Real-Time Simulation of Modelica-based Models

Torsten Blochwitz Thomas Beutlich
ITI GmbH
Webergasse 1, 01067 Dresden, Germany
{blochwitz,beutlich}@iti.de

Abstract

This paper shows the various steps a simulation tool has to perform to create a real-time-capable model from a Modelica model. Reduction techniques are often necessary for complex models to meet the real-time requirements. For non-linear models with discontinuities no automatic methods of model reduction are known. The analysis methods supporting developers in identifying critical model parts are explained by means of an illustrating example model.

Keywords: real-time simulation; hardware-in-the-loop; model reduction

1 Introduction

The method of physical modeling is more and more establishing itself in the engineering departments of OEMs and component suppliers. The engineers do no longer formulate the model equations by hand but compile their models using sophisticated model libraries. Thus, detailed models are built up in comparatively short time. These models simulate the dynamic behavior of the system in detail. E.g., the vibrational behavior of drive trains or hydraulic systems is explored.

During software development of Electronic Control Units (ECU) offline (non real-time), system simulations are performed using Model-in-the-Loop (MIL) techniques. In this development stage the detailed simulation models from the system design can still be used. During the test phase of the ECU, Hardware-in-the-loop (HIL) techniques are used requiring the simulation models to run in real-time.

Costs and resources can be saved if the plant models built up during system design can be reused for real-time simulation [1, 2]. The SimulationX® [3] high-level system simulation tool supports the engineer in reusing and reducing the simulation models. The prospects and limitations of such model reuse and reduction are shown.

2 Real-Time Requirements

In the general case physical models can be represented by a DAE (differential algebraic equation) system of the form

$$0 = f(x,\dot{x},z,u,p,t) \tag{1}$$

$$y = g(x,\dot{x},z,u,p,t) \tag{2}$$

with

x	...	Continuous states variables
z	...	Discrete states variables
u	...	Inputs
y	...	Outputs
p	...	Parameters
t	...	Time.

Appropriate implicit DAE solvers can directly solve the DAE system in offline simulation.

The explicit ODE (ordinary differential equation) form

$$\dot{x} = f(x,z,u,p,t) \tag{3}$$

is numerically easier to solve than the DAE form.

Real-time capable models need to be solved within a predictable execution time per time step. The model execution time has to be less than the step size.

Implicit solvers needed for DAE calculations work by iterative methods. The execution time depends on the number of executed iterations. A common workaround is to limit the number of iterations. However, this limitation might lead to numerical inaccuracies. Additionally the Jacobian matrix needs to be updated from time to time. Hence the execution time of iterative methods is not predictable making them inapplicable for HIL simulation.

Explicit solvers do meet this requirement but can only be used for solving ODE systems. Therefore the physical model needs to be translated to the explicit ODE (3) form.

Efficient offline solvers are characterized by step size adaptations. E.g. the step size is decreased for a robust calculation of high-frequency oscillations. However, real-time capable solvers require a constant step size.

Variable step size solvers are also used to precisely detect the discontinuities and events. This cannot be guaranteed under real-time conditions; hence a robust formulation of discontinuities and events is required to prevent improper model behavior after an event.

The maximal model step size for stable calculation of a differential equation using a given solver integration algorithm depends on the natural frequencies, time constants and non-linearities. In other words if real-time is required the dynamics and non-linearities need to be limited, too.

Finally the model complexity is limited by the computing power of the target hardware. The model execution time must not exceed the available calculation time.

Summing up, the real-time requirements are

- Explicit ODE form of the system,
- Limited dynamics and non-linearities,
- Robust treatment of discontinuities,
- Limited model complexity.

3 Model Generation for HIL Simulation

The steps shown in Fig. 1 are necessary to get from a physical model to a HIL model.

In the first step the user defines the interfaces of the HIL model, i.e. the model inputs, outputs and parameters. The SimulationX Modelica compiler translates the model to explicit ODE form. The translated model is then written as C code to file.

The SimulationX Code Export Wizard guides the user step by step through the workflow. The model independent code parts (i.e. the solver code and the target specific code) are generated for selected real-time targets. For other HIL environments based on Simulink® and the MATLAB® Real-Time Workshop® these code parts are generated afterwards during the Real-Time Workshop code generation.

