2 + m_r l^2 - I_z\right).
\end{aligned}$$

Thus, equations (4–6) form the closed DAE system for the skateboard motion.

3 Stability of the Skateboard Straight-Line Motion

Equations (6) have a particular solution

$$u = u_0 = \text{const}, \quad \gamma = 0, \tag{7}$$

which corresponds to a uniform straight-line motion of the skateboard. The stability conditions of this particular solution have the following form [1]-[6]:

$$Bu_0 > 0, \quad Du_0^2 + k_1 - (m_b h + m_r l)\,g > 0. \tag{8}$$

From the first condition of (8) we can conclude that the stability of motion (7) depends on its direction. If one direction of motion is stable the opposite direction is necessary unstable. Such a behavior is peculiar to many nonholonomic systems. First of all, we can mention here the classical problem the rattleback motion (aka wobblestone or celtic stone, see e. g. [7]-[9]). In this problem the stability of permanent rotations of a rattleback also depends on the direction of rotation.
Suppose that the coefficient B is positive, $B > 0$. Then for $u_0 > 0$ the skateboard moves in "stable" direction, and for $u_0 < 0$ it moves in "unstable" direction. When $u_0 = 0$ the skateboard is in equilibrium position on the plane. The necessary and sufficient condition for stability of this equilibrium have a form [1]-[6]:

$$k_1 - (m_b h + m_r l)\,g > 0. \tag{9}$$

Assuming that condition (9) holds, let us consider the behavior of the system near the equilibrium position. Solving equations (6) with respect to $\dot{u}$ and $\ddot{\gamma}$ and assuming that u, γ and $\dot{\gamma}$ are small, we can write equations of perturbed motion taking into account the terms which are quadratic in u, γ and $\dot{\gamma}$ as follows

$$\dot{u} = \frac{B\Omega^2}{A}\gamma^2, \quad \ddot{\gamma} + \Omega^2\gamma = -\frac{Bu\dot{\gamma}}{E}, \tag{10}$$

where we introduce the following notation

$$\Omega^2 = \frac{k_1 - (m_b h + m_r l)\,g}{E}.$$

Note, that the linear terms in the second equation of the system (10) have a form which corresponds to a normal oscillations. For investigation of nonlinear system (10) we reduce it to a normal form [10]. To obtain the normal form of the system (10) first of all we make a change of variables and introduce two complex-conjugate variables z_1 and z_2 such that

$$\gamma = \frac{z_1 - z_2}{2i}, \; \dot{\gamma} = \frac{z_1 + z_2}{2}\Omega, \; u = z_3.$$

In variables z_k, $k = 1,2,3$ the linear part of the system (10) has a diagonal form and the derivation of its normal form reduces to separating of resonant terms from the nonlinearities in the right-hand sides of the transformed system (10). Finally, the normal form of the system (10) may be written as follows

$$\begin{aligned}
\dot{z}_1 &= i\Omega z_1 - \frac{B}{2E} z_1 z_3, \\
\dot{z}_2 &= -i\Omega z_2 - \frac{B}{2E} z_2 z_3, \\
\dot{z}_3 &= \frac{B\Omega^2}{2A} z_1 z_2.
\end{aligned}$$

Introducing real polar coordinates according to the formulae

$$\begin{aligned}
z_1 &= \rho_1\left(\cos\sigma + i\sin\sigma\right), \\
z_2 &= \rho_1\left(\cos\sigma - i\sin\sigma\right), \\
z_3 &= \rho_2
\end{aligned}$$

we obtain from the system (10) the normalized system of equations of perturbed motion which is then split into two independent subsystems:

$$\dot{\rho}_1 = -\frac{B}{2E}\rho_1\rho_2, \quad \dot{\rho}_2 = \frac{B\Omega^2}{2A}\rho_1^2, \tag{11}$$

$$\dot{\sigma} = \Omega. \tag{12}$$

Terms of order higher than the second in (11) and those higher than the first in ρ_k, $k = 1,2$ in (12) have been omitted here.

In the ε-neighborhood of the equilibrium position the right-hand sides of equations (11) and (12) differ from the respective right-hand sides of the exact equations of perturbed motion by quantities of order ε^3 and ε^2 respectively. The solutions of the exact equations are approximated by the solutions of system (11–12) with an error of ε^2 for ρ_1, ρ_2 and of order ε for σ over time interval of order $1/\varepsilon$. Restricting the calculations to this accuracy, we will consider the approximate system (11–12) instead of the complete equations of perturbed motion.

Equation (12) is immediately integrable, and we obtain

$$\sigma = \Omega t + \sigma_0.$$

System (11) describes the evolution of the amplitude ρ_1 of the board oscillations and also the evolution of the velocity ρ_2 of a the skateboard straight-line motion. One can see that this system has the first integral

$$E\rho_1^2 + \frac{A}{\Omega^2}\rho_2^2 = An_1^2, \tag{13}$$

where n_1 is a constant, specified by initial conditions. We will use this integral for solving of the system (11) and for finding the variables ρ_1 and ρ_2 as functions of time: $\rho_1 = \rho_1(t)$, $\rho_2 = \rho_2(t)$. Expressing ρ_1^2 from the integral (13) and substitute it to the second equation of the system (11) we get

$$\dot{\rho}_2 = \frac{B}{2E}\left(\Omega^2 n_1^2 - \rho_2^2\right). \tag{14}$$

The general solution of equation (14) has the following form:

$$\rho_2(t) = \frac{\Omega n_1\left(1 - n_2\exp\left(-\frac{B\Omega n_1}{E}t\right)\right)}{\left(1 + n_2\exp\left(-\frac{B\Omega n_1}{E}t\right)\right)}, \tag{15}$$

where n_2 is a nonnegative arbitrary constant. Now, using the integral (13), we can find the explicit form of the function $\rho_1(t)$ in the following way

$$\rho_1(t) = 2n_1\sqrt{\frac{An_2}{E}}\frac{\exp\left(-\frac{B\Omega n_1}{2E}t\right)}{1 + n_2\exp\left(-\frac{B\Omega n_1}{E}t\right)}. \tag{16}$$

Let us consider the properties of the solutions (15), (16) of system (11) and their relations to the properties of the skateboard motion. System (11) has an equilibrium position

$$\rho_1 = 0, \quad \rho_2 = \Omega n_1. \tag{17}$$

These particular solutions can be obtained from general functions (15–16) if we suppose in that functions $n_2 = 0$. An arbitrary constant n_1 can be both positive and negative. The positive values of this constant correspond to the skateboard straight-line motions with small velocity in "stable" direction and the negative ones do in "unstable" direction. Indeed, if we linearize equations (11) near the equilibrium position (17) we get

$$\dot{\rho}_1 = -\frac{B}{2E}\Omega n_1\rho_1, \ \dot{\rho}_2 = 0.$$

Thus, for $n_1 > 0$ the equilibrium position (17) is stable and for $n_1 < 0$ it is unstable.

Evolution of the functions ρ_1 and ρ_2 gives the complete description of behavior of a skateboard with small velocities. Let us suppose, that at initial instant the system is near the stable equilibrium position $(n_1 > 0)$ and $\rho_2(0) \geq 0$, i. e. $n_2 \leq 1$. The case of $n_1 > 0$, $n_2 > 1$ is similar to the case of $n_1 < 0$, $n_2 < 1$, which will be investigated below. These initial conditions correspond to the situation correspond to the skateboard to take the small velocity

$$\rho_2(0) = \Omega n_1\frac{1 - n_2}{1 + n_2}$$

in the "stable" direction at initial instant. Then in the course of time the "amplitude" ρ_1 of the board oscillations decreases monotonically from its initial value

$$\rho_1(0) = \frac{2n_1}{1 + n_2}\sqrt{\frac{An_2}{E}}$$

to zero, while the velocity of a skateboard ρ_2 increases in absolute value. In the limit the skateboard moves in stable direction with a constant velocity Ωn_1, see Figure 5–6.

