New Real-time Capabilities in Modelica for Embedded Systems

Hilding Elmqvist
Dassault Systèmes, Lund

Content

- Introduction to Modelica
- Modelica for Embedded Systems
- StateGraph - A New Formalism for Modeling of Reactive and Hybrid Systems
- FMI - Functional Mockup Interface – Overview
- Outlook

Introduction to Modelica

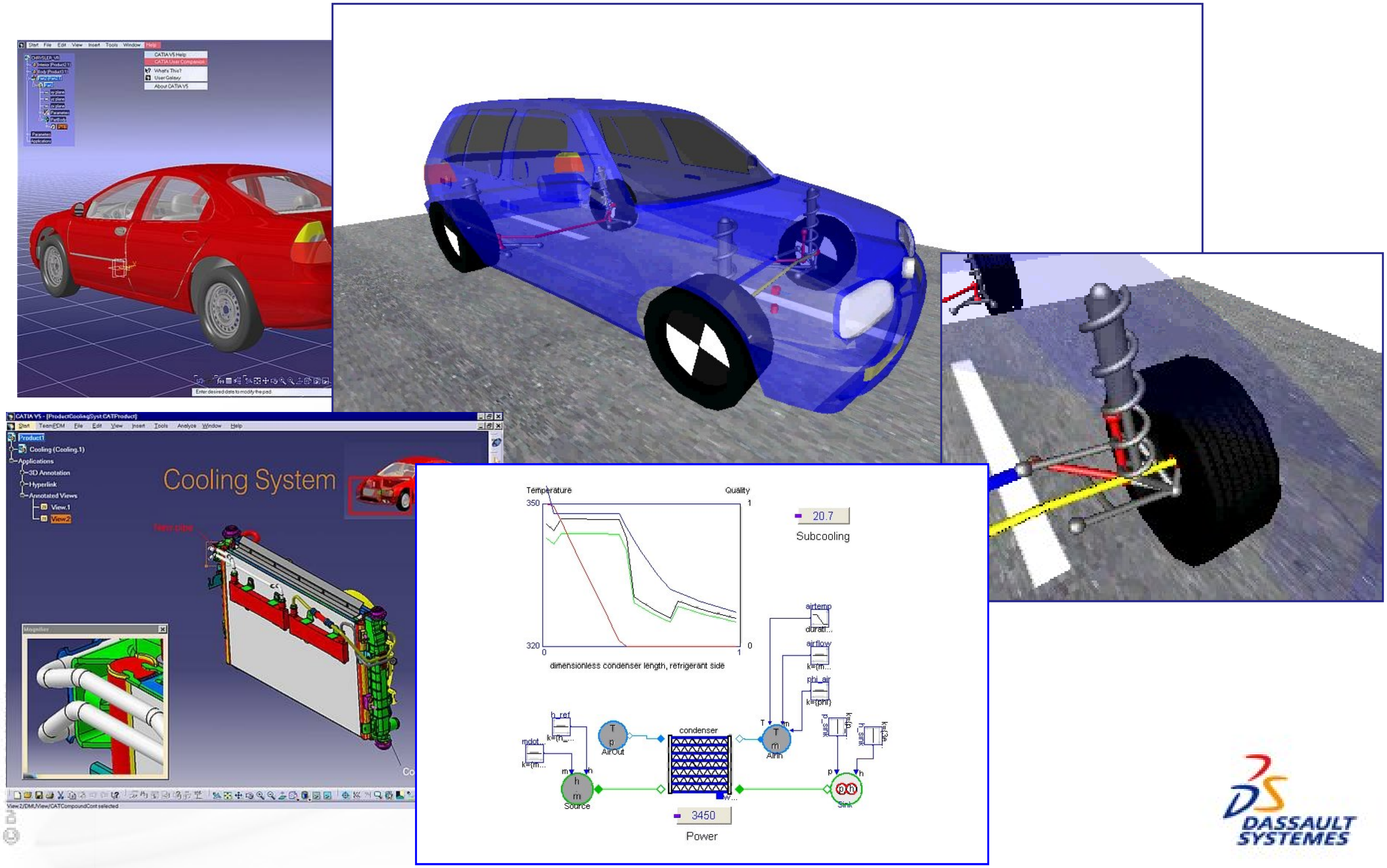
DASSAULT SYSTEMES

Content

- Introduction
- Need for Dynamic Behaviour Models
- A language – Modelica
- Organizing modeling knowhow – Modelica libraries
- A solver – Dymola
- Example

DASSAULT SYSTEMES

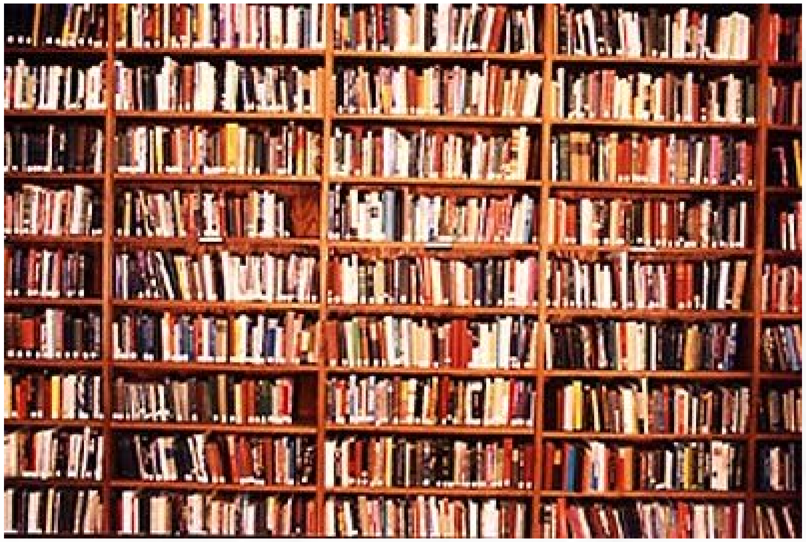
Modelica / Dymola Makes Objects Dynamic



Model Knowledge

Dynamic system model knowledge

- **has been developed over many centuries**
- **is stored in books, articles, reports and human minds**
- **which computers can not access**



A formal modeling language is needed to capture and store models for reuse

MODELICA

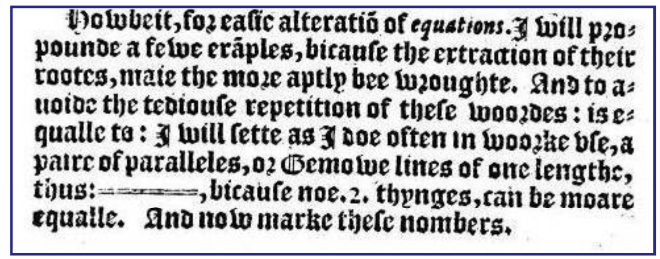
DASSAULT SYSTEMES

Dynamic system model knowledge

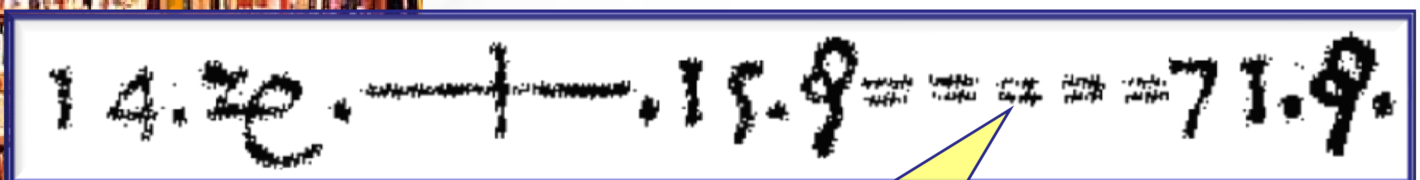
Hybrid Differential Algebraic Equations (DAE)
- **Algebraic equations**
- **Ordinary differential equations**
- **Event handling and sampled control**

$$0 = f\left(\frac{\mathrm{d}x}{\mathrm{d}t}, x, w, p, u, y\right)$$

$$0 = g\,(x, w, p, u, y)$$



Howbeit, for easie alteratiō of *equations*. I will propounde a fewe exāples, bicause the extraction of their rootes, maie the more aptly bee wroughte. And to auoide the tediouse repetition of these woordes : is equalle to : I will sette as I doe often in woorke use, a paire of paralleles, or Gemowe lines of one lengthe, thus: ═══════, bicause noe. 2. thynges, can be moare equalle. And now marke these nombers.



14.ze. —+— .15.9 ═══════ 71.9.

Equality sign introduced in 1557 by Robert Recorde

Modelica Association

- Non profit organization (www.Modelica.org)
- Defines Modelica language and standard library
- Started 1996, >50 members, >60 design meetings
- Released Modelica language specification 3.1 on May 27, 2009

MODELICA

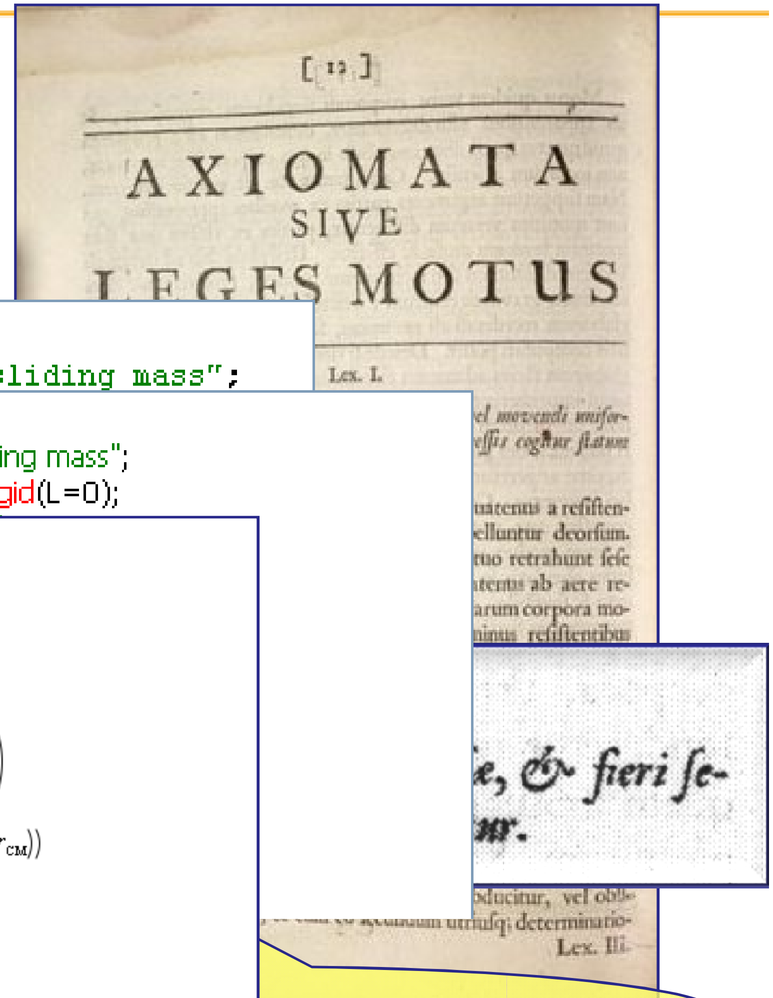
53'rd Modelica Design Meeting at Dynasim