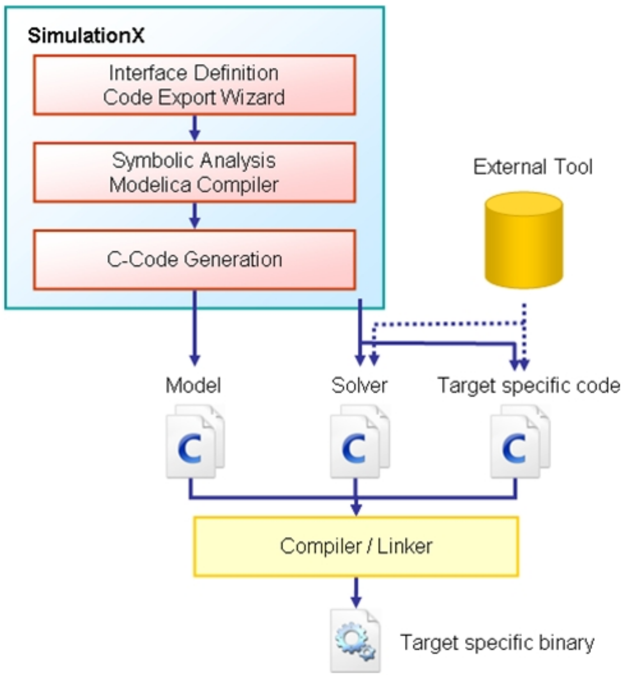


Figure 1: Workflow of HIL model generation

4 SimulationX Guidance

During the HIL model generation the simulation tool can influence the compliance with the real-time requirements. If such supporting measures are not sufficient model reduction techniques need to be taken into account. As before SimulationX supports the user in model reduction, too.

4.1 Symbolic Preprocessing

Using a modeling description language like Modelica requires symbolic preprocessing of the algorithms/equations of the entire dynamic system resulting in a simplified system of equations prepared for numerical integration.

The SimulationX Modelica compiler can either create the DAE or explicit ODE form of the system of equations. The translated model can be calculated within the simulation tool or be exported as C code (explicit ODE form only).

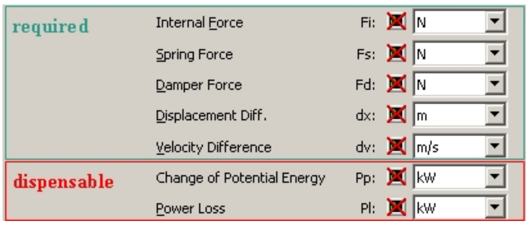
At the time of the symbolic preprocessing all model equations are known and can be optimized (even for offline simulation). The general optimization techniques involve

- Simplification of complex expressions,
- Constants are only assigned once,
- Elimination of dead branches of conditional alternatives,
- One-time calculation of repeatedly used expressions,
- Expansion of vectors and matrices,
- Loop unrolling.

For real-time simulation the optimization can even be continued. Since the user defines the necessary inputs, outputs and parameters of the model all other model parts that do not contribute to the calculation of the outputs can be cancelled. E.g. for the mechanical spring-damper in Fig. 2 the change of potential energy and the power loss are dispensable results as displayed in Fig. 3.

SpringDamper
ctr1
ctr2

Figure 2: SimulationX Spring-Damper library element



required	Internal Force	Fi:	N
	Spring Force	Fs:	N
	Damper Force	Fd:	N
	Displacement Diff.	dx:	m
	Velocity Difference	dv:	m/s
dispensable	Change of Potential Energy	Pp:	kW
	Power Loss	Pl:	kW

Figure 3: Required and dispensable results of the mechanical spring-damper

If the physical model contains implicit relationships (algebraic loops) the symbolic preprocessing tries to solve them when translating the model to explicit ODE form. Non-solvable relationships are transformed to local blocks of equations that additionally need to be solved along with the calculation of the RHS (right hand side) of the explicit ODE. Linear and non-linear systems are detected and separately solved. The non-linear implicit systems are solved by iterative methods that actually are inconsistent with the real-time requirements. However, a fast calculation is guaranteed

- by a small dimension (2...10) of the non-linear implicit blocks,
- as a symbolic Jacobian matrix is provided that results in superlinear convergence,
- as well-chosen start values for the iteration are given. (Assuming a low rate of change of the unknown variables the results from the previous time step can be used as start values for the current iteration.)