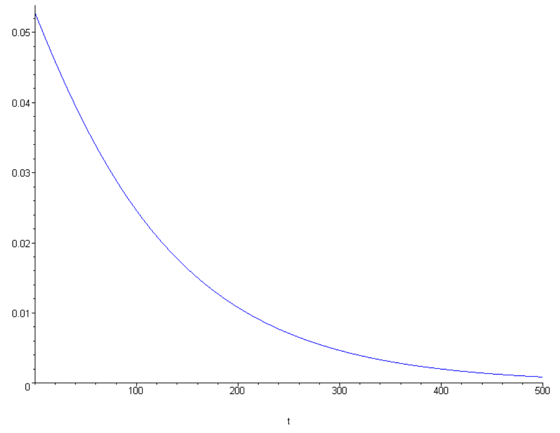


Figure 5: Evolution of the Amplitude ρ_1 of the Board Oscillations in Time for the Case $n_1 > 0$, $n_2 \leq 1$.

Suppose now that at initial instant the system is near the unstable equilibrium position $n_1 < 0$. Suppose

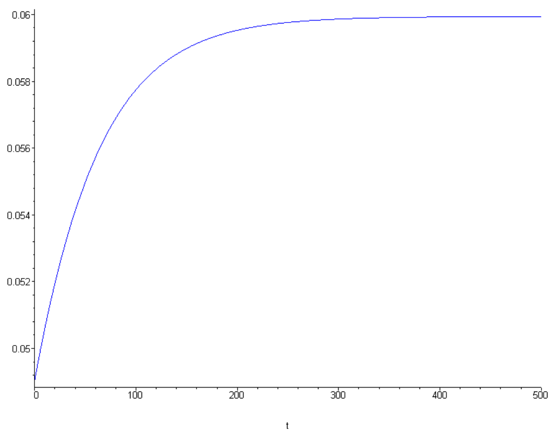


Figure 6: Evolution of the "Velocity" ρ_2 of the Skateboard in Time for the Case $n_1 > 0$, $n_2 \leq 1$.

again, that at initial instant $n_2 < 1$, i. e. $\rho_2(0) < 0$. The case $n_1 < 0$, $n_2 > 1$ is similar to the case $n_1 > 0$, $n_2 < 1$ which was considered above. These initial conditions correspond to the situation if at initial instant the skateboard takes the small velocity

$$\rho_2(0) = \Omega n_1 \frac{1-n_2}{1+n_2}$$

in “unstable” direction. In this case the limit of the system motions is the same as for $\rho_2(0) \geq 0$ but the evolution of the motion is entirely different. For

$$0 < t < t_* = \frac{E \ln(n_2)}{B\Omega n_1}$$

the absolute value of the oscillation “amplitude” ρ_1 increases monotonically and the skateboard moves in unstable direction with decreasing velocity. At the instant $t = t_*$ the velocity vanishes and the oscillation “amplitude” ρ_1 reaches its maximum absolute value

$$\rho_1(t_*) = n_1 \sqrt{\frac{A}{E}}.$$

When $t > t_*$ the skateboard already moves in stable direction with an increasing absolute value of its velocity and the oscillation amplitude decreases monotonically. Thus when $\rho_2(0) < 0$ during the time of evolution of the motion a change in the direction of motion of the skateboard occurs, see Figure 7–8. The similar nonlinear effects, like the change of the motion direction, were observed earlier in other problems of nonholonomic mechanics, for example in a classical problem of dynamics of the rattleback [7]-[9]). Thus, we describe here the basic features of the simplest skateboard model dynamics, proposed in [1, 2] and developed by us.

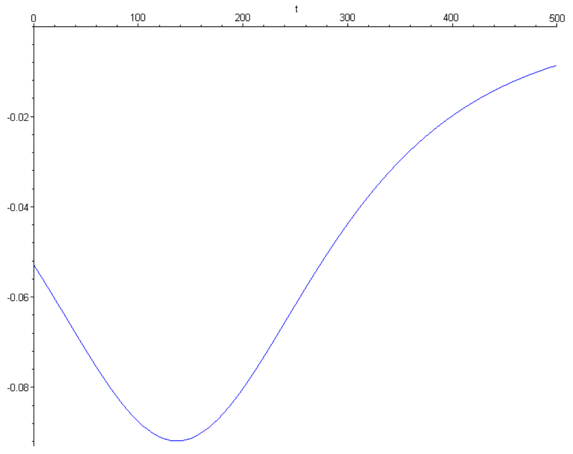


Figure 7: Evolution of the Amplitude ρ_1 of the Board Oscillations in Time for the Case $n_1 < 0$, $n_2 \leq 1$.

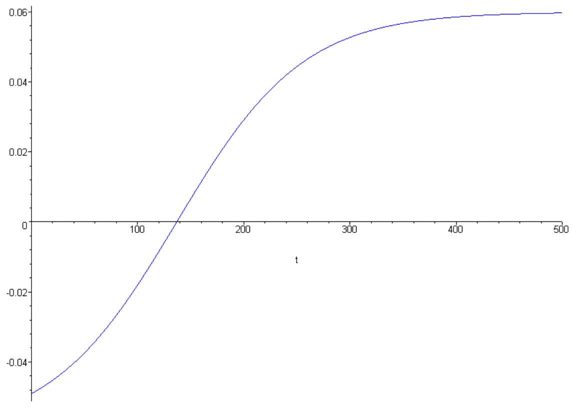


Figure 8: Evolution of the "Velocity" ρ_2 of the Skateboard in Time for the Case $n_1 < 0$, $n_2 \leq 1$.

4 Implementation and Experimental Validation

Evidently an analytic modeling and a numeric simulation may be useful to predict the dynamical properties of the sports equipment, the skateboard in our case. To verify a possibility of the behavior described above, i. e. an asymmetry property of stability depending on the rider relocation on the board, an attempt was undertaken to create the model of this device, see Figure 9.

To achieve the goal announced we used an approach and components applied earlier to the one else sports appliance: the snakeboard [11]. However, we have a serious differences with the snakeboard model now.

First, we used a spheroids of different shapes instead of ideal disks. That seems more natural and allows to consider as a wheels more plausible models of the elastic bodies rolling in future. The main current dif-

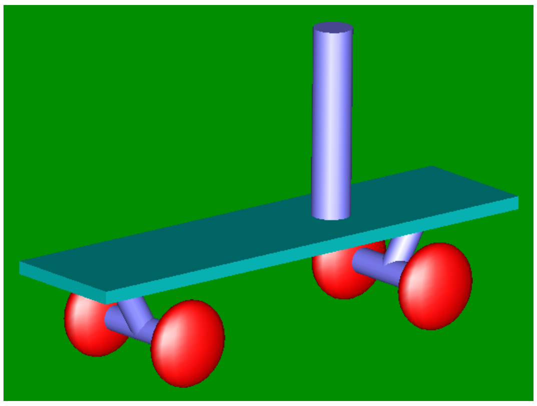


Figure 9: The Skateboard Model Animation

ference is that we applied the Hertz model and its volumetric modification for the contact of the wheel and the floor [12]. This made it possible to avoid entirely an application of the compliances artificially introduced to the snakeboard model in [11].

The wheelset model, see in Figure 10 its visual model, thus equipped by the objects of a simple revolute joint class `FixedJoint` instead of the joint model `SpringJoint` with elastic compliance along its axis. The joint connects the wheel with the shaft of the wheelset axis.