DASSAULT SYSTEMES

Model Knowledge

Mechanics

- Newton's laws




```
model Mass "Sliding mass with inertia"
  parameter Modelica.SIunits.Mass m(min=0) "mass of the sliding mass";
  extends Mo
  Modelica.S
  Modelica.S
equation
  v = der(s)
  der(m*v) =
end Mass;
```

```
model Mass "Sliding mass with inertia"
  parameter Modelica.SIunits.Mass m(min=0) "mass of the sliding mass";
  extends Modelica.Mechanics.Translational.Interfaces.PartialRigid(L=0);
  Modelica.S
  Modelica.S
```

$$v =$$

$$\frac{d\, m \cdot v}{d\, t} =$$

```
end Mass
```

```
SI.Angle phi[3](start=phi_start, stateSelect=if enforceStates then (if
    useQuaternions then StateSelect.never else StateSelect.always) else
    StateSelect.avoid)
  "Dummy or 3 angles to rotate world frame into frame_a of body";
SI.AngularVelocity phi_d[3](stateSelect=if enforceStates then (if
    useQuaternions then StateSelect.never else StateSelect.always) else
    StateSelect.avoid) "= der(phi)";
SI.AngularAcceleration phi_dd[3] "= der(phi_d)";
```

$$r_0 = \text{frame}_a.r_0$$

$$\text{frame}_a.R = \text{Frames.from\_Q}\left(Q, \text{Frames.Quaternions.angularVelocity2}\left(Q, \frac{d\,Q}{d\,t}\right)\right)$$

$$\{0\} = \text{Frames.Quaternions.orientationConstraint}\,(Q)$$

$$g_0 = \text{world.gravityAcceleration}\left(\text{frame}_a.r_0 + \text{Frames.resolve1}\left(\text{frame}_a.R, r_{CM}\right)\right)$$

$$v_0 = \frac{d\ \text{frame}_a.r_0}{d\,t}$$

$$a_0 = \frac{d\,v_0}{d\,t}$$

$$w_a = \text{Frames.angularVelocity2}\,(\text{frame}_a.R)$$

$$z_a = \frac{d\,w_a}{d\,t}$$

$$\text{frame}_a.f = m \cdot \left(\text{Frames.resolve2}\left(\text{frame}_a.R, a_0 - g_0\right) + z_a, r_{CM} + w_a, w_a, r_{CM}\right)$$

$$\text{frame}_a.t = I \cdot z_a + w_a, I \cdot w_a + r_{CM}, \text{frame}_a.f$$

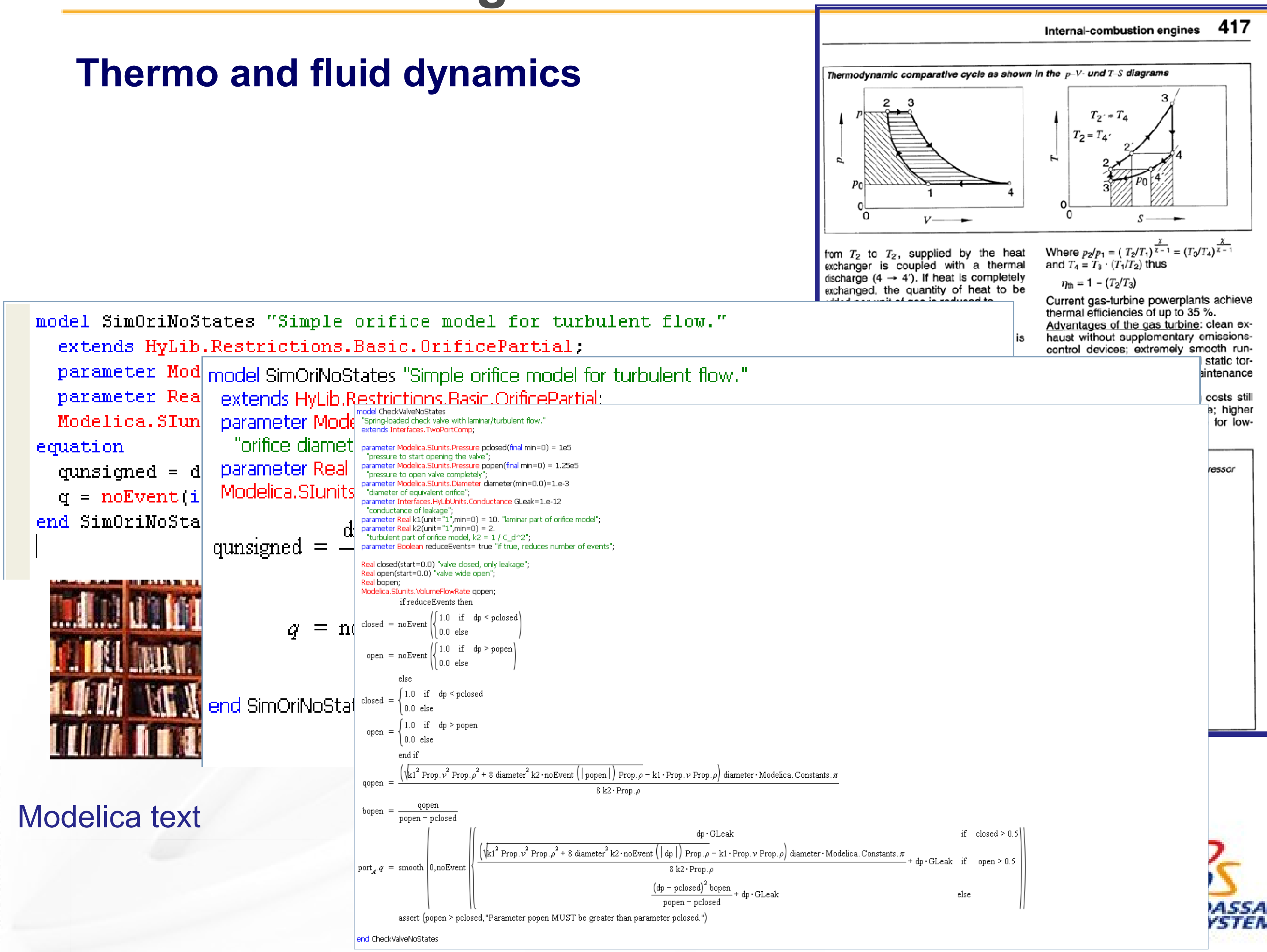
```
end Body
```

aw (July 5, 1687)

Modelica text

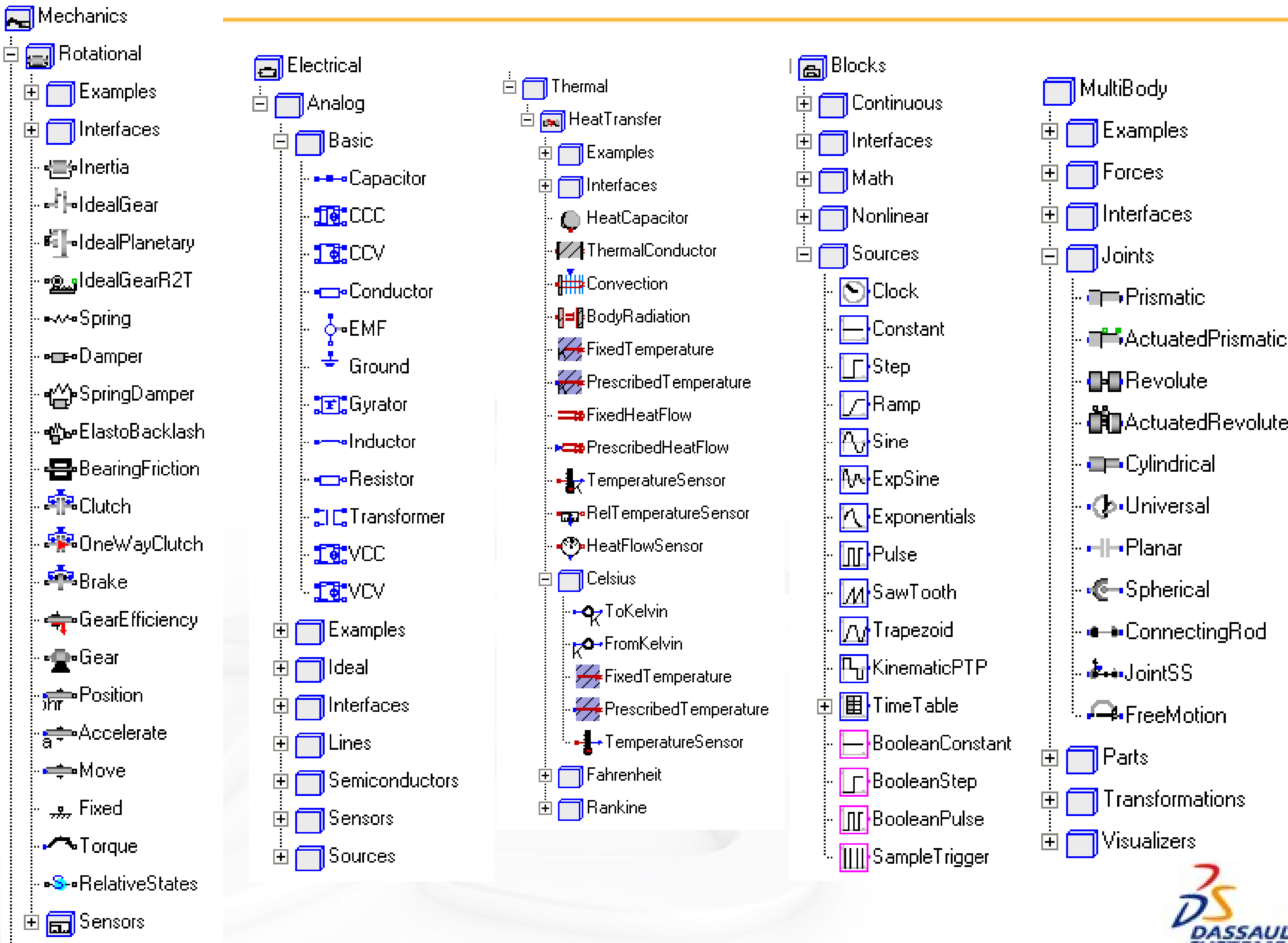
Model Knowledge

Thermo and fluid dynamics



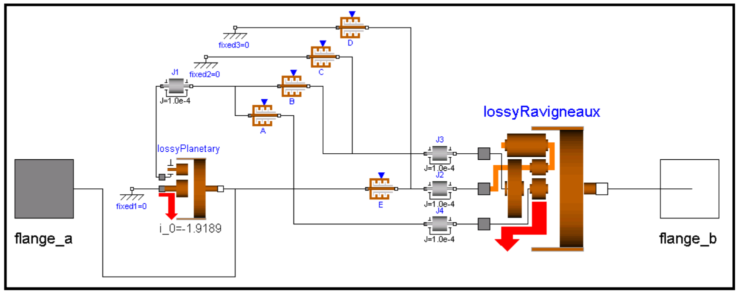
Modelica text

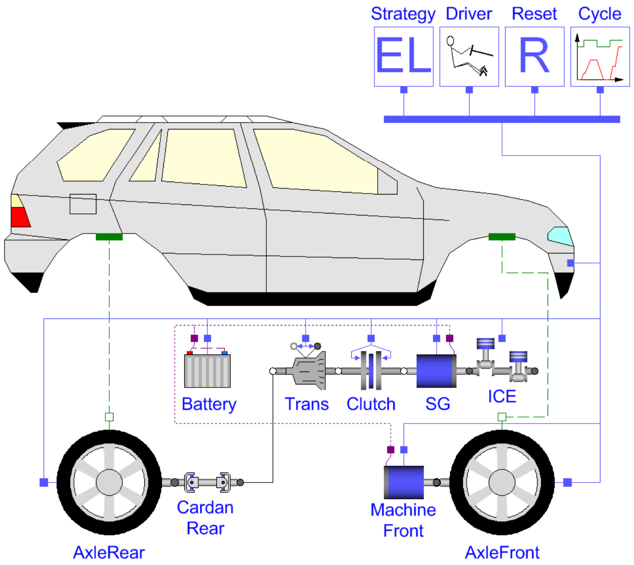
Part of Modelica Standard Library

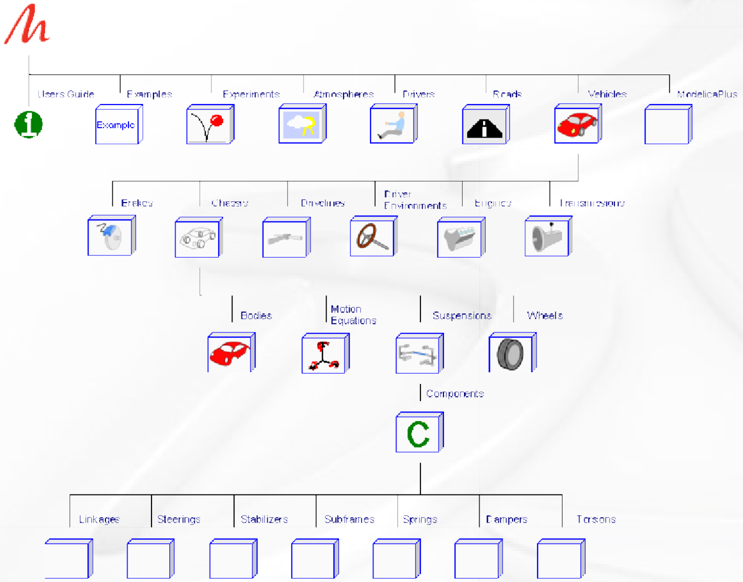


Examples of Commercial libraries

- PowerTrain
- Electric drives
- Vehicle Dynamics







EUROSYSLIB in one slide

- Initiated by DS & DLR
- 2.75 Years Duration
  Oct. 2007 – June 2010
- 19 Partners
- 101.5 Person Year effort
- 16 Mill. € total budget
- 8 Sub-Projects
- 32 Work Packages
- 27 Modelica Libraries
  to be developed
  (free + commercial)

Companies

Dassault Systemes, Dynasim, EDF, LMS Engineering Innovation, Dassault Aviation, XRG Simulation GmbH, ABB, Claytex, Siemens, Kämmerer AG (Engineering · Consulting · Solutions), Schneider Electric, Modelon, ZF Driveline and Chassis Technology