If performance problems are still an issue the user is informed of the blocks of implicit equations and the unknown variables. This information finally allows specific model changes.

Additional steps (such as index reduction and minimum dynamic state selection) might be necessary for higher index DAE systems.

4.2 Solver

In complex systems the execution time of the model mainly depends on the calculation of the model. By the introduction of a modified stability region it was shown that the well-established Euler Forward solver is the most efficient solver for complex models and most suitable for HIL applications [4]. Additionally the Euler Forward solver has the lowest numerical error on discontinuities.

Complex numerical solvers require multiple calculations of the model per time step. The stability region increases with multiple calculations of the model, i.e. the model step size can be increased as well. However, the increased model step size does not compensate the increased calculation time due multiple calculations of the model. A stabilized fixed step size solver was developed that performs better for special model classes. The distinction between the model sample rate and the integration step size allows oversampling leading to excellent results as proven by experience from numerous applications.

4.3 C Code Generation

The result of the code generation is target independent C code with defined interfaces [5].

If there are loops within auxiliary functions (e.g. characteristic curves with non-equidistant nodes, delay buffers with variable dead time) efficient search algorithms are applied. It is also ensured that no dynamic memory is allocated or freed during the model runtime.

Own implementations are provided for suboptimally implemented functions in the runtime libraries. The SimulationX Code Export Wizard can interface the following HIL environments

- Targets based on Simulink and the Real-Time Workshop are addressed by Simulink C coded S-Functions [6].
- DS1006 Processor Board [7] from dSPACE,
- SCALE-RT [8] from CosateQ,
- NI VeriStand [9] from National Instruments.

The architecture of the target specific code is different for each of the targets. In case of the dSPACE target the complete application code consisting of the model code, the solver code and the simulation engine code needs to be generated. The I/O function calls for selected dSPACE I/O boards are realized by external C function calls used within custom library elements as demonstrated in Fig. 4.

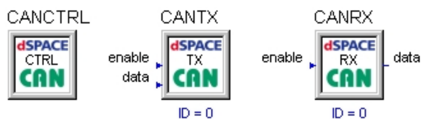


Figure 4: Custom CAN library elements for the DS1006 Processor Board target

SCALE-RT provides a simulation framework that is addressed by the SimulationX model. Using distinct custom I/O library elements the handling of the I/O function calls is similar to the dSPACE target.

The NI VeriStand target provides an extended model simulation framework. Aside from the model code only the solver code with a matching interface has to be generated. No custom I/O library elements need to be modeled as all I/O hardware access is handled outside the physical model. The NI VeriStand System Explorer accomplishes the mapping between the physical I/O channels and the model inputs and outputs after the code compilation.

5 Model Reduction

Whereas automatic model reduction techniques are neither available nor known a formal model reduction approach is described in [10]. The reduction steps closely depend on the user know-how. The following features and analysis methods of SimulationX support the user by the demanding model reduction task.

5.1 Switchable Complexity

Most complex library elements feature switchable complexity. E.g. the gear drive in Fig. 5 has to be elastically modeled for Noise - Vibration - Harshness (NVH) analyses.

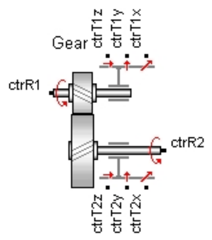
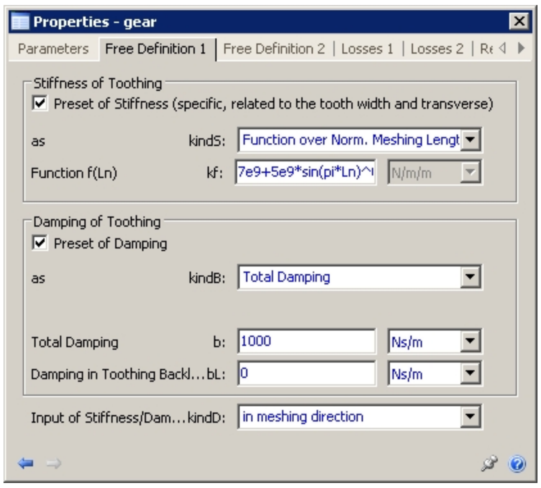


Figure 5: SimulationX Gear library element

Fig. 6 shows the complex parameterization of the stiffness and the damping of the toothing.