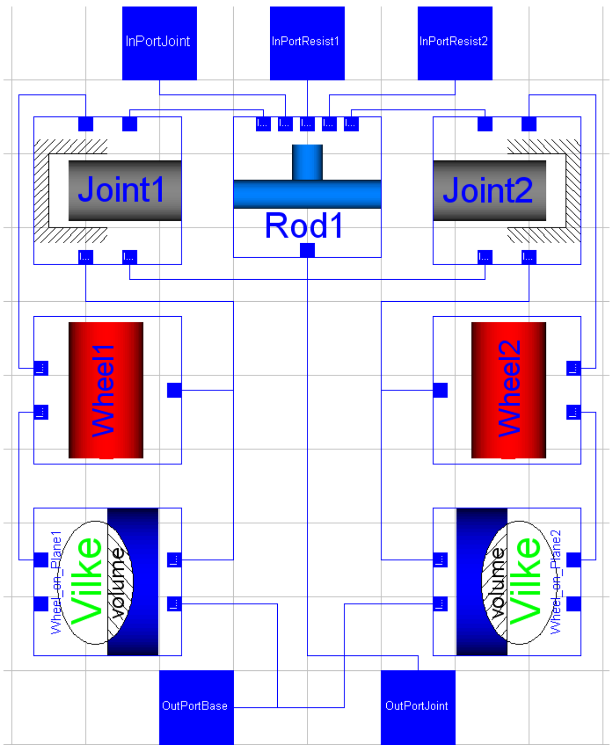


Figure 10: The Wheelset Visual Model

We saw above that in difference with the snakeboard an axis of the joint connection of the board and the wheelset is not vertical and allows the rider an effective possibility to maneuver along the road. Besides to ensure the stable riding the manufacturers frequently equip their skateboards by an elastic connections the wheelset axle and the board. Such a construct includes two springs of a high stiffness. An example of the so-called "Seismic" truck invented by D. Gesmer and M. Haug [13] see in Figure 11. The whole skateboard visual model including the spring elements is shown in Figure 12.

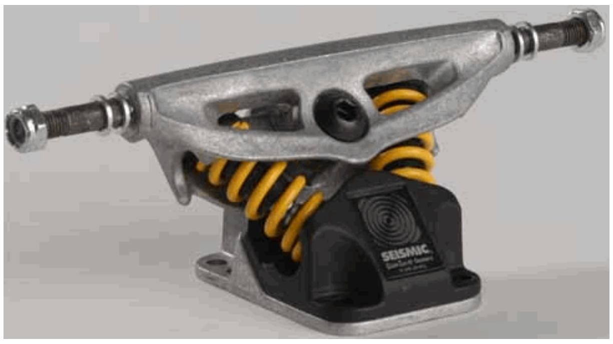


Figure 11: The Truck with Springs

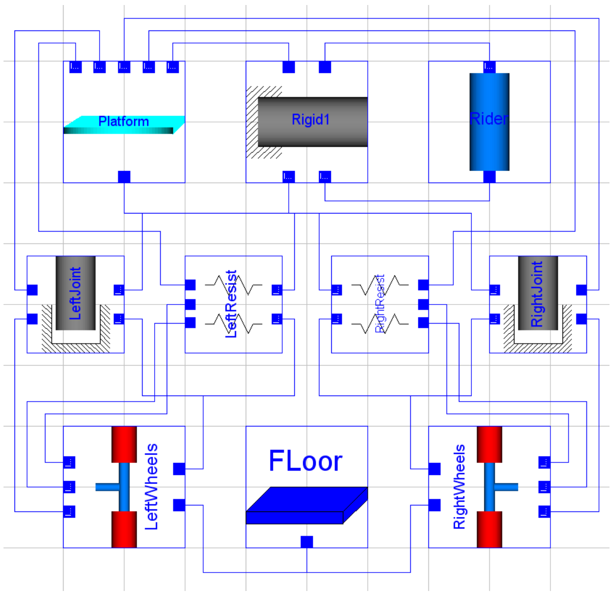


Figure 12: The Skateboard Visual Model

The visual model of the spring connection see in Figure 13. Here tne side A of a particular spring elements, `Spring1` and `Spring2`, is connected with the wheelset axle model, while the B-sides of these objects merge to one point producing one total effort. Further the model `Spring` is a usual spatial spring element resisting both the compression and the stretch. Its Modelica code has the following easy to read form:

```
model Spring
  extends Constraint;
  //undeformed spring length
  parameter SI.Length l;
  //spring stiffness
  parameter Real c;
  //fixed point on Body A
  parameter SI.Position[3] rA ;
  //fixed point on Body B
  parameter SI.Position[3] rB;
  //fixed point on Body A in abs.  syst.
  SI.Position[3] RA;
  //fixed point on Body B in abs.  syst.
  SI.Position[3] RB;
  SI.Length[3] RAB;
  SI.Length deltal;
equation
  RA = InPortA.r + InPortA.T*rA;
  RB = InPortB.r + InPortB.T*rB;
  OutPortA.P = RA;
  OutPortB.P = RB;
  RAB = RB - RA;
  deltal = sqrt((RAB -
           l*RAB/sqrt(RAB*RAB))*
           (RAB - l*RAB/sqrt(RAB*RAB)));
  OutPortB.F = -c*deltal*RAB/
           sqrt(RAB*RAB);
  OutPortB.M = zeros(3);
end Spring;
```

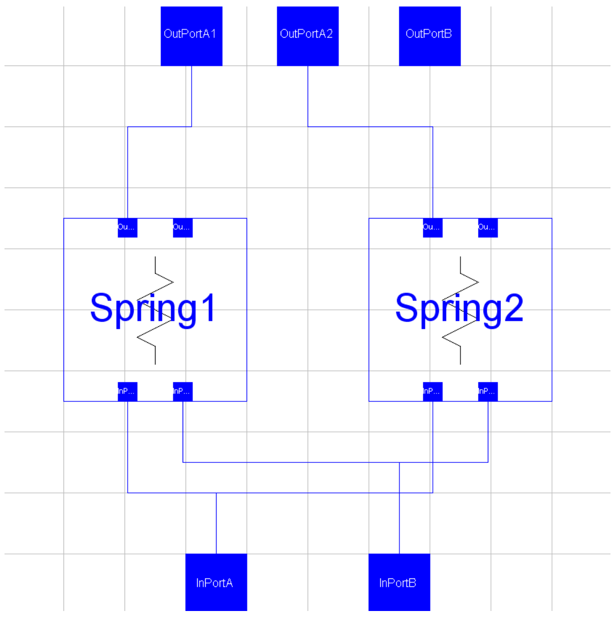


Figure 13: The Spring Connection Visual Model

Let us continue a description of the skateboard visual model in Figure 12. It is quite natural for the rider to be included into the vehicle dynamics. In our case the rider reduced simply to the cylinder standing perpendicular to the board top surface and being connected to it rigidly, by the constraint of the class `Rigid`.

A various numeric experiments performed with the skateboard model under consideration. In particular, to verify the dynamic effect of the stability of motion asymmetry, being similar to the stability asymmetry of the rattleback rotation, the cylinder playing a role of the rider motionlessly standing on the board was shifted to the right away from the board masscenter. In this case according to results outlined above if one pushes the whole skateboard to the right then the skateboard will keep this motion all the time of simulation. Otherwise, if one directs an initial skateboard velocity to the left then soon the skateboard would stop its translatory motion and then will start it to the right direction thus demonstrating instability of the left translatory motions, see the board masscenter velocity x-coordinate depending on the time in Figure 14 and the corresponding 2D-plot of the board masscenter x-coordinate itself in Figure 15.