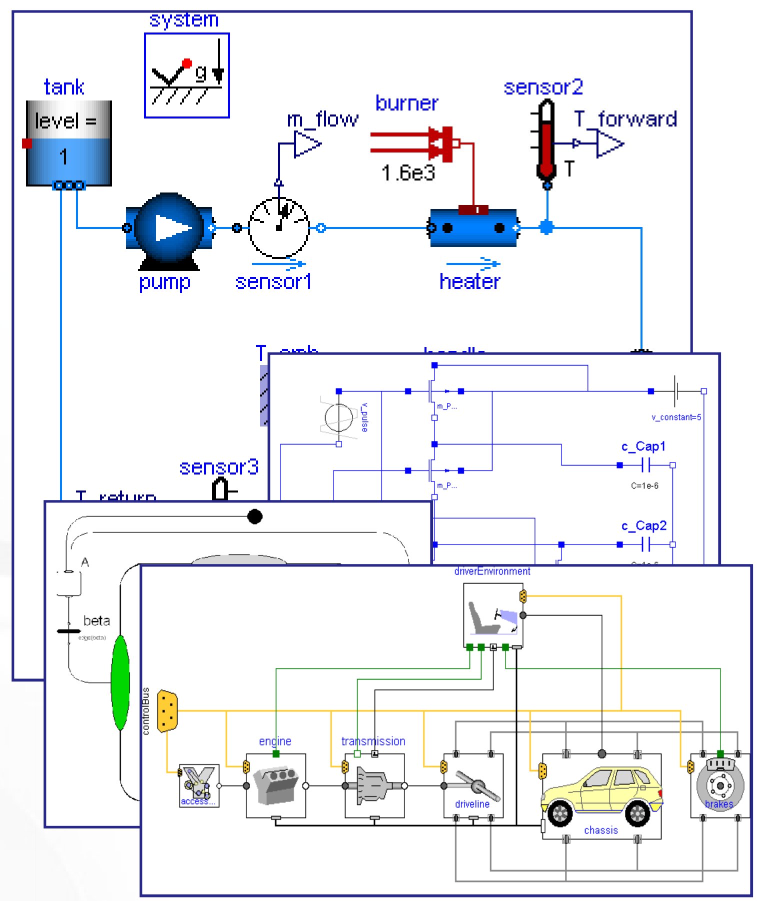
Research Institutes & Universities

DLR, Fraunhofer Gesellschaft (FhG), INRIA, AIT Austrian Institute of Technology, Lund University, IFP Innovation Energy Environment, Dassault Systemes

Outcomes - Free Modelica Libraries

It is planned that the following 11 EUROSYSLIB libraries will be provided without cost and as open source software:

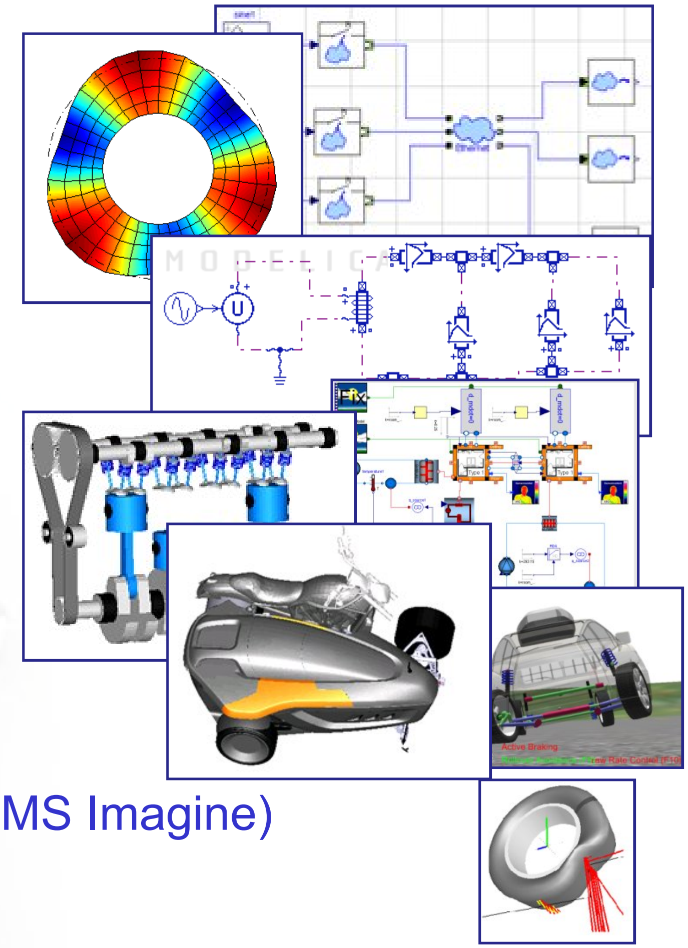
- Modelica_Fluid (ABB, DLR, Dynasim…)
- Electric.Analog lib extension (Fraunhofer)
- SPICE library (Fraunhofer)
- FluidDissipation (XRG)
- Two PowerPlant libs (EDF, ABB/Siemens)
- Modelica_LinearSystems2 (DLR-RM)
- Modelica_Controller (DLR-RM)
- StateGraph2 (Dynasim, DLR-RM)
- EmbeddedSystems (Dynasim, DLR-RM)
- VehicleInterfaces (DLR-RM)



Outcomes - Commercial Modelica Libs

It is planned that the following 16 EUROSYSLIB libraries will be commercial :

- DesignOptimization (DLR-RM)
- MultiField library (DLR-RM)
- SmartElectricDrives library (AIT)
- ElectroMechanical library (LMS)
- ThermoHydraulics library (LMS)
- HumanComfort library (XRG)
- ControlDesign library (DLR-RM)
- TrueTime library (Lund University)
- Engine Libraries (Claytex, IFP)
- Tyre Library (Kämmerer)
- Mechatronic Opening library (Kämmerer)
- Heat Exchanger Stack and Under-hood library (LMS Imagine)
- VehicleControl library (DLR-RM)



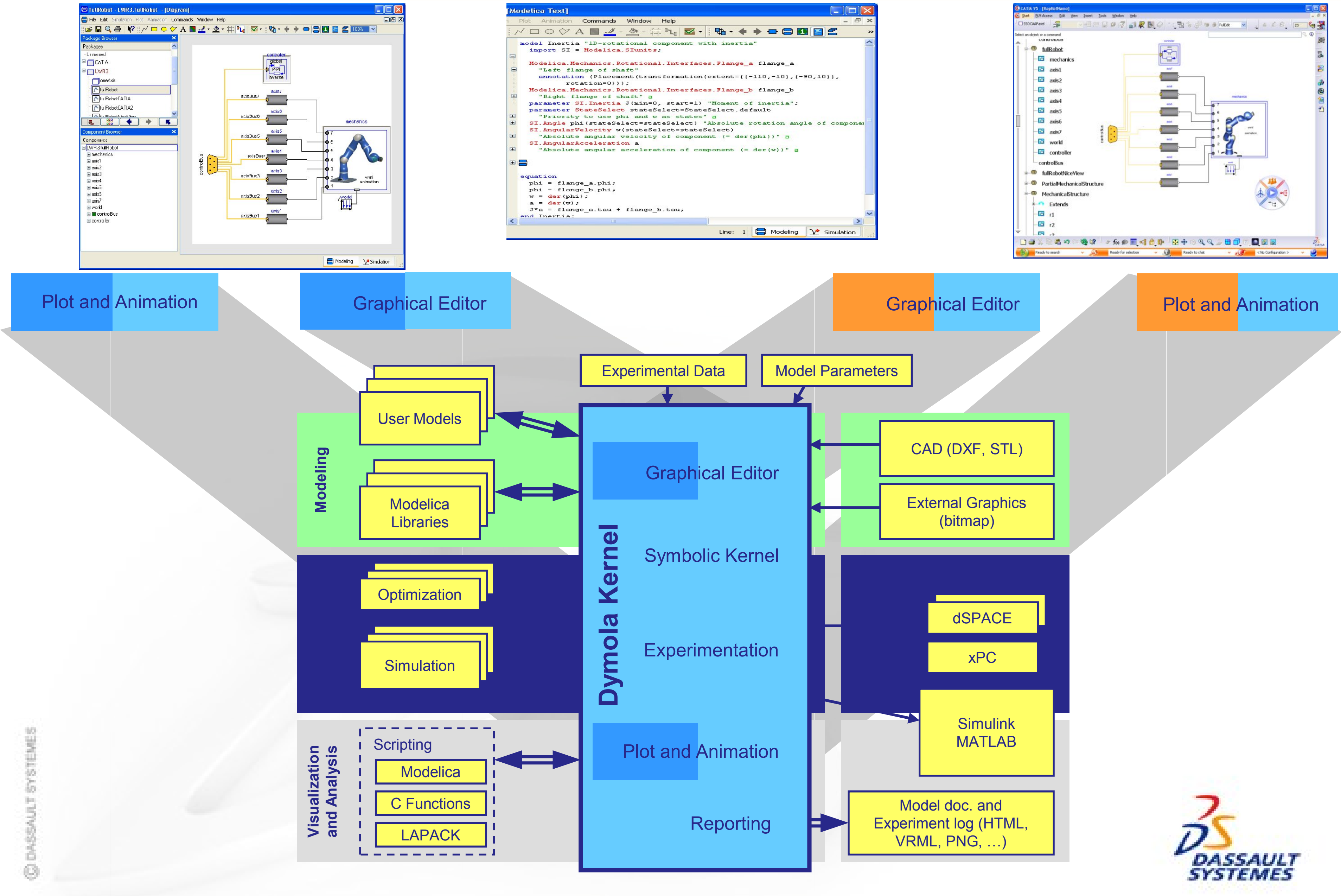
EUROSYSLIB in the Modelica Conference 2009

- *Tutorial 3: Simulation of Electrical Machines and Drives Using the Machines and the SmartElectricDrives Lib*
- *Stream Connectors- Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena*
- *Standardization of Thermo-Fluid Modeling in Modelica.Fluid*
- *FluidDissipation for Applications a Library for Modelling of Heat Transfer and Pressure Loss in Energy Syst*
- *Preliminary Design of Electromechanical Actuators with Modelica*
- *Operator Overloading in Modelica 3.1*
- *Advanced Simulation of Modelica Models within LMS Imagine.Lab AMESim Environment*
- *HumanComfort Modelica-Library Thermal Comfort in Buildings and Mobile Applications*
- *Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica*
- *Investigating the Multibody Dynamics of the Complete Powertrain System*
- *Modelica for Embedded Systems*
- *A New Formalism for Modeling of Reactive and Hybrid Systems*
- *Improvement of MSL Electrical Analog Library*
- *News in Dymola*
- *SPICE3 Modelica Library*
- *Modelica Libraries for Linear Control Systems*
- *Linear Analysis Approach for Modelica Models*
- *TrueTime Network - a Network Simulation Library for Modelica*
- *Simulation of the Dynamic Behavior of Steam Turbines with Modelica*
- *The AdvancedMachines Library: Loss Models for Electric Machines*
- …

Modelica environments: Dymola and CATIA Systems

Dymola

CATIA Systems



Virtual testing of system behavior

Play

Modelica for Embedded Systems

H. Elmqvist[1], M. Otter[2], D. Henriksson[1], B. Thiele[2], S.E. Mattsson[1]

[1]Dassault Systèmes (Dynasim), Lund, Sweden
[2]DLR - Institut for Robotics and Mechatronics, Oberpfaffenhofen, Germany

Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft DLR

DASSAULT SYSTEMES

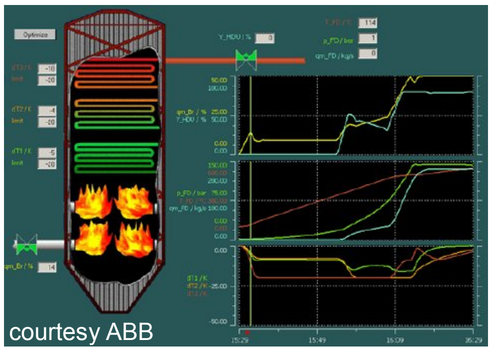
ITEA 2 INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

eurosyslib

Outline

1. Overview

- Modelica is used for advanced controller applications since 2000 (using the non-linear plant model in the embedded system). Examples:



Boiler startup optimization by ABB
(non-linear model predictive control)

Autoland controller by DLR
(non-linear dynamic inversion)