Properties - gear

Parameters | Free Definition 1 | Free Definition 2 | Losses 1 | Losses 2 | Re

Stiffness of Toothing

[✓] Preset of Stiffness (specific, related to the tooth width and transverse)

as kindS: Function over Norm. Meshing Lengt

Function f(Ln) kf: 7e9+5e9*sin(pi*Ln)^ N/m/m

Damping of Toothing

[✓] Preset of Damping

as kindB: Total Damping

Total Damping b: 1000 Ns/m

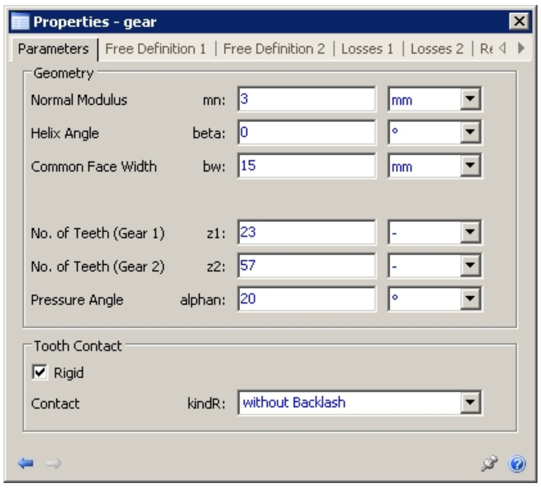
Damping in Toothing Backl...bL: 0 Ns/m

Input of Stiffness/Dam...kindD: in meshing direction

Figure 6: Elastic gear modeling with non-linear stiffness, damping and backlash

On the other hand the gear toothing is considered as rigid for HIL applications. Due to its complexity and high dynamics the vibration behavior is no longer part of the real-time simulation. If the gear parameter *rigid* is selected the gear works as ideal rigid transmission with reduced dynamics, dimension and complexity as displayed in Fig. 7.

All deactivated parameters are disabled and there values are saved.



Properties - gear

Geometry			
Normal Modulus	mn:	3	mm
Helix Angle	beta:	0	°
Common Face Width	bw:	15	mm
No. of Teeth (Gear 1)	z1:	23	-
No. of Teeth (Gear 2)	z2:	57	-
Pressure Angle	alphan:	20	°
Tooth Contact			
☑ Rigid			
Contact	kindR:	without Backlash	

Figure 7: Rigid gear modeling without backlash

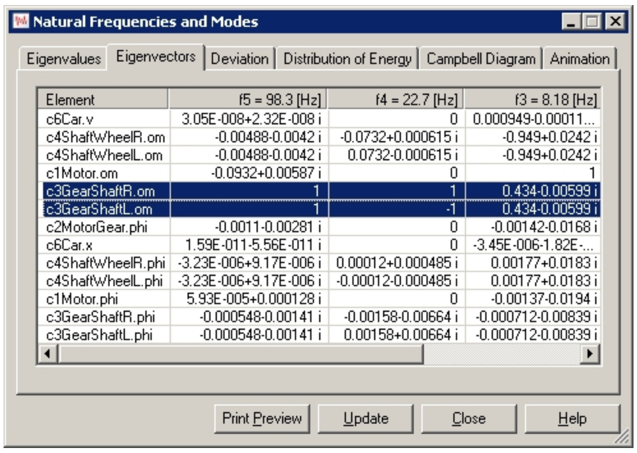
5.2 Model Analysis

Certain components need to be alternatively modeled in order to reduce the eigenvalue spectrum. E.g. mechanical elastic components need to be regarded as rigid components or hydraulic throttles must be neglected. It always is a non-trivial task to identify those model components.

A simple automotive drive train (Fig. 8) is used as illustrating example.