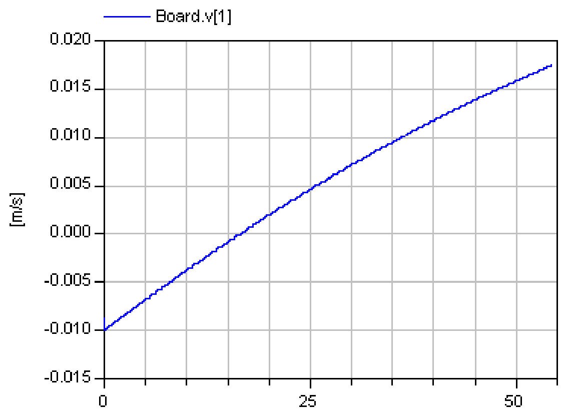


Figure 14: The Skateboard Velocity

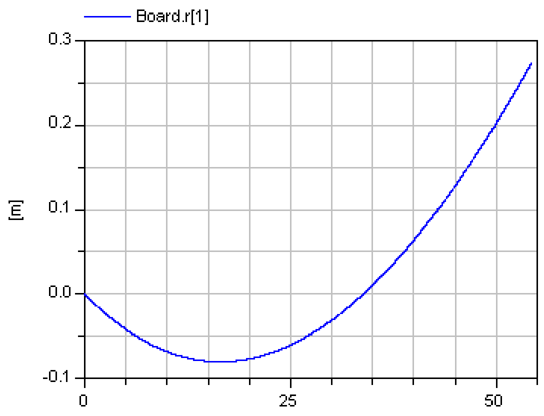


Figure 15: The Skateboard Position

Remark that the skateboard model built up turned out to be quite effective dynamic "toy" allowing to simulate the skateboard roll overs, tumbling, jumps, and bouncing over the road. It is clear to simulate a control of such motion the rider model has to be far more complicated.

5 Conclusions

An analytic analysis and numeric experimentation performed on the skateboard dynamics allow us to have simultaneously several conclusions:

- The analytic analysis results showing acceptable consistence with the numeric simulations of the models created using the physical oriented approach still remains an effective tool to investigate the skateboard dynamics.
- On the other hand the model itself can be verified reliably enough using the proper constructed analytical tools.
- Modelica turned out to be useful instrument in field of sporting and more wider in field of biomechanical applications.

And finally the nearest plans for the future work are about to investigate the complicated types of the skateboard motion including in particular the jumps.

6 Acknowledgement

The paper was prepared with partial support of Russian Foundation for Basic Research, projects 05-01-00308-a, 05-08-65470, 05-01-00454, 07-01-00290, SS-6667.2006.1.

References

[1] Hubbard M., Lateral Dynamics and Stability of the Skateboard // Journal of Applied Mechanics, 1979, Vol. 46, pp. 931–936.

[2] Hubbard M., Human Control of the Skateboard // Journal of Biomechanics, 1980, Vol. 13, pp. 745–754.

[3] Kuleshov A. S., Mathematical Model of the Skateboard // Proceedings of XXIV Int. Symp. on Biomechanics in Sports, Salzburg, Austria, 2006, Vol. 2, pp. 715–719.

[4] Kuleshov A. S., Mathematical Model of a Skateboard with One Degree of Freedom // Doklady Physics, 2007, Vol. 52, No. 5, pp. 283–286.

[5] Kremnev A. V., Kuleshov A. S., Nonlinear Dynamics and Stability of a Simplified Skateboard Model, 2007. `http://akule.pisem.net/Kuleshov2.pdf`

[6] Österling A. E. MAS 3030. On the Skateboard, Kinematics and Dynamics. School of Mathematical Sciences. University of Exeter. UK. 2004. `http://akule.pisem.net/theSkateboard.pdf`

[7] Lindberg R. E., Longman R. W. On the Dynamic Behavior of the Wobblestone // Acta Mechanica, 1983, Vol. 49, pp. 81–94.

[8] Bondi H. The Rigid Body Dynamics of Unidirectional Spin // Proceedings of the Royal Society of London, Series A, 1986, Vol. 405, pp. 265–274.

[9] Garcia A., Hubbard M. Spin Reversal of the Rattleback: Theory and Experiment // Proceedings of the Royal Society of London, Series A, 1988, Vol. 418, pp. 165–197.

[10] Bruno A. D. Local Method in Nonlinear Differential Equation — Springer–Verlag: Berlin, 1989.

[11] Kosenko I. I., Loginova M. S., Obraztsov Ya. P., Stavrovskaya M.S. Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation // Proceedings of the 5th International Modelica Conference, arsenal research, Vienna, Austria, September 4–5, 2006, pp. 213–223.

[12] Kosenko I. I., Alexandrov E. B. Implementation of the Hertz Contact Model and Its Volumetric Modification on Modelica // Submitted to Modelica'2008 Conference.

[13] Gesmer D., Haug M. Skateboard truck assembly. 1993. `http://www.freepatentsonline.com/5263725.pdf`

Design and validation of an annotation-concept for the representation of 3D-geometries in Modelica

Thomas Hoeft[1] Christoph Nytsch-Geusen[1, 2]
[1]Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstraße 7, 12489 Berlin, Germany
[2]University of Arts Berlin, Hardenbergstraße 33, 10623 Berlin, Germany
christoph.nytsch@first.fraunhofer.de

Abstract

Simulation models of complex technical systems need beside the description of their physical behaviors also a representation of their 3-dimensional geometry and topology. Up to now, the Modelica language specification [1] includes only rules for 2D-primitives in form of specialized annotations. Starting from this point, this paper illustrates the design and validation of an advanced annotation-concept for embedded 3D-geometries in physical models. The basic idea consists in the combination of specialized 3D-annotions for classes and objects with a standardized description of 3D-geometries and topologies. Therefore the X3D-standard [2] is used by the authors. Based on the founded similarities and parallelisms in the object-oriented concept of Modelica and the node concept of X3D an annotation concept for the embedding of the 3D-geometries was designed. Further an extension for the Modelica-simulator MOSILAB [3] in form of a 3D-editor plug-in was developed for the generation of X3D/Modelica-scenes and the validation of the new annotation concept. Finally the annotation-concept was evaluated in a simulation use case, where the physical model of a simplified Pool-Billiard game [4] was combined with its 3D-geometry description.

Keywords: 3D-annotation concept; X3D; 3D-representation of physical models; 4D-animation

1 Introduction

Up to now the Modelica language specification does not comprise means of expressions for code integrated description of 3D-geometries. The first fundamental analysis and conceptual work in this direction was done by [5]. Two alternative ways were discussed by the author for the integration of 3D object information in Modelica:

1. Definition of a basic set of “graphical classes”, which make a representation of primitive 3D objects (e.g. *Triangle*, *Sphere*) and position operations with this objects (e.g. *Translation*, *Rotation*) in user defined physical models possible.
2. Direct integration of the 3D object information as “graphical annotations” into the physical models self.

Further the embedding of external graphical formats like STL, VRML or DXF in Modelica models as annotations information was shortly discussed in this paper.

The Modelica-simulator Dymola [6] supports with an additional software component the visualization of 3D-objects, mainly for the MultiBody-Library. For this, external definitions of 3D-shapes via dxf-files are utilized.