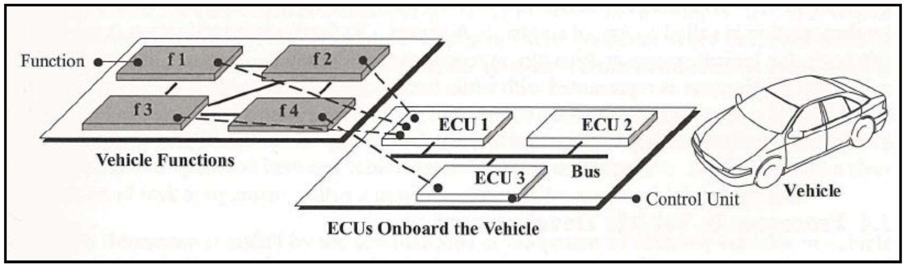
Robot vibration control by DLR
(non-linear inverse model)

But:
Currently only for specialist, with manual coding for production code

Goal:
Complete tool chain for model based controllers,
especially with non-linear Modelica models in the controller:

non-linear plant model → controller design → target deployment
(including cheap microprocessors without floating point unit)

➝ <u>Complex control</u> systems in cars

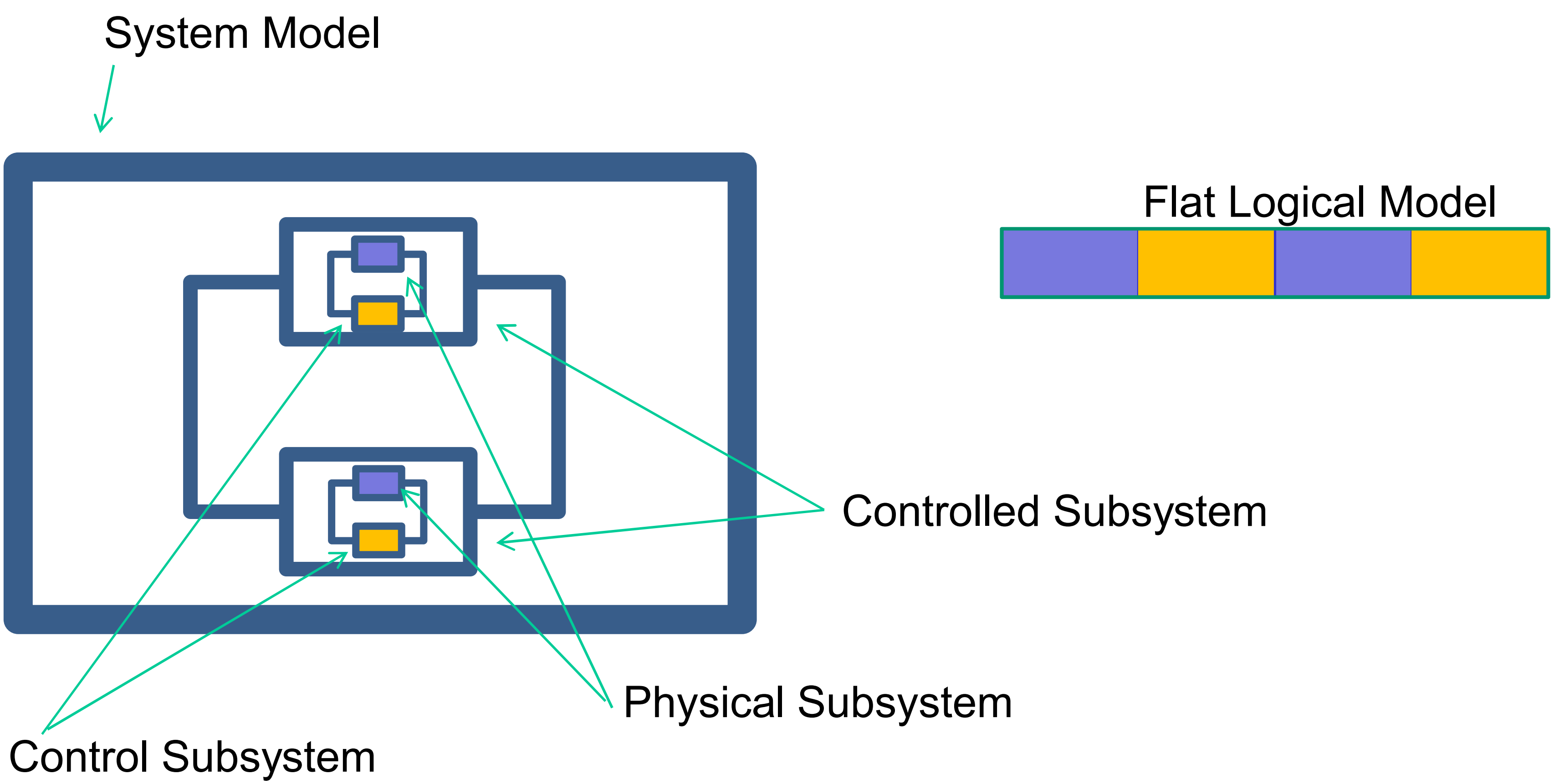


Schäuffele and Zurawka; *Automotive Software Engineering*; SAE International, 2005

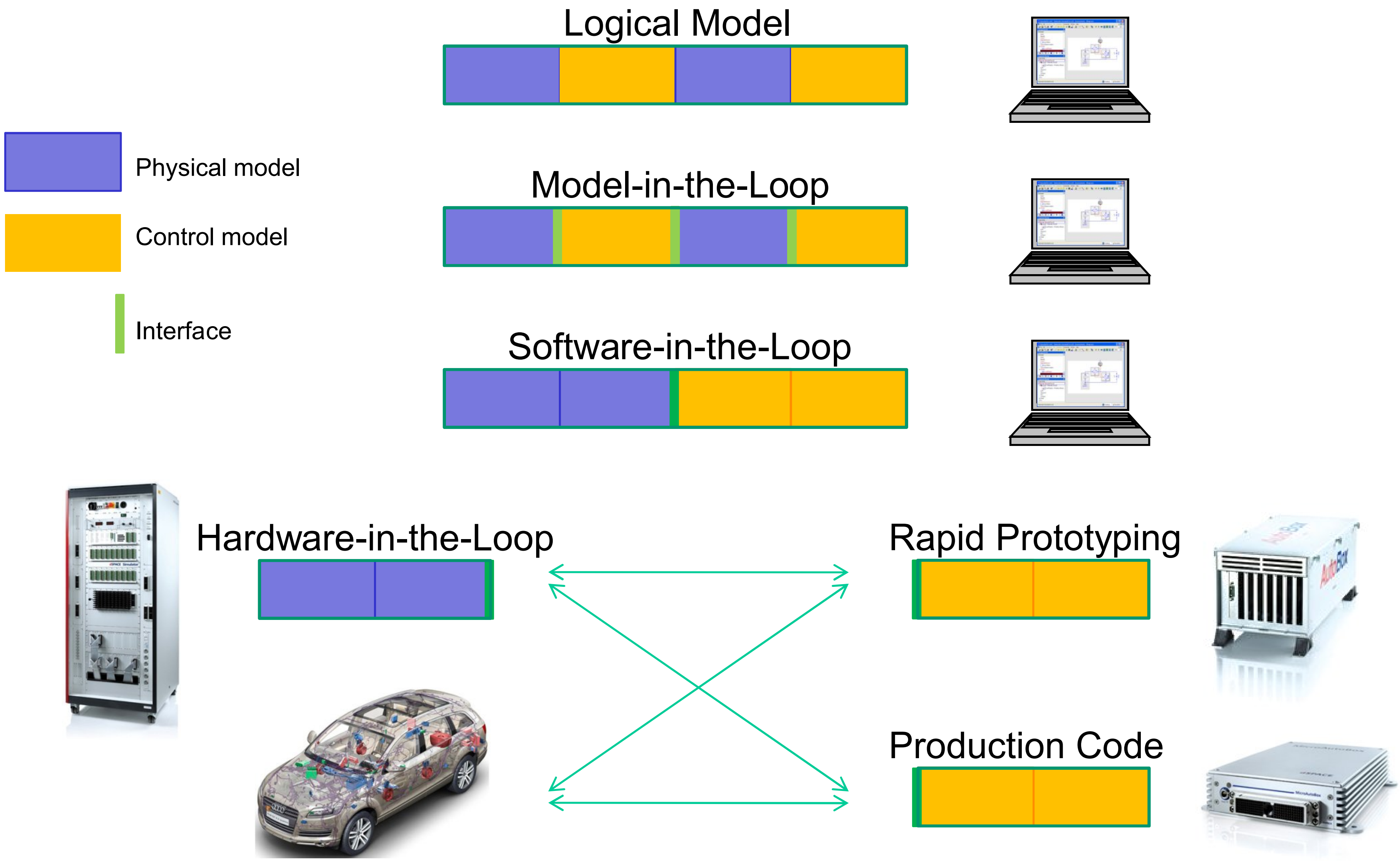
→ Separation of logical design (vehicle functions) and mapping to physical architecture (ECUs)

<u>But:</u>
The logical model needs to be partitioned for different applications

Controlled Systems



X-in-the-loop Simulation

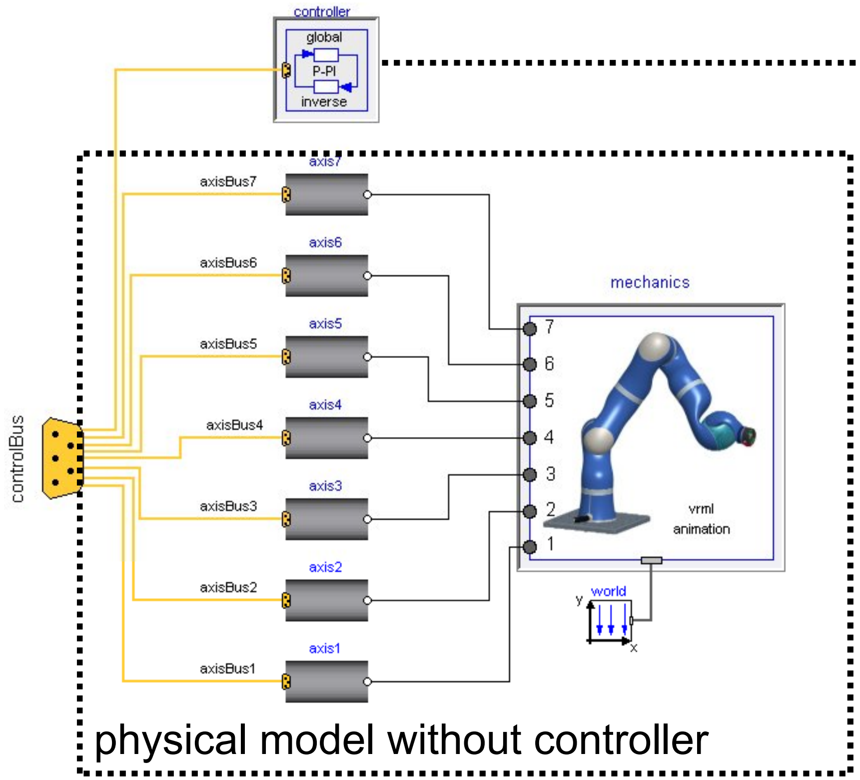


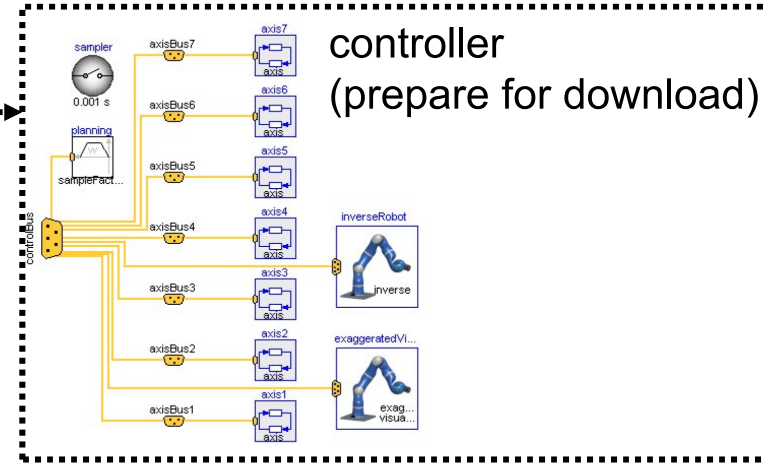
Partitioning of Logical Model

- Simulation of logical model
  - variable step size integrators
  - Controller: ideal [continuous], synchronous controllers
- Model-In-the-Loop (MIL) simulation
  - Controller – Plant interface modeled (sampled, delays, noise, etc)
- Software-In-the-Loop (SIL) simulation
  - Controller – Plant decomposition, Task – subtask decomposition, [fixed point representation]
- Hardware-In-the-Loop (HIL) simulation (real-time)
  - Plant: fixed step size integrators, multi-rate, I/O coupling
- Rapid prototyping (real-time)
  - Controller: multi-tasking, I/O channel assignment, bus communication
- Production code (real-time)
  - Controller: embedded in ECUs, multi-tasking, [fixed-point representation,] I/O channel assignment, bus communication