5.2.1 Analysis of Natural Frequencies and Mode Shapes

The analysis of the natural frequencies and mode shapes calculates the eigenvalues and eigenvectors at the current working point. The eigenvectors provide information on the influence of the state variables on the respective mode shape. Fig. 9 displays the eigenvectors corresponding to the highest three natural frequencies of a drive train. Thus the critical state variables can easily be identified.



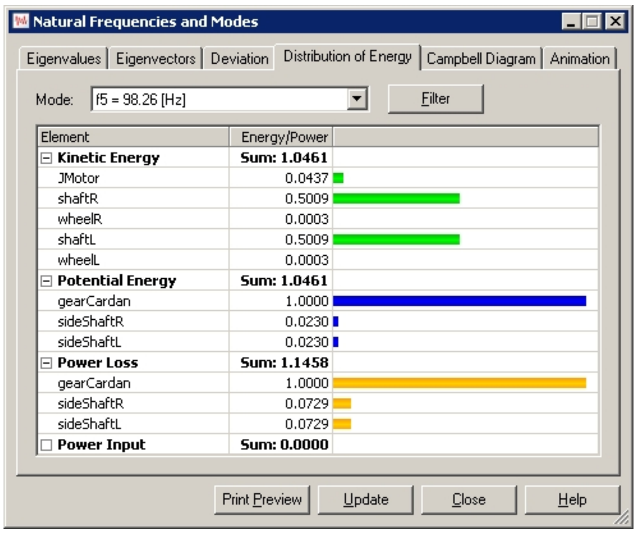
Natural Frequencies and Modes

Element	f5 = 98.3 [Hz]	f4 = 22.7 [Hz]	f3 = 8.18 [Hz]
c6Car.v	3.05E-008+2.32E-008 i	0	0.000949-0.00011...
c4ShaftWheelR.om	-0.00488-0.0042 i	-0.0732+0.000615 i	-0.949+0.0242 i
c4ShaftWheelL.om	-0.00488-0.0042 i	0.0732-0.000615 i	-0.949+0.0242 i
c1Motor.om	-0.0932+0.00587 i	0	1
c3GearShaftR.om	1	1	0.434-0.00599 i
c3GearShaftL.om	1	-1	0.434-0.00599 i
c2MotorGear.phi	-0.0011-0.00281 i	0	-0.00142-0.0168 i
c6Car.x	1.59E-011-5.56E-011 i	0	-3.45E-006-1.82E-...
c4ShaftWheelR.phi	-3.23E-006+9.17E-006 i	0.00012+0.000485 i	0.00177+0.0183 i
c4ShaftWheelL.phi	-3.23E-006+9.17E-006 i	-0.00012-0.000485 i	0.00177+0.0183 i
c1Motor.phi	5.93E-005+0.000128 i	0	-0.00137-0.0194 i
c3GearShaftR.phi	-0.000548-0.00141 i	-0.00158-0.00664 i	-0.000712-0.00839 i
c3GearShaftL.phi	-0.000548-0.00141 i	0.00158+0.00664 i	-0.000712-0.00839 i

Figure 9: Eigenvectors of a drive train

5.2.2 Distribution of Energy

Especially for mechanical systems the energy analysis as shown in Fig. 10 graphically displays the component effects on the respective mode shape.



Natural Frequencies and Modes

Mode: f5 = 98.26 [Hz]

Element	Energy/Power
Kinetic Energy	Sum: 1.0461
JMotor	0.0437
shaftR	0.5009
wheelR	0.0003
shaftL	0.5009
wheelL	0.0003
Potential Energy	Sum: 1.0461
gearCardan	1.0000
sideShaftR	0.0230
sideShaftL	0.0230
Power Loss	Sum: 1.1458
gearCardan	1.0000
sideShaftR	0.0729
sideShaftL	0.0729
Power Input	Sum: 0.0000

Figure 10: Energy distribution for a selected mode

Thus the components with the highest influence on the critical eigenvalues can be identified.