In our approach for a model integrated representation of 3D objects, we have introduced *Extensible 3D Graphics* (X3D) - an open international standard for 3D on the Web and the official successor of VRML - into the Modelica language as a new annotation-type. We think this approach offers a number of important advantages:

- X3D represents a sophisticated (and international accepted) concept for complex and hierarchical structured 3D-scenes, which fits well to the object-oriented Modelica language concept.
- The prototype-concept of X3D allows an efficient integration in the object-oriented concept of Modelica.
- The annotation concept of Modelica supports the X3D integration by adding the 3D-geometrical information as X3D-strings on class level or object level. Modelica-tools, which don’t understand those X3D-annotations, are not bothered.
- The use of X3D in Modelica classes enables a simple export of the 3D representation of a physical model as a X3D-scene.

2 Annotation concept for 3D-object representation in Modelica

For the integration and validation of X3D in the Modelica language we have done following three steps:

1. **Design of an annotation concept for the representation of 3D-geometries in Modelica**
   This comprises
   - the definition of the language subset of X3D, which is necessary for the representation of 3D-objects in Modelica,
   - the definition of the annotation syntax for X3D information,
   - the Modelica class definition of a set of 3D-primitives as a base for complex 3D-scenes,
   - the rules to instantiate this 3D-classes in physical models and
   - a syntax for the coupling between the X3D object attributes and the Modelica variables for 3D-animated simulation experiments.
2. **Development of a 3D-editor for the generation and validation of Modelica models, which contain 3D-objects, described in X3D**
   This comprises
   - the definition of a set of 3D-base objects and their attributes, which shall be supported by the editor,
   - the design and the implementation of the construction interface for 3D-scenes and
   - the integration of the 3D-editor in the MOSILAB-IDE [3] as a plug-in.
3. **Evaluation of the annotation-concept with the help of a use case**
   The analyzed system model and its graphical representation
   - have to have a nontrivial recursive hierarchical structured geometry and
   - have to include static and animated subcomponents.

2.1 Graphical representation in X3D

The X3D specification uses a hierarchical node concept by the use of the XML-Syntax. A single node is described by its node type and a number of fields (node attributes). Each field has to be declared with one of the 26 X3D data types. The following simple X3D-scene, composed of a blue and a red ball, explains the main features of X3D for our use in the context with Modelica. In a first step, a reusable prototype *Ball* is defined with the *ProtoDeclare* node. The first subnode, named *ProtoInterface*, contains the field declarations, for which values can be set during the instantiation of the prototype:

```
<X3D profile="Immersive">
 <Scene>
  <ProtoDeclare name="Ball">
   <ProtoInterface>
    <field accessType="initializeOnly"
    name="radius" type="SFFloat" value="1.0"/>
    <field accessType="initializeOnly"
    name="diffuseColor" type="SFColor"
    value="0.8 0.8 0.8"/>
    <field accessType="initializeOnly"
    name="translation" type="SFVec3f"
    value="0.0 0.0 0.0"/>
    ...
   </ProtoInterface>
```

The second subnode, named *ProtoBody*, defines the functionality of the prototype. The three-dimensional geometry of the ball is described by the node for the X3D-primitive *Sphere* and its optical appearance (*diffuseColor*, *transparency …*) by the *Material*-node. The ball position and orientation is defined by the *Transform*-node with the fields *translation* and *rotation*. Further, the code snippet shows some connections between a *nodeField* of a subnode within the *ProtoBody*-node and a declared *protoField* of the *ProtoInterface*-node. This concept makes the access to these quantities possible during the instantiation of the prototype:

```
   <ProtoBody>
    <Transform>
     <IS>
      <connect nodeField="translation"
      protoField="translation"/>
      ...
     </IS>
     <Shape>
      <Sphere>
       <connect nodeField="radius"
       protoField="radius"/>
      </Sphere>
      <Appearance>
       <Material>
        <connect nodeField="diffuseColor"
        protoField="diffuseColor"/>
        ...
       </Material>
      </Appearance>
     </Shape>
    </Transform>
   </ProtoBody>
  </ProtoDeclare>
```

In the second step, the both objects *ballBlue* and *ballRed* with the prototype *Ball* are instantiated, whereas the *radius* value is set on the typical size for a billiard ball (2.65 cm) and the *diffuseColor* value is set on the RGB-values for blue and red. The red ball is displaced from the origin at 25 cm by setting the value of the *transform* field:

```
  <ProtoInstance name="Ball">
   <MetadataString name="ballBlue"/>
   <fieldValue name="diffuseColor"
   value="0.0 0.0 1.0"/>
   <fieldValue name="radius" value="0.0265"/>
  </ProtoInstance>

  <ProtoInstance name="Ball">
    <MetadataString name="ballRed"/>
    <fieldValue name="diffuseColor"
    value="1.0 0.0 0.0"/>
    <fieldValue name="radius" value="0.0265"/>
    <fieldValue name="translation"
    value="0.25 0.0 0.0"/>
   </ProtoInstance>
 </Scene>
</X3D>
```

Figure 1 shows the visualization of this short X3D-scene. As the example illustrates, X3D has not a real object-oriented concept, but the *ProtoDeclare*-node with its *ProtoInterface* and *ProtoBody* subnodes has strong parallelism to the object composition in Modelica.

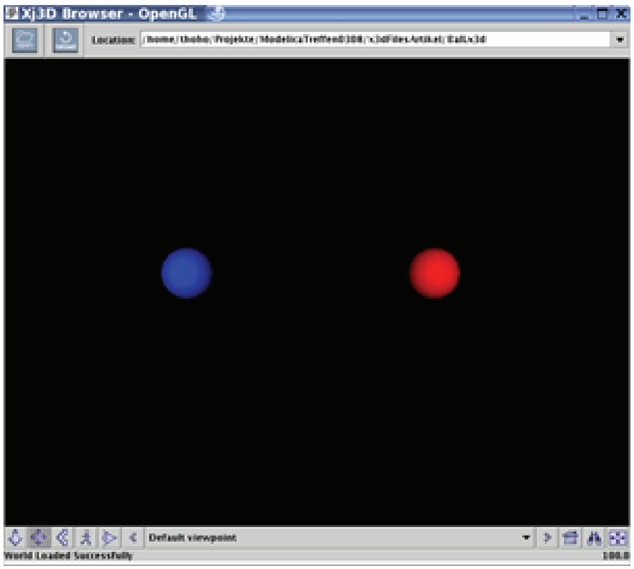


Figure 1: Simple X3D-scene with two balls

2.2 Physical behavior in Modelica

The Modelica model of the ball describes its physical behavior with simplified equations of motion of a concentrated mass. Up to now, the model has not a representation of its three-dimensional geometry:

```
import Modelica.SIunits;
...
model Ball
  parameter SIunits.Mass m = 0.2;
  parameter Real f_r = 0.05 "friction coeffient";
  SIunits.Length x, y;
  SIunits.Velocity v_x, v_y;
equation
  m * der(v_x) = - v_x * f_r; der(x) = v_x;
  m * der(v_y) = - v_y * f_r; der(y) = v_y;
end Ball;
```

2.3 Integration of X3D in Modelica

Annotations in Modelica can be used as containers for additional information, which have no influence on the modeled physical behavior of a model class. Well known examples are the definitions of graphical 2D-objects for the model icons or the model documentation in form of embedded html-Code.

In our concept we have defined a new type of annotations, which contains parts of X3D-scenes as strings and give a Modelica-model a representation of its 3D-geometry. These annotations are labeled by a new element, named *Object3D* and can be used for classes and objects:

Use in the class context:

```
model ClassName
  annotation(Object3D(x3d="X3D-String”))
  ...
end ClassName;
```

Use in the object context:

```
model ClassName
  ...
  ClassType objectname
    annotation(Object3D(x3d="X3D-String”));
  ...
end ClassName;
```

At first, based on this syntax, we have defined a set of Modelica basic types for the 3D-modeling in the package *BasicBodies*:

- Sphere3D,
- Cone3D,
- Box3D,
- Cylinder3D,
- Point3D,
- PolyLine3D.