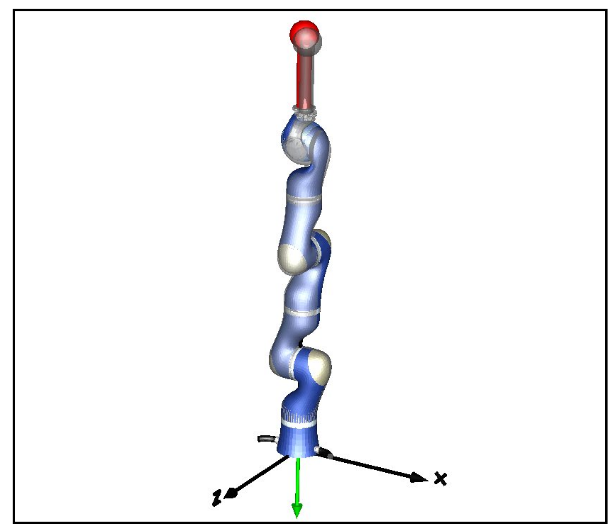
Goal: One logical model, many different mappings
without changing the logical model

Example: Todays Approach





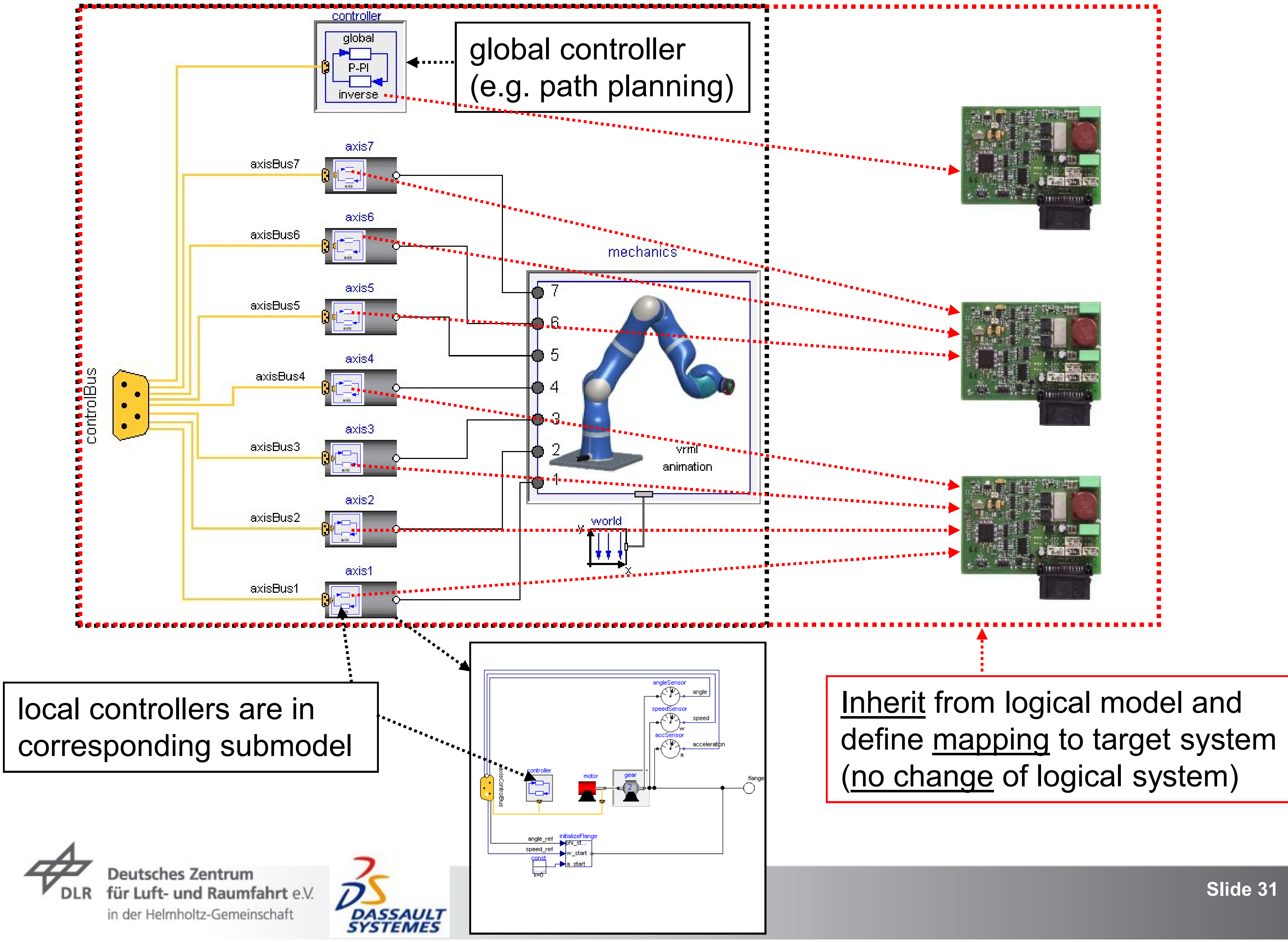
grey : reference motion
color: exaggerated
actual motion
(factor = 50)

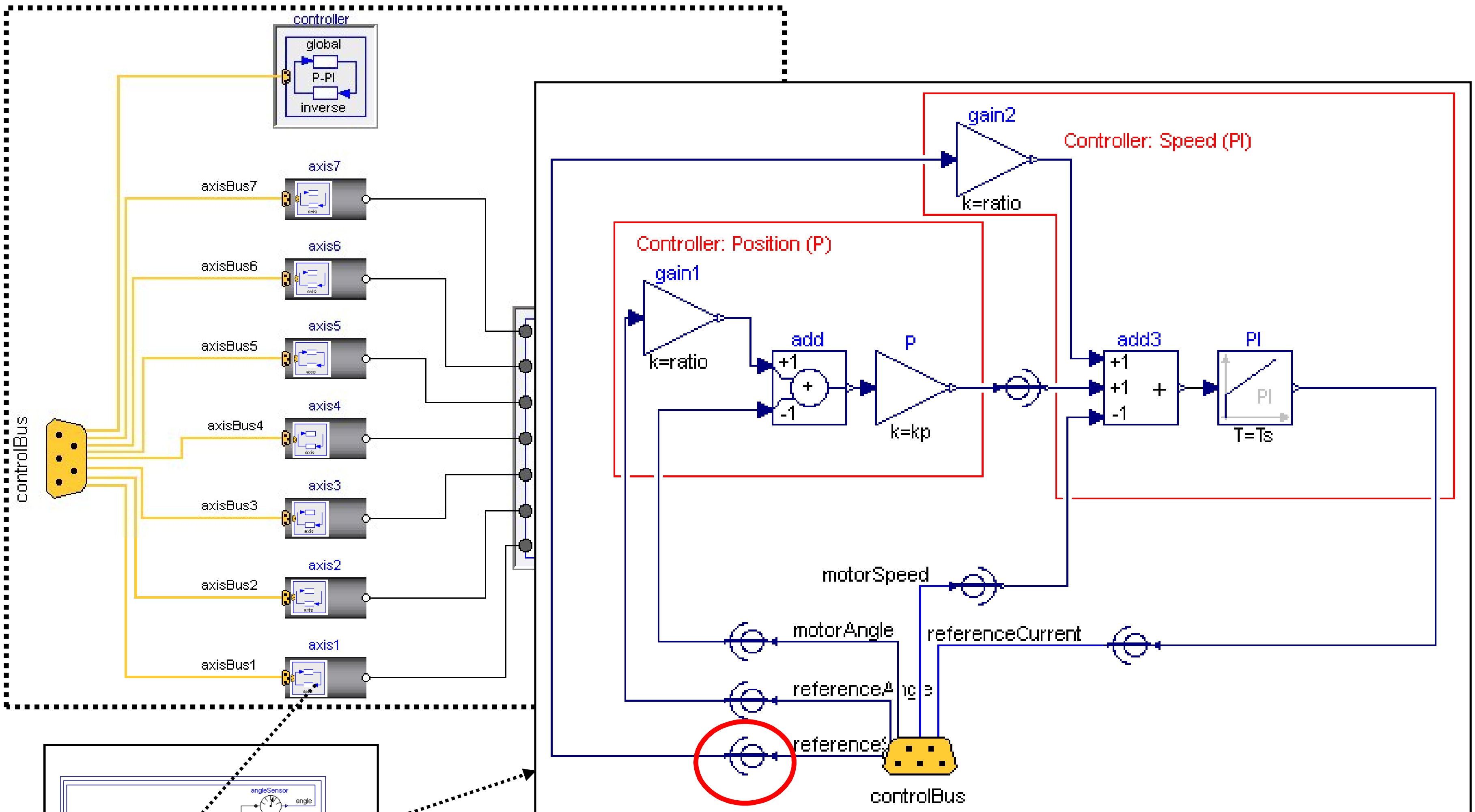


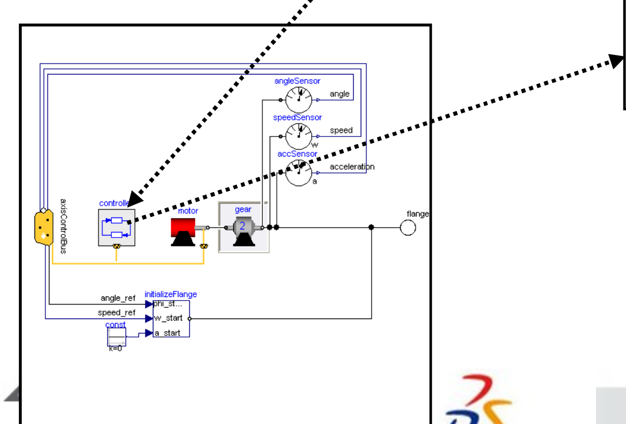
Every mapping, like,
- continuous/sampled simulation,
- download to one or three processors,
- other device drivers
- with/without floating point unit on target