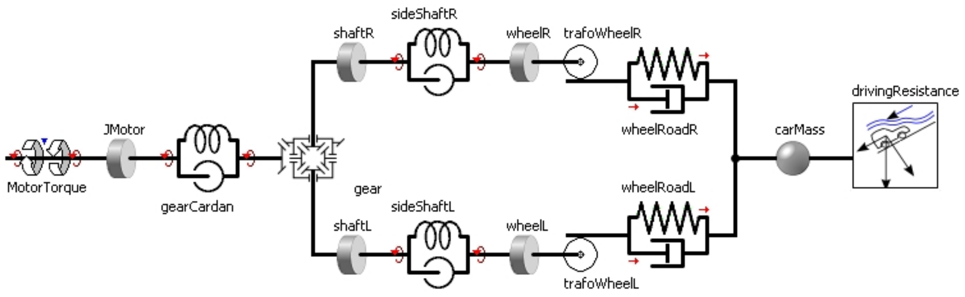
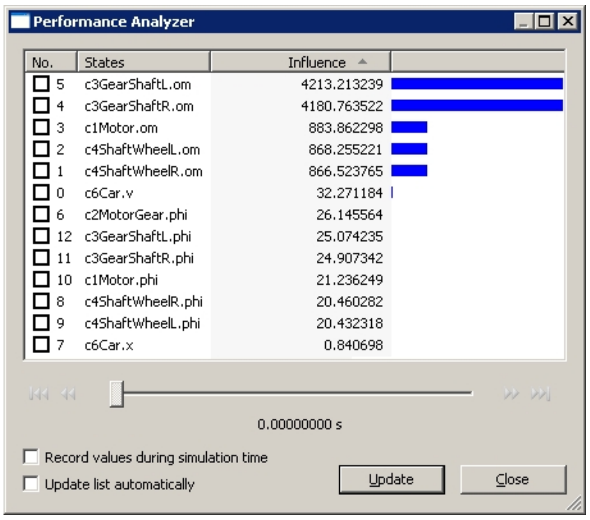


Figure 8: SimulationX model of a simple drive train

5.2.3 Performance Analysis

Both the analysis of the natural frequencies and the energy analysis operate at the current working point of the linearized system. Often a conclusion over the complete simulation period is required. The Performance analysis of Fig. 11 records an error criterion for each state variable during the offline simulation by summing up all local error estimates. Therefore it can be applied to identify critical model parts, e.g. stiff components or strong non-linearities.



No.	States	Influence
☐ 5	c3GearShaftL.om	4213.213239
☐ 4	c3GearShaftR.om	4180.763522
☐ 3	c1Motor.om	883.862298
☐ 2	c4ShaftWheelL.om	868.255221
☐ 1	c4ShaftWheelR.om	866.523765
☐ 0	c6Car.v	32.271184
☐ 6	c2MotorGear.phi	26.145564
☐ 12	c3GearShaftL.phi	25.074235
☐ 11	c3GearShaftR.phi	24.907342
☐ 10	c1Motor.phi	21.236249
☐ 8	c4ShaftWheelR.phi	20.460282
☐ 9	c4ShaftWheelL.phi	20.432318
☐ 7	c6Car.x	0.840698

0.00000000 s

☐ Record values during simulation time
☐ Update list automatically

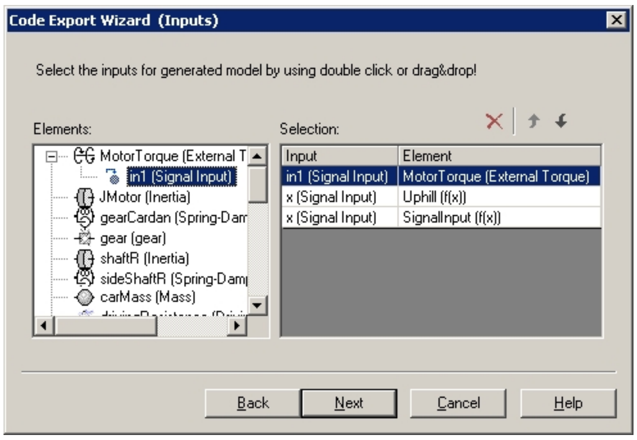
Figure 11: Performance Analyzer

6 Modelica 3.1 Language Extensions

The new Modelica language 3.1 specification [11] introduces language extensions that ease the mapping of models to execution environments. These language extensions are useful for the generation of HIL models, too.