As an example, the following code shows the implementation of the basic type Sphere 3D:

```
package BasicBodies
  model Sphere3D annotation(Object3D(x3d = "
    <ProtoDeclare name=\" Sphere3D\">
     <ProtoInterface>
      <field accessType=\" initializeOnly\"
      name=\" radius\" type=\" SFFloat\"
      value=\" 1.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" transparency\" type=\" SFFloat\"
      value=\" 0.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" diffuseColor\" type=\" SFColor\"
      value=\" 0.8 0.8 0.8\"/>
      <field accessType=\" initializeOnly\"
      name=\" translation\" type=\" SFVec3f\"
      value=\" 0.0 0.0 0.0\"/>
      <field accessType=\" initializeOnly\"
      name=\" rotation\" type=\" SFRotation\"
      value=\" 0.0 0.0 1.0 1.0\"/>
     </ProtoInterface>
```

```
      <ProtoBody>
       <Transform>
        <IS>
         <connect nodeField=\" translation\"
         protoField=\" translation\"/>
         <connect nodeField=\" rotation\"
         protoField=\" rotation\"/>
        </IS>
        <Shape>
         <Sphere>
          <connect nodeField=\" radius\"
           protoField=\" radius\"/>
         </Sphere>
         <Appearance>
          <Material>
           <connect nodeField=\" diffuseColor\"
            protoField=\" diffuseColor\"/>
           <connect nodeField=\" transparency\"
            protoField=\" transparency\"/>
          </Material>
         </Appearance>
        </Shape>
       </Transform>
      </ProtoBody>
     </ProtoDeclare>"));
  end Sphere3D;
end BasicBodies;
```

The other basic 3D-types are described in a similar manner. Starting from these basic types, the configuration of complex 3D-models in Modelica can take place.

2.4 Coupling of the physical and geometrical model description

The decisive connection between the variables of the physical model and field-values of its X3D-representation is realized by the introduction of the annotation-element *coupling*. The syntax is defined as follows:

```
model ClassName
  annotation(Object3D(x3d="X3D-String",
  coupling(protoFieldName1={v1,v2,0.0},
           protoFieldName2={v3}, ...)))
  ...
end ClassName;
```

At this, *protoFieldName* stands for the field in the 3D-representation, which shall be updated dynamically during the simulation (e.g. the object position or its size or color) and *v1, v2, v3* the corresponding Modelica variables. Thus, a physical model can have a number of coupled protoFields.

The next code piece shall illustrate this coupling concept with the help of the ball example in paragraphs 2.2 and 2.3. For this purpose, the *ProtoInterface* definition of the X3D description is integrated as an annotation on the class level, because this information concerns only the class interface. The *ProtoDeclare* node is omitted, because this information is implicit contained in the Modelica class-name itself:

```
...
import BasicBodies3D;
...
model Ball annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" radius\" type=\" SFFloat\"
   value=\" 0.0265\"/>
   ...
   <field accessType="initializeOnly"
   name=\" translation\" type=\" SFVec3f\"
   value=\" 0.0 0.0 0.0\"/>
  </ProtoInterface>"),
  coupling(translation={x,y,0.0});
```

The representation of the 3D-geometry of the class *Ball* takes place by the instantiation of the basic 3D-type *Sphere3D* as an object within the class:

```
  BasicBodies3D.Sphere3D ball
  annotation(Object3D(x3d="
    <ProtoBody>
     <ProtoInstance
      name=\" BasicBodies3D.Sphere3D\">
      <MetadataString name=\" ball\"/>
      <connect nodeField=\" radius\"
      protoField=\" radius\"/>
      ...
      <connect nodeField=\" translation\"
      protoField=\" translation\"/>
     </ProtoInstance>
    </ProtoBody>")));
  parameter SIunits.Mass m = 0.2;
  parameter Real f_r = 0.05 "friction coeffient";
  SIunits.Length x, y;
  SIunits.Velocity v_x, v_y;
equation
  m * der(v_x) = - v_x * f_r; der(x) = v_x;
  m * der(v_y) = - v_y * f_r; der(y) = v_y;
end Ball;
```

3 Use case Pool-Billard game for validating the annotation concept

In the use case, which shall validate our annotation concept for embedded 3D-geometry representations, we have used a model of a simplified Pool-Billiard game with three balls and one hole [4]. This simulation model suits well to the problem, because its geometry is hierarchical structured and includes static (table) and dynamic sub-components (billiard balls).

3.1 Modeling process

In the first step, a leg model (class *TableLeg*) from the billiard table shall be configured from the three submodels *bottom* (type *Cylinder3D*), *adapter* (type *Cone3D*) and *shaft* (type *Cylinder3D*):

```
import BasicBodies.*
...
model TableLeg annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" bottom.height\" type=\" SFFloat\"
   value=\" 0.025\"/>
   <field accessType=\" initializeOnly\"
   name=\" bottom.radius\" type=\" SFFloat\"
   value=\" 0.125\"/>
   ...
   <field accessType=\" initializeOnly\"
   name=\" translation\" type=\" SFVec3f\"
   value=\" 0.0 0.0 0.0\"/>
   <field accessType=\" initializeOnly\"
   name=\" rotation\" type=\" SFRotation\"
   value=\" 0.0 0.0 1.0 0.0\"/>
  </ProtoInterface>"));

Cylinder3D bottom annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    <MetadataString name=\" bottom\"/>
    <connect nodeField=\" radius\"
    protoField=\" bottom.radius\"/>
    <connect nodeField=\" height\"
    protoField=\" bottom.height\"/>
    ...
    <IS>
     <fieldValue name=\" translation\"
     value=\" 0.0 -0.4 0.0\"/>
    </IS>
   </ProtoInstance>"));

Cone3D adapter annotation(Object3D(x3d="
   <ProtoInstance name=\" Cone3D\">
    ...
   </ProtoInstance>"));

  Cylinder3D shaft annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    ...
   </ProtoInstance>"));
end TableLeg;
```

Figure 2 shows the visualization of the previous defined 3D-representation of the *TableLeg* model class.