requires copying and restructuring of model

Example: New Approach

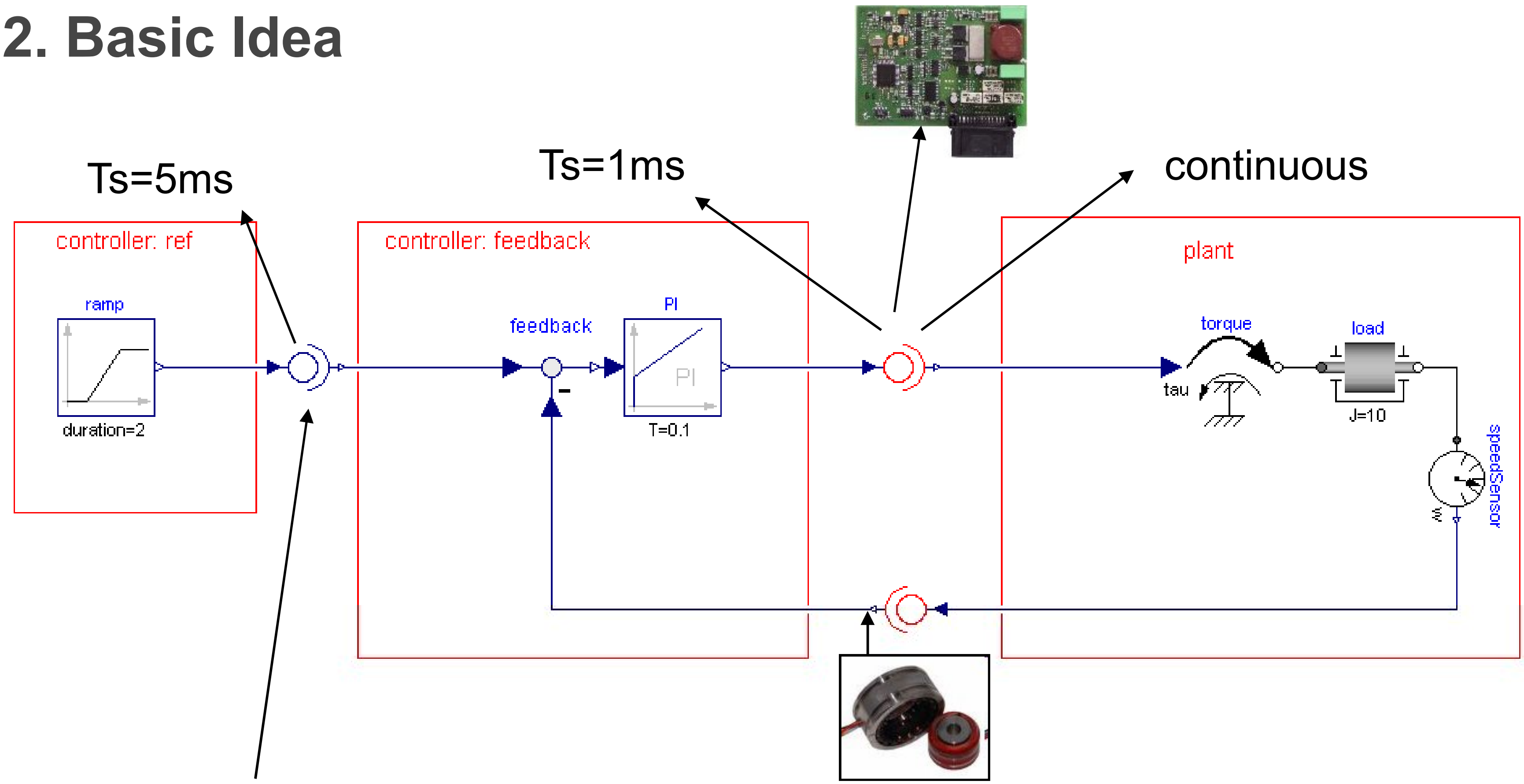




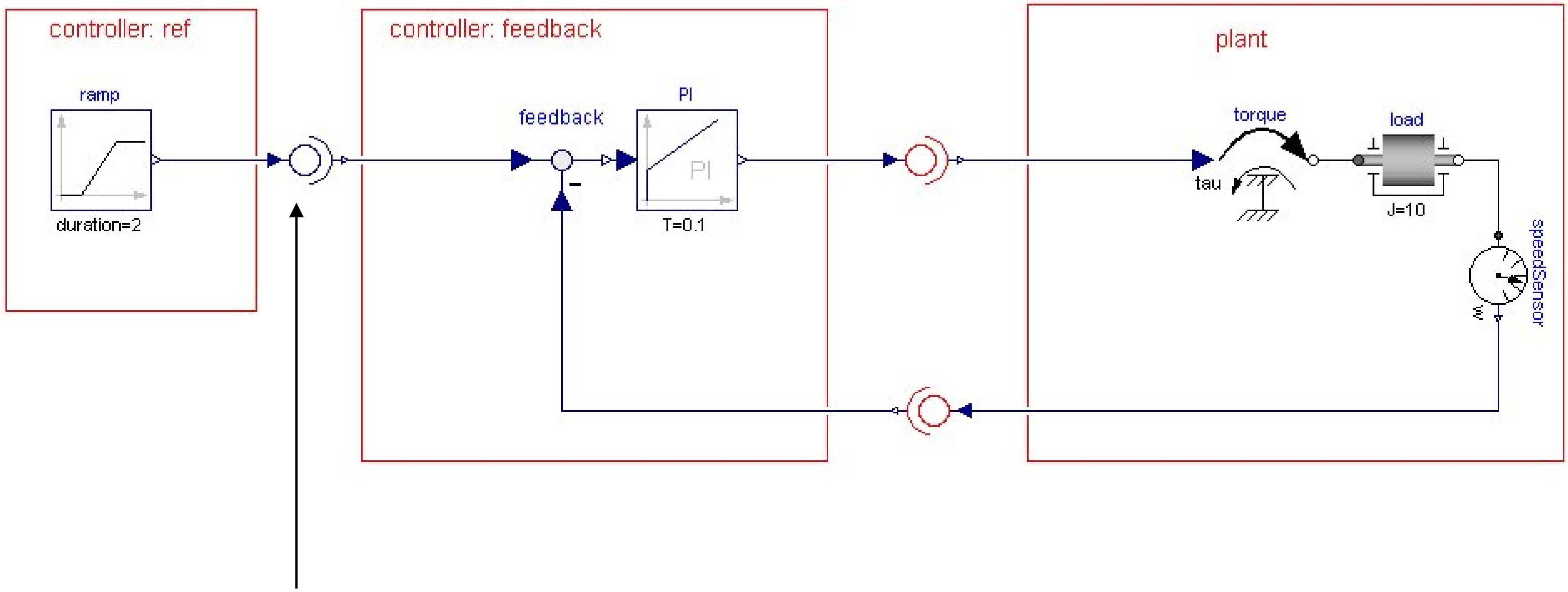


Important:
Mark submodels to be specially handled

2. Basic Idea



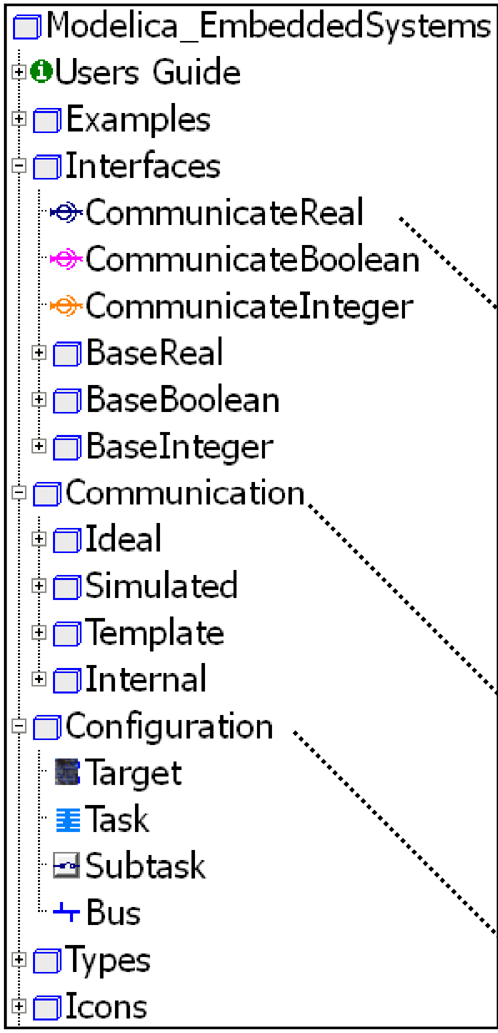
1. Mark boundaries ⟶ Logical model is partitioned in to submodels
2. Make a new model and inherit from logical model
3. Add submodel properties at the boundaries (for input and/or output)
4. Add device drivers at the boundaries (replaceable models)
5. Add target definitions at the boundaries; download selected submodels



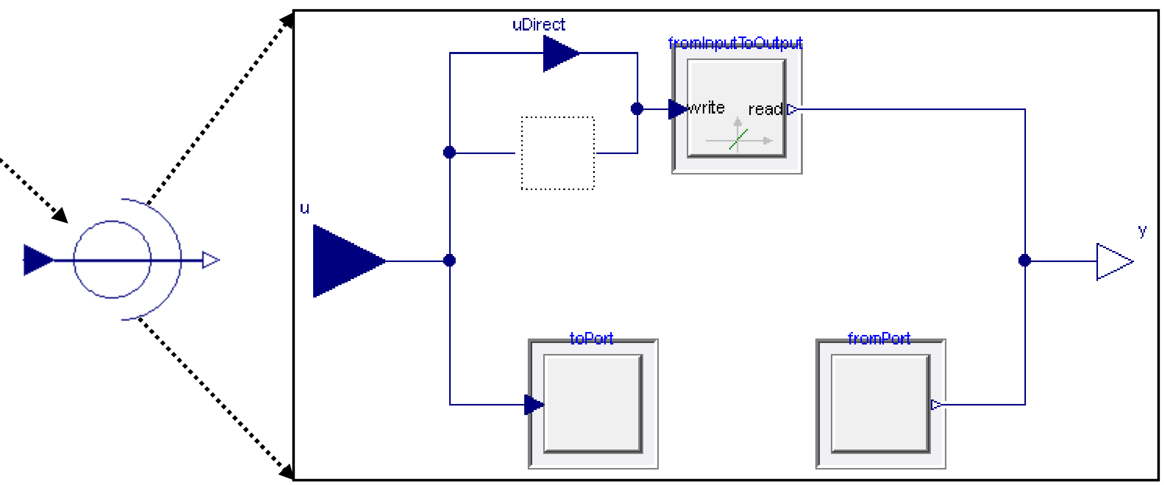
Different communication possibilities:

- Direct communication (ideal simulation: y = u)
- Simulated communication (y = f(u); sampling, delay, value discretization, ...)
- Communication between two subtasks
  (in same task but different sampling; synchronous equations)
- Communication between "two tasks", or "task to device" or "device to task"
  (communication via devices, e.g., shared memory, UDP, CAN-bus, ...;
  asynchronuous equations)

3. Modelica_EmbeddedSystems Library



Free library to define the mapping for embedded systems



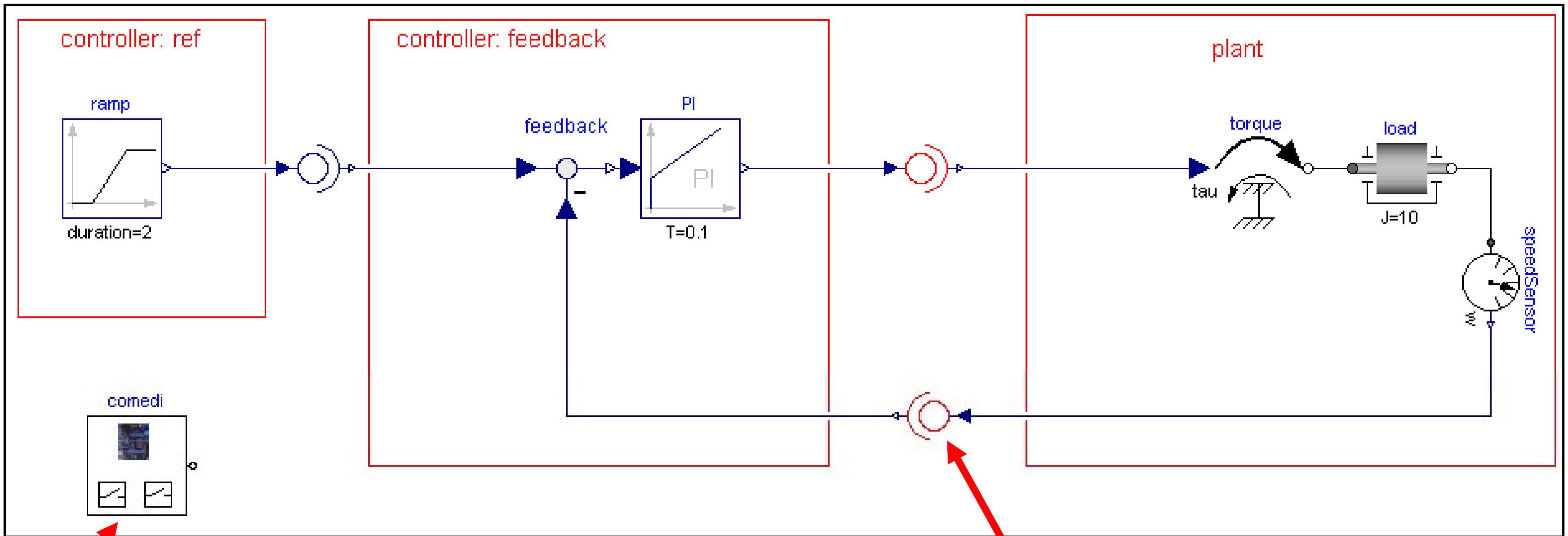
Actual blocks for communication

Defining task, subtask, target

replaceable blocks for

- direct communication (for simulation)
- signal to hardware driver
- signal from hardware driver