The SimulationX Code Export Wizard is used to create a HIL model for a chosen HIL environment. As shown in section 3 the HIL target is selected here and inputs, outputs and parameters are defined (Fig. 12).

The Code Export Wizard also manages the subsequent steps (code generation, compilation and upload to the real-time target). This proceeding is very convenient if the complete model is mapped to one HIL platform. If the real-time model consists of several parts, or a model has to be split to run on several processor cores this approach becomes a little bit cumbersome. The user has to break up such models, copy each part to separate submodels and generate C code for each of them.



Select the inputs for generated model by using double click or drag&drop!

Elements:

- MotorTorque (External T
  - in1 (Signal Input)
- JMotor (Inertia)
- gearCardan (Spring-Dam
- gear (gear)
- shaftR (Inertia)
- sideShaftR (Spring-Dam
- carMass (Mass)

Selection:

Input	Element
in1 (Signal Input)	MotorTorque (External Torque)
x (Signal Input)	Uphill (f(x))
x (Signal Input)	SignalInput (f(x))

Figure 12: SimulationX Code Export Wizard with inputs page, model tree view and some inputs selected

Using the new Modelica 3.1 language extensions

- `decouple()` operator,
- `mapping` annotation,
- the task/subtask definition

a model can be separated in place and exported at once.

These new Modelica features look very promising for HIL targets that support the option to run models in several parallel tasks.

7 Conclusions

A simulation tool can already provide fundamental real-time support by

- Symbolic preprocessing,
- Efficient model code generation,
- Appropriate solvers.

For complex physical models a reduction is mostly additionally required and supported by SimulationX by

- Switchable model complexity,
- Analysis of natural frequencies and mode shapes,
- Distribution of energy,
- Performance analysis for state variables.

The consideration of the potential real-time capability of physical models already during the modeling stage results in better performance since similar models for offline and real-time simulation can shorten the model reduction steps. An appropriate model structuring (e.g. replaceable types) can also ease the model reduction and lead to higher process reliability.

ITI systematically deals with the real-time challenge. E.g. the TEMO project [12] is a joint research project by TLK Thermo GmbH, Braunschweig University of Technology, Visteon Deutschland GmbH, Daimler AG and ITI GmbH for preparation of real-time capable model components in heat conduction and thermal-fluid applications.

References

[1] Kurz, S., Wittler, G.: Hardware-in-the-Loop-Simulation: Eine Technologie im Wandel der Zeit. In: Proceedings of the 7th Haus-der-Technik-Tagung “HIL Simulation”, München, Germany, 27-28 February 2007.

[2] Blochwitz, T., Uhlig, A.: Modellgenerierung für HIL-Simulationen auf der Basis physikalischer Ansätze. In: Proceedings of the 8th Haus-der-Technik-Tagung “HIL Simulation”, Kassel, Germany, 16-17 September 2008.

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

[4] Richter, S.: Untersuchung zur Echtzeitsimulation von Modellen aus ITI SimulationX. Dresden, Germany: Master thesis, Dresden University of Technology, Faculty of Electrical Engineering and Information Technology, Institute of Automation, 2006.

[5] Blochwitz, T., Kurzbach, G., Neidhold, T.: An External Model Interface for Modelica. In: Proceedings of the 6th Modelica Conference 2008, Bielefeld, Germany, Modelica Association, 3-4 March 2008.

[6] Simulink: Writing S-Functions. The MathWorks, Inc., Natick, USA, March 2009.

[7] dSPACE DS1006 Processor Board: http://www.dspace.de

[8] SCALE-RT: http://www.scale-rt.com

[9] NI VeriStand: http://www.ni.com/veristand

[10] Rodionow, P., Grützner, S., Schreiber, U.: Erstellung, Reduktion und Validierung von Simulationsmodellen am Beispiel eines kompletten Kfz-Antriebsstranges. In: Proceedings of the 1st SimPEP Kongress, Veitshöchheim, Germany, 14-15 June 2007.

[11] Modelica Association: Modelica, A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.1, 27 May 2009.

[12] TEMO – Thermische Echtzeitfähige Modelle: http://www.pt-it.pt-dlr.de/_media/Infoblatt_TEMO.pdf