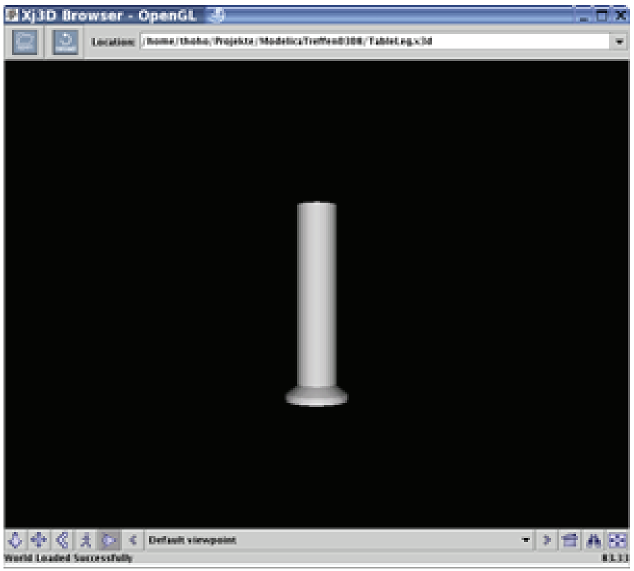


Figure 2: 3D-representation of the *TableLeg* model

On the next hierarchy level the submodels for the billiard table model are instantiated from two predefined model classes (*TableLeg*, *Border*) and from two 3D basic types classes (*Box3D*, *Cylinder3D*). The model class *BillardTable* includes the submodels for the plate, the borders, the legs and the hole. Figure 3 shows the visualization of this table model.

```
model BillardTable annotation(Object3D(x3d="
  <ProtoInterface>
   <field accessType=\" initializeOnly\"
   name=\" hole.height\" type=\" SFFloat\"
   value=\" 0.081\"/>
   <field accessType=\" initializeOnly\"
   name=\" hole.radius\" type=\" SFFloat\"
   value=\" 0.15\"/>
   ...
  </ProtoInterface>"));
  parameter SIunits.Length width,length;

  Box3D plate annotation(Object3D(x3d="
   <ProtoInstance name=\" Box3D\">
    <MetadataString name=\" plate\"/>
    <fieldValue name=\" size\"
    value=\" 2.54 0.08 1.27\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 1.0 0.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.0 0.0 0.0\"/>
   </ProtoInstance>"));

  Border borderUp annotation(Object3D(…));
  Border borderDown annotation(Object3D(…));
  Border borderLeft annotation(Object3D(…));
  Border borderRight annotation(Object3D(…));

  Cylinder3D hole annotation(Object3D(x3d="
   <ProtoInstance name=\" Cylinder3D\">
    ...
    <connect nodeField=\" height\"
    protoField=\" hole.height\"/>
    <connect nodeField=\" radius\"
    protoField=\" hole.radius\"/>
    <IS><fieldValue name=\" translation\"
    value=\" 1.26 0.0 -0.635\"/></IS>
   </ProtoInstance>"));

  TableLeg legDownLeft annotation(Object3D(x3d="
   <ProtoInstance name=\" TableLeg\">
    <MetadataString name=\" legDownLeft\"/>
    <fieldValue name=\" translation\"
    value=\" -1.06 -0.375 0.5\"/>
   </ProtoInstance>"));

  TableLeg legDownRight annotation(Object3D(…));
  TableLeg legUpLeft annotation(Object3D(…));
  TableLeg legUpRight annotation(Object3D(…));
end BillardTable;
```

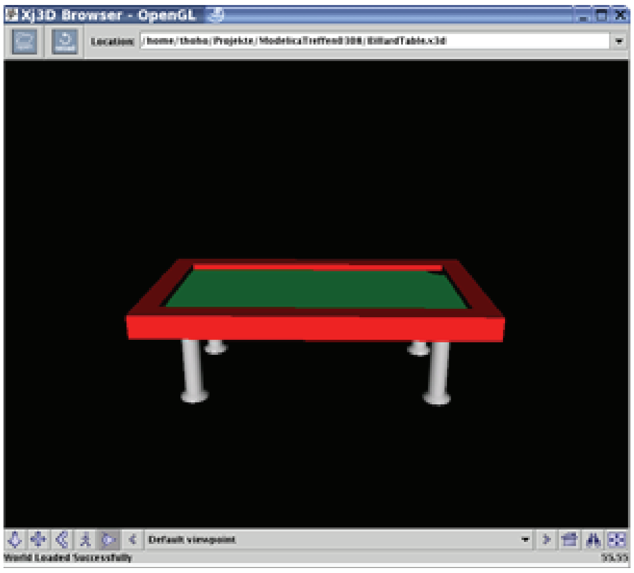


Figure 3: 3D-representation of the *BillardTable* model

The class *SystemModel* integrates the static table model and the three physical ball models. The physical model of the simplified Pool-Billiard game shall be drafted only roughly in this paper. A detailed description is given in [4]. The implementation of this example was realized with the language extension for Modelica for model structural dynamics from the GENSIM project [7, 8]. The different events of a billiard game (reflections, collisions) and a varying number of balls can be efficiently described with the concept of object-oriented statecharts and object dynamics:

```
...
model SystemModel
  annotation(Object3D(...));
  parameter Integer n_balls = 3;
  parameter Real v_x, v_y;
  parameter Real d_balls = 0.0572;
  parameter Real d_holes = 0.15;
  Point p[n_balls];

  dynamic Ball bw annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bw\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 1.0 1.0 1.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.8 0.066 -0.2\"/>
   </ProtoInstance>"));

  dynamic Ball bb annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bb\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 0.0 0.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.6 0.066 -0.2\"/>
  </ProtoInstance>"));

  dynamic Ball bc annotation(Object3D(x3d="
   <ProtoInstance name=\" Ball\">
    <MetadataString name=\" bc\"/>
    <fieldValue name=\" diffuseColor\"
    value=\" 0.0 0.0 1.0\"/>
    <fieldValue name=\" translation\"
    value=\" 0.4 0.066 -0.2\"/>
   </ProtoInstance>"));

  BillardTable t(width = 1.27, length = 2.54)
   annotation(Object3D(x3d="
   <ProtoInstance name=\" BillardTable\">
    <MetadataString name=\" t\"/>
   </ProtoInstance>"));

  event Boolean disappear_bw(start = false);
  event Boolean collision_bw_bb(start = false);
  ...
equation
  disappear_bw =
    if((p[1].x-0.0)^2+(p[1].y-0.0)^2)^0.5<d_holes
    then true else false;
  collision_bw_bb =
    if((p[2].x-p[1].x)^2+(p[2].y-p[1].y)^2)^0.5
       <d_balls then true else false;
  ...
statechart
  state SystemSC extends State;
    State startState(isInitial=true);
    State Playing, GameOver;
```

```
transition Playing->Playing
  event disappear_bw action
  disconnect(bw.p,p[1]); remove(bw);
  bw:=new Ball(d=d_balls, width=t.width,
               length = t.length,
               x(start = 1.27/2.0),
               y(start = 0.6));
  connect(bw.p,p[1]);
end transition;

transition Playing->Playing
  event collision_bw_bb action
  v_x := bw.v_x; v_y := bw.v_y;
      bw.v_x := bb.v_x; bw.v_y := bb.v_y;
      bb.v_x := v_x; bb.v_y := v_y;
    end transition;
  end SystemSC;
end SystemModel;
```

Figure 4 illustrates the complete system model with the static and dynamic model parts.

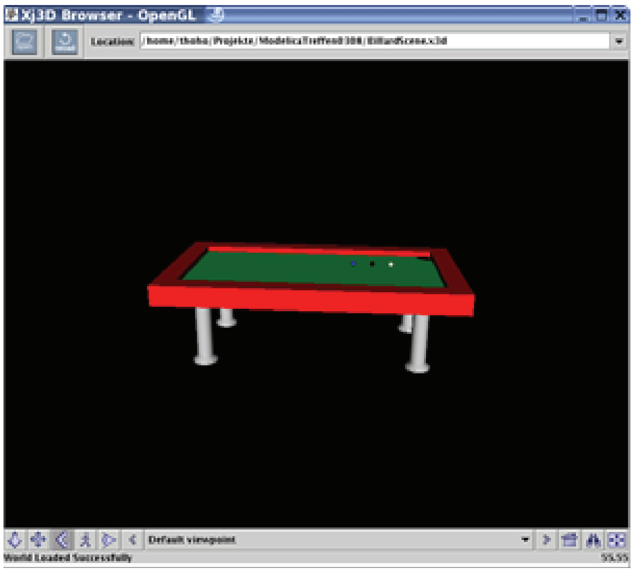


Figure 4: 3D-representation of the *SystemModel*

3.2 Simulation experiment

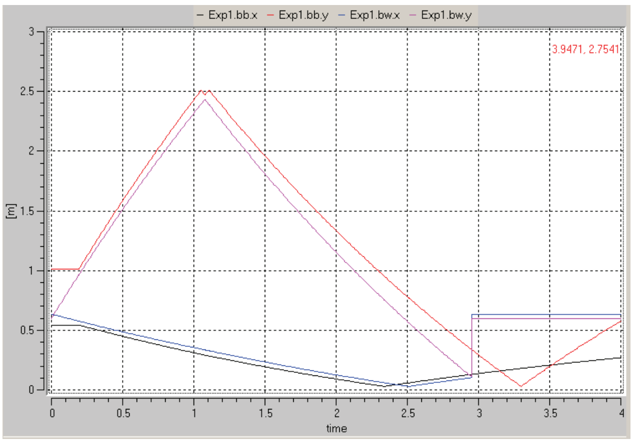


Figure 5: Simulation experiment for the Pool-Billiard game over 4 seconds.