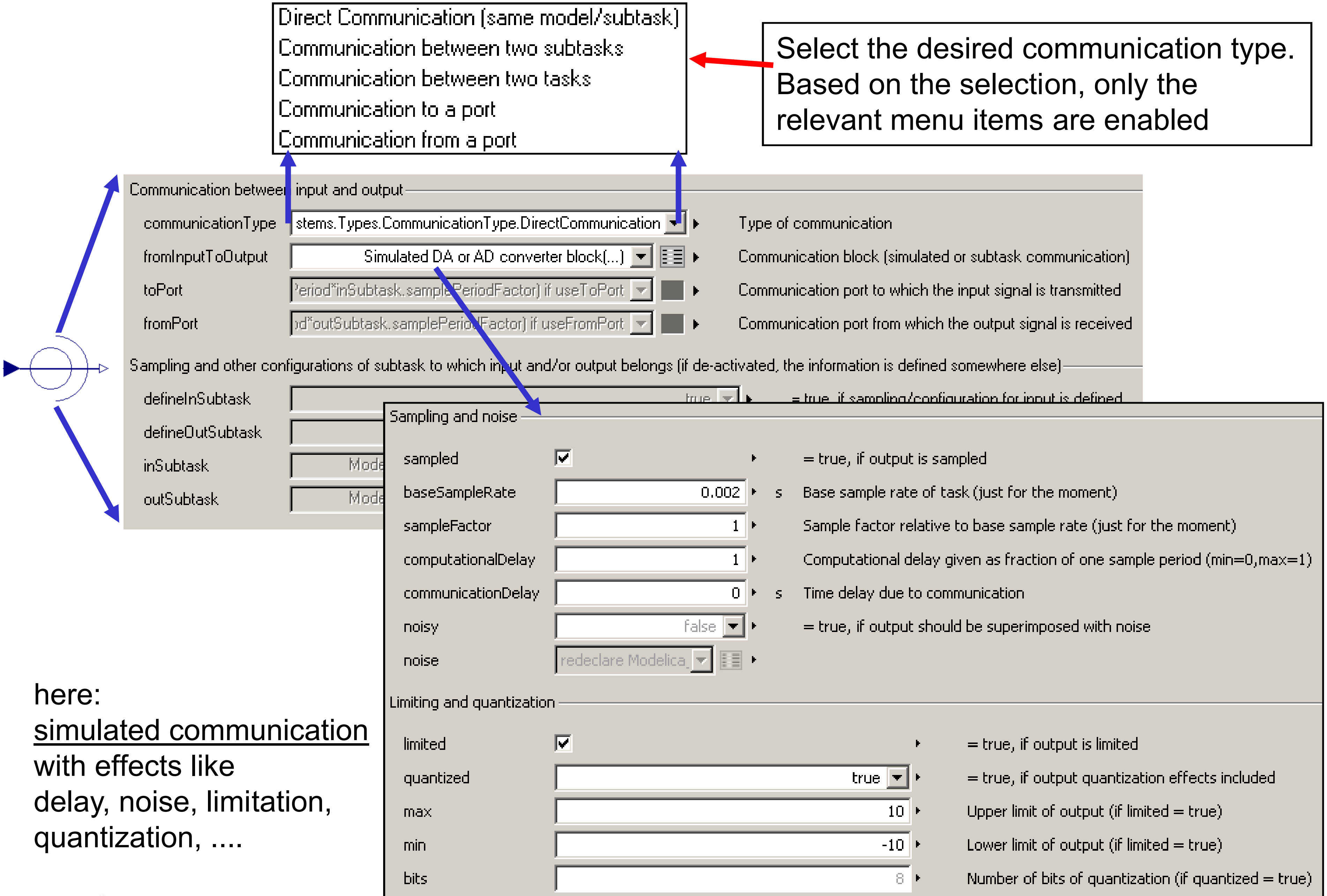
Example



defines configuration of embedded system (sampling, ECUs, initialization of device drivers, ...)

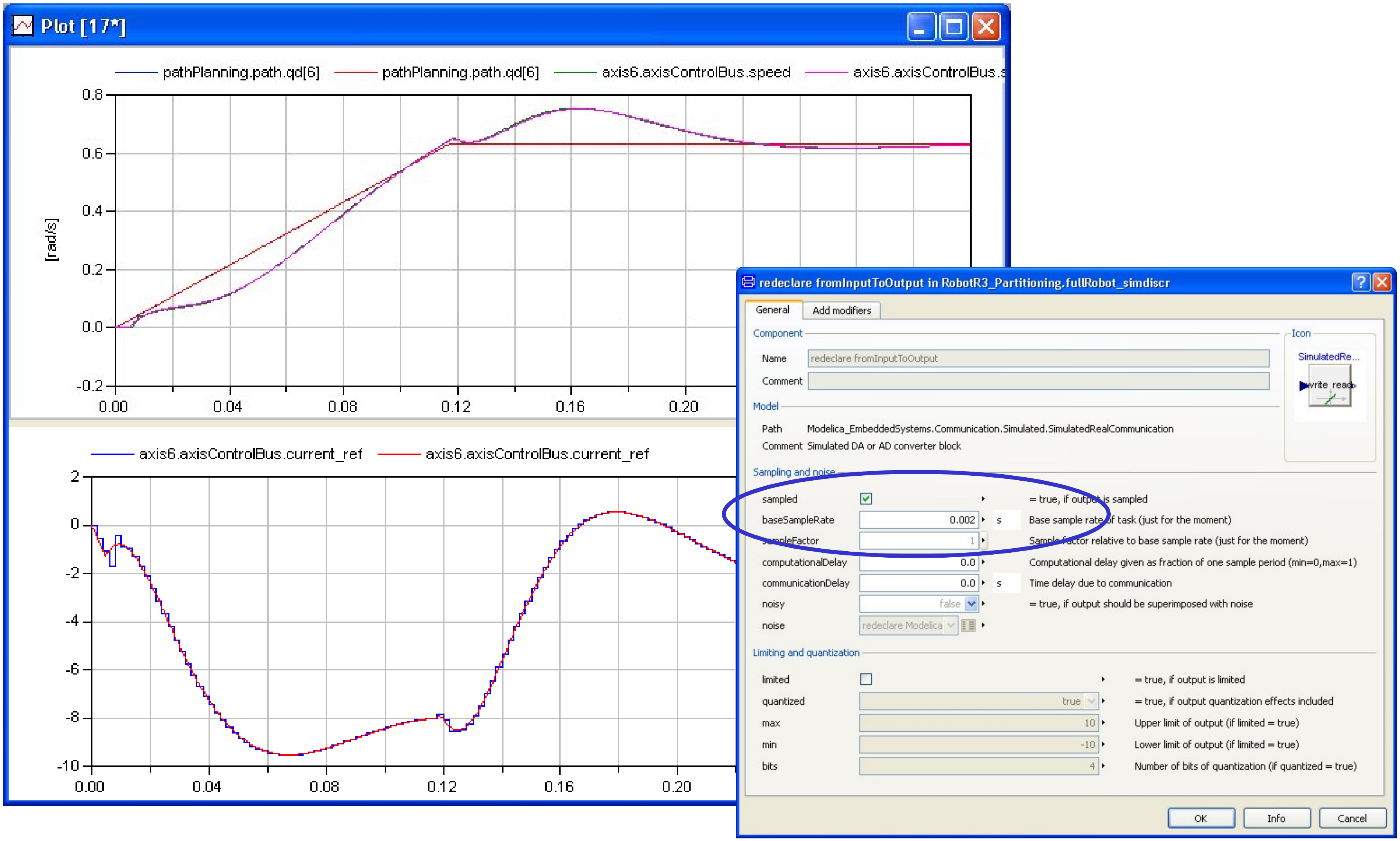
Defines splitting in task/sub-task, device drivers, references configuration

Select the desired communication type. Based on the selection, only the relevant menu items are enabled

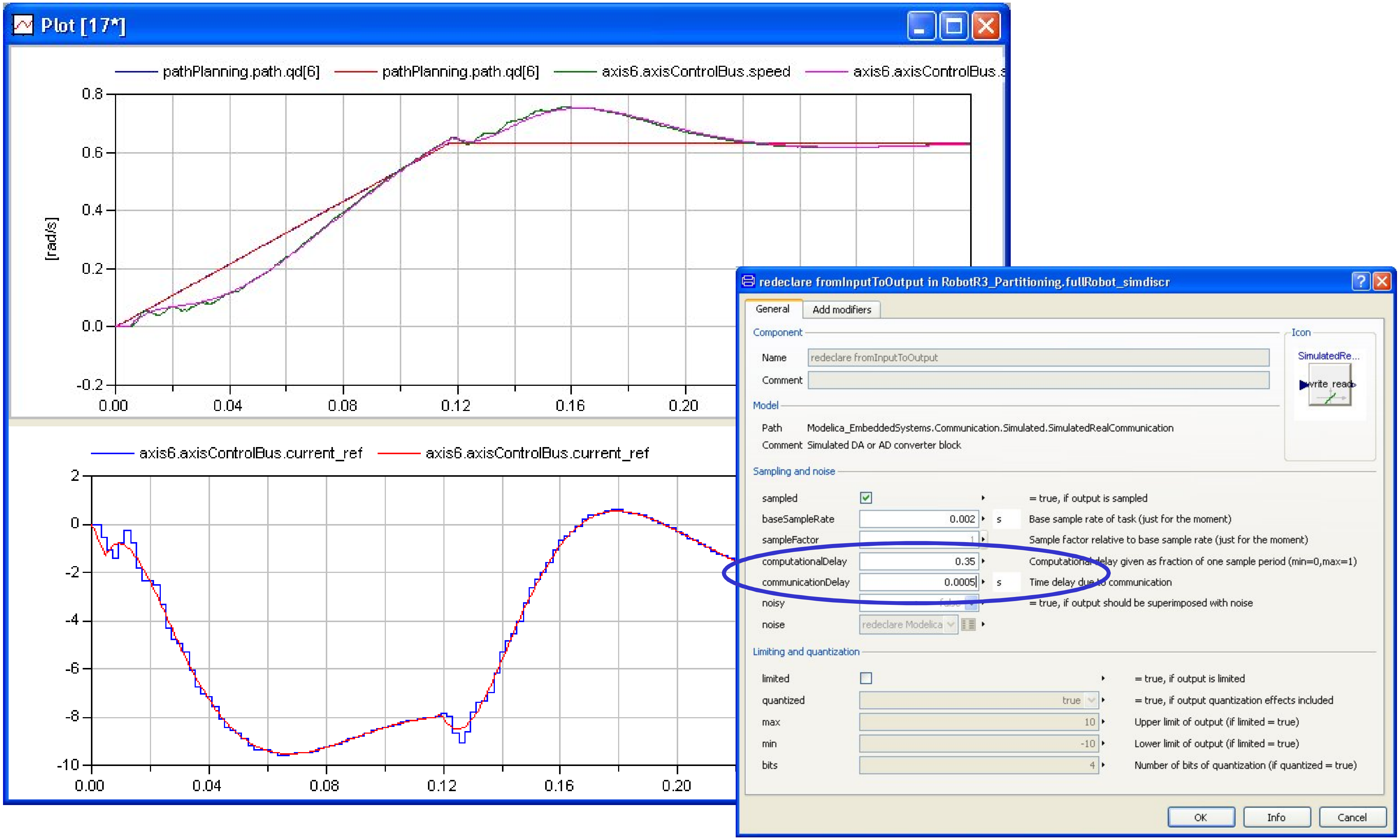


here:
<u>simulated communication</u>
with effects like
delay, noise, limitation,
quantization, ...

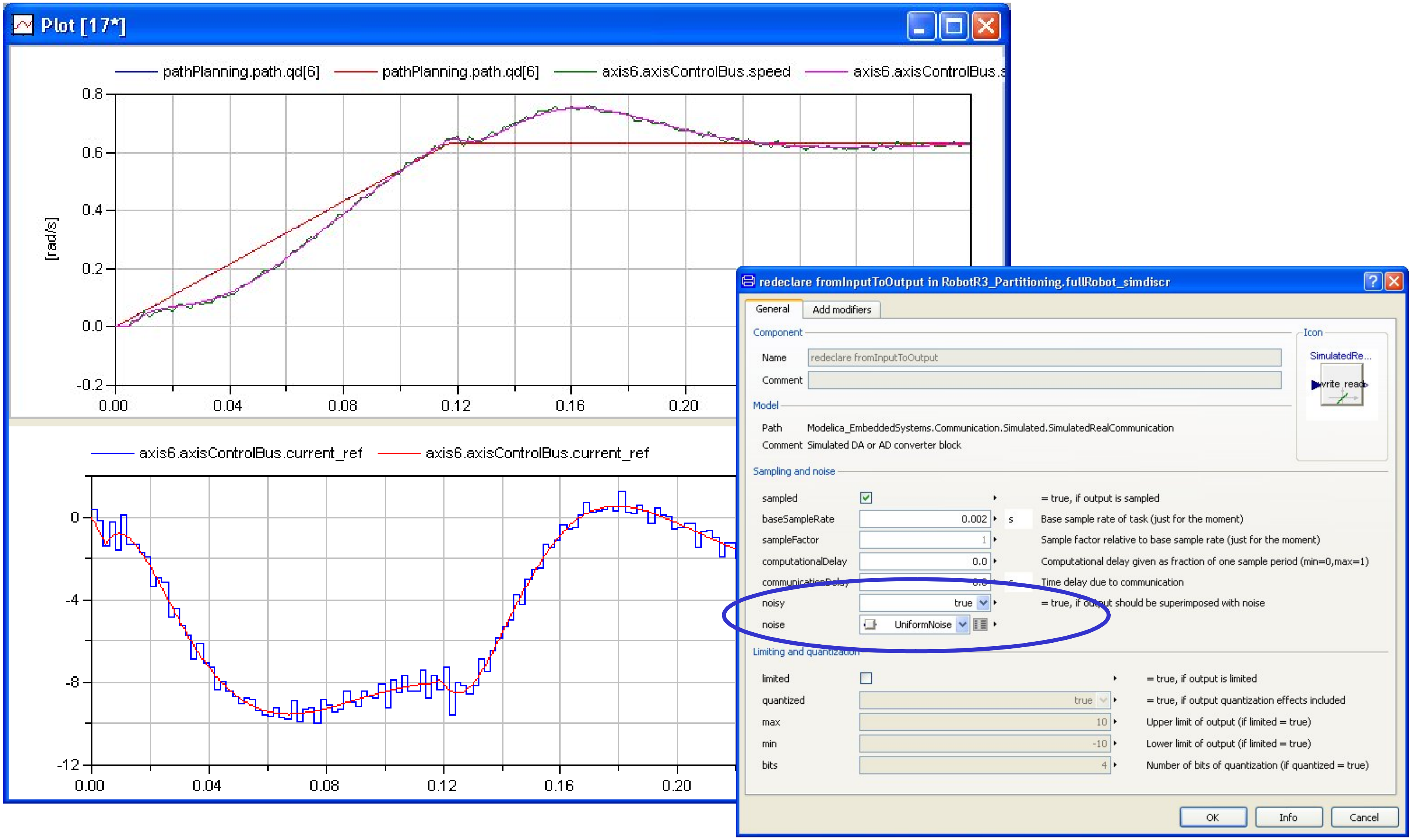
Sampled Controller



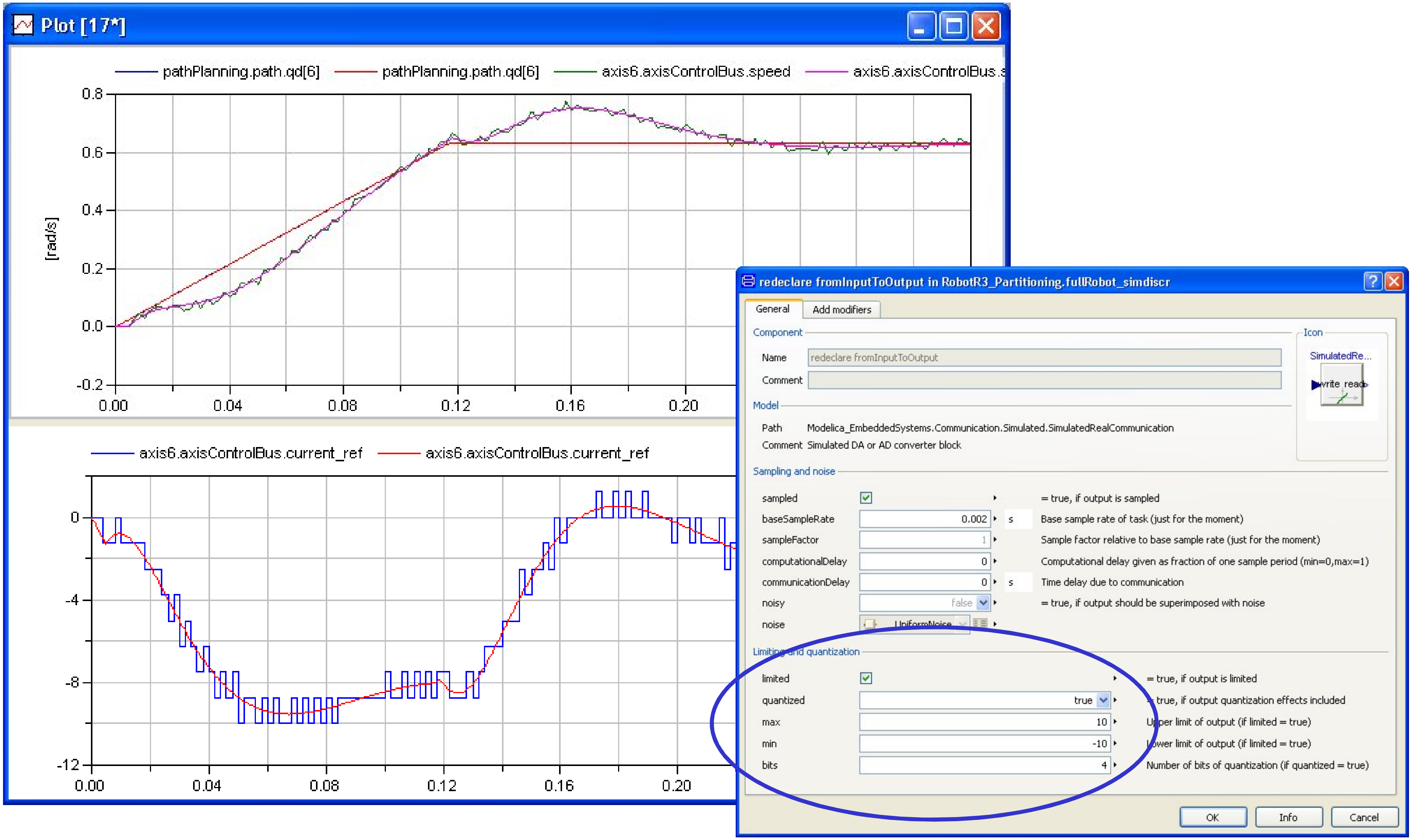
Sampled Controller with Delay

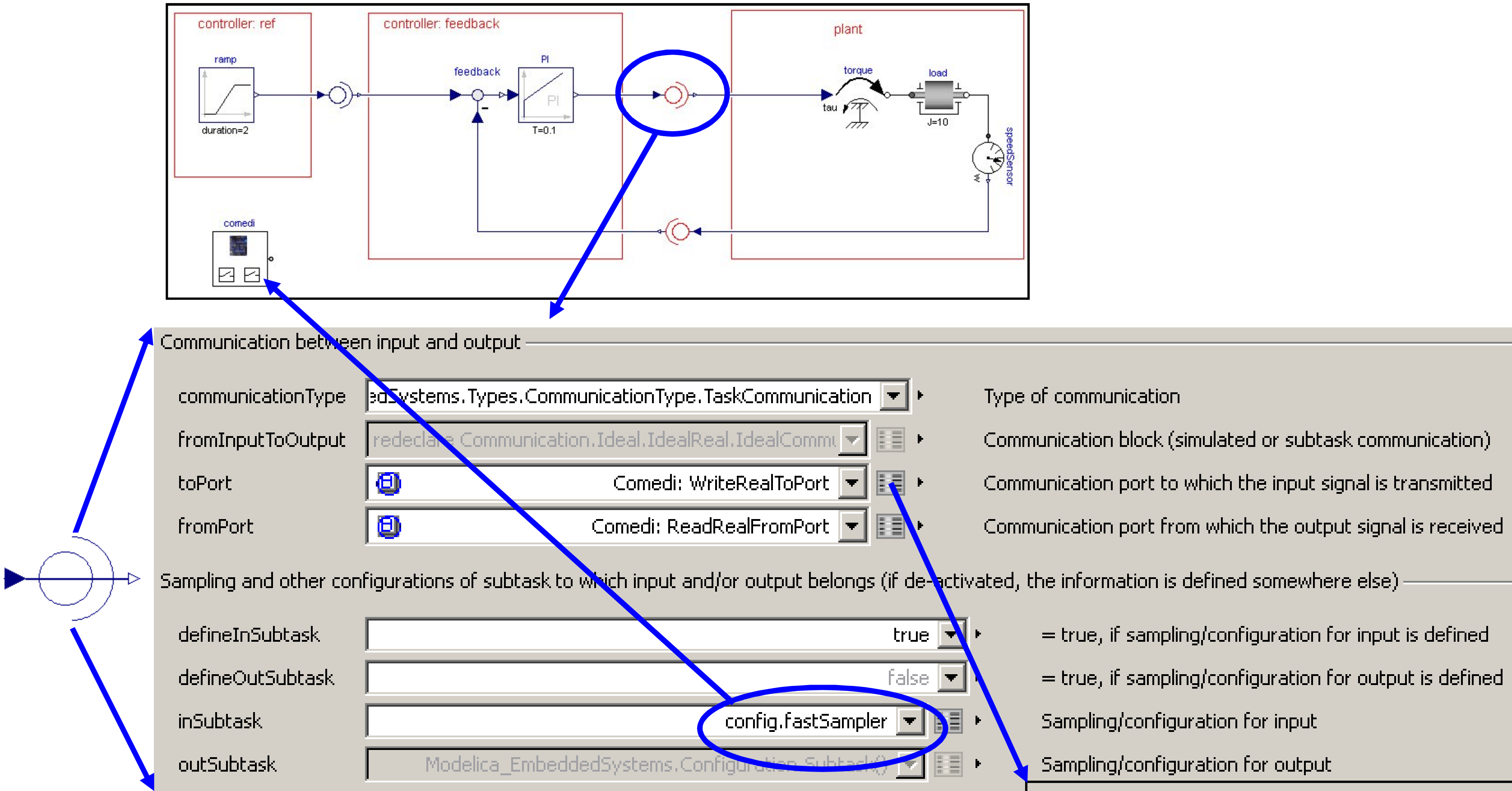


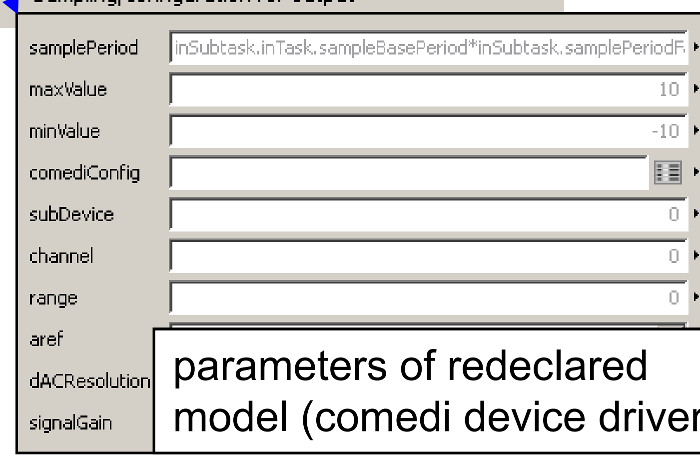
Sampled Controller with Input Noise



Sampled Controller with Output Quantization

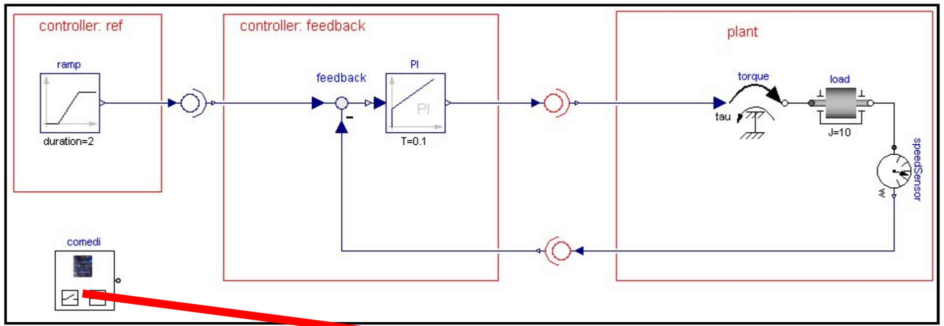






parameters of redeclared model (comedi device driver)

here:
<u>communication between two tasks</u>
input : Writes to hardware (comedi driver)
ouput: Reads from hardware (comedi driver)
The properties of the task at the input are defined by config.fastSampler

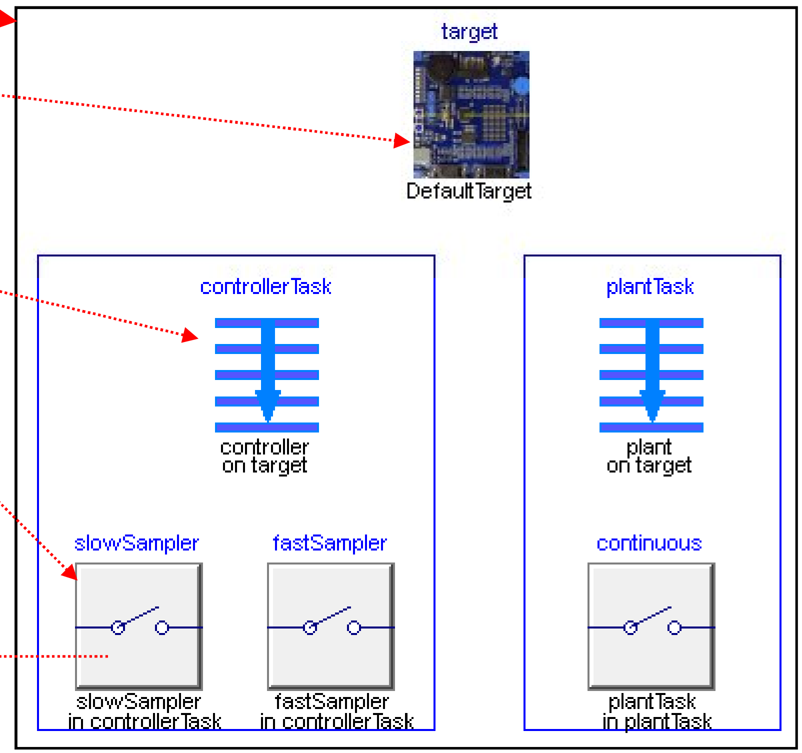