Figure 5 shows the positions of the white and the black ball during a simulation period of 4 seconds.

After 0.2 seconds, the white ball collides with the black ball. After 1.0 second, the black ball is reflected twice in a short time period on the top side on the billiard-table and both balls collide again between its reflections. After 2.3 and 2.5 seconds the balls reflect on the left border. At 2.95 seconds the white ball drops into the hole. At the end, the white ball is set again on its starting position.

Figure 6 to Figure 9 show the 4D-animation of the same simulation experiment. Because the calculated x- and y-coordinates of both ball models are connected with their 3D-representations by the annotation-element *coupling* (compare with paragraph 2.4), a 4D-animation (3 space coordinates plus the time) of the experiment can be automatically generated.

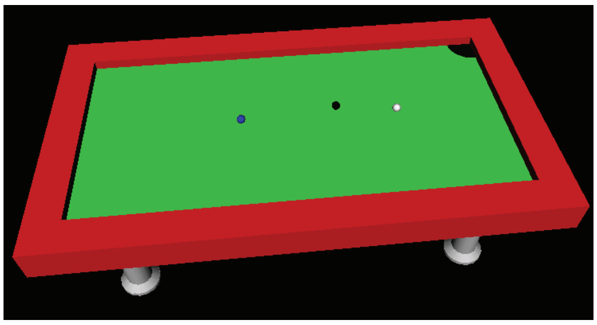


Figure 6: Simulation experiment at time=0 seconds

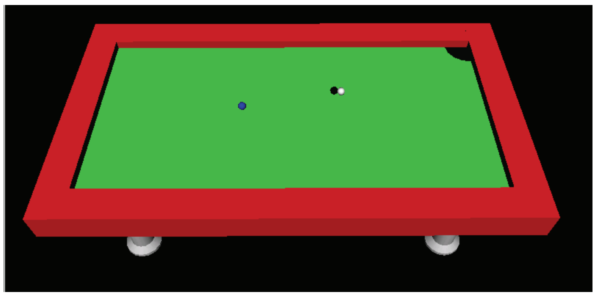


Figure 7: Simulation experiment at time=0.2 seconds

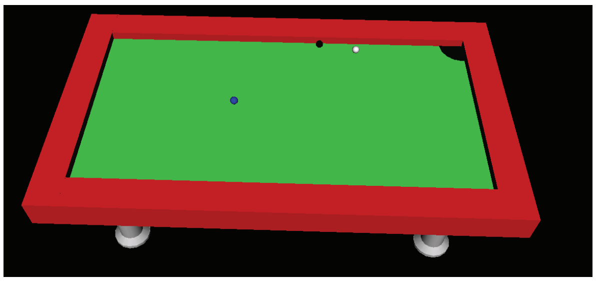


Figure 8: Simulation experiment at time=2.3 seconds

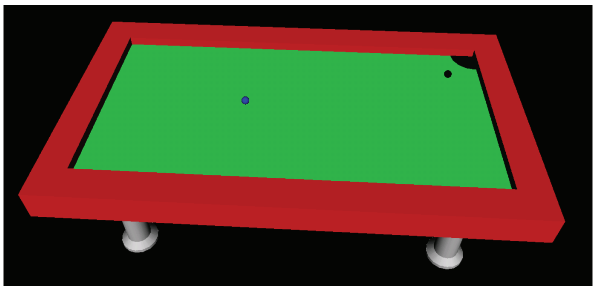


Figure 9:Simulation experiment at time=2.95 seconds

4 Development of a 3D-Model editor

For the validation of the previous described annotation concept for 3D-geometries in Modelica, a 3D-model editor is being developed by the authors. This editor supports the definition of 3D-scenes and generates the Modelica-code with the embedded X3D-description. Because the editor is implemented as a plug-in for the graphical user interface of the simulation tool MOSILAB [3], the 3D-modeling works close together with the other modeling features of the MOSILAB-IDE (compare with Figure 10).

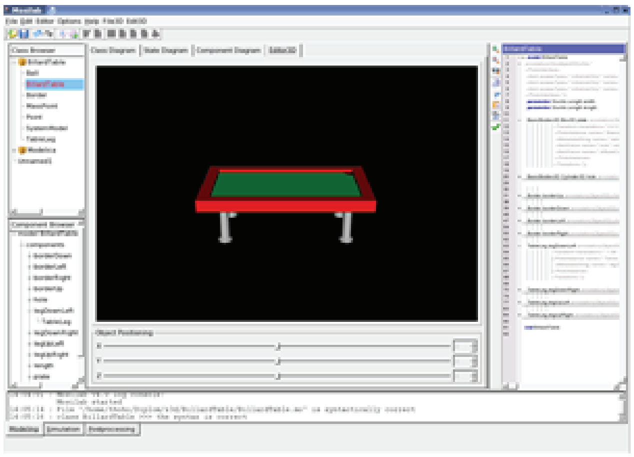


Figure 10: 3D-model editor as plug-in for the simulation tool MOSILAB .

5 Conclusions

The new designed annotation-concept for the representation of 3D-geometries in Modelica, based on the X3D-standard, offers a number of advantages and new perspectives:

- An integrated description of the equation based physical behavior and a corresponding representation of the 3D geometry in a unitary Modelica-model is an excellent precondition for an efficient communication between the physical and the geometrical model aspects.
- The use of X3D and its *ProtoDeclare*-concept fits well to the object-oriented concept of Modelica.
- 3D-objects, based on Modelica standard types, can be added directly to the physical models and could be connected by the coupling-annotation element.
- The modeling process of complex 3D-scenes can take place recursively on a multitude of hierarchical layers, because each Modelica/X3D-class can be reused on the next hierarchy-level (see

the modeling process of the billiard table in chapter 3.1).

- The use of the standard X3D-format for the modeling of the 3D-geometries within the Modelica language enables the import and export of 3D-scenes from/to X3D.
- Future works will focus on a closer integration of the 3D-editor in the simulation tool MOSILAB.

References

[1] Modelica Association Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.0, September 2007.

[2] Brutzman D. and Daly L. X3D: Extensible 3D Graphics for Web Authors. The Morgan Kaufmann Series in Interactive 3D Technology, 2007.

[3] MOSILAB-Homepage: http://www.mosilab.de

[4] Nytsch-Geusen C. The use of the UML within the modelling process of Modelica-models. EOOLT´2007, 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, Berlin 2007.

[5] Engelson V. 3D Graphics and Modelica – an integrated approach. Linköping Electronic Articles in Computer and Information Science. Linköping universitet, 2000.

[6] Dymola-Homepage: http://www.dynasim.se

[7] 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, Hamburg, 2005.

[8] Nytsch-Geusen C. et al. Advanced modeling and simulation techniques in MOSILAB: A system development case study. Proceedings of the 5th International Modelica Conference, Arsenal Research, Wien, 2006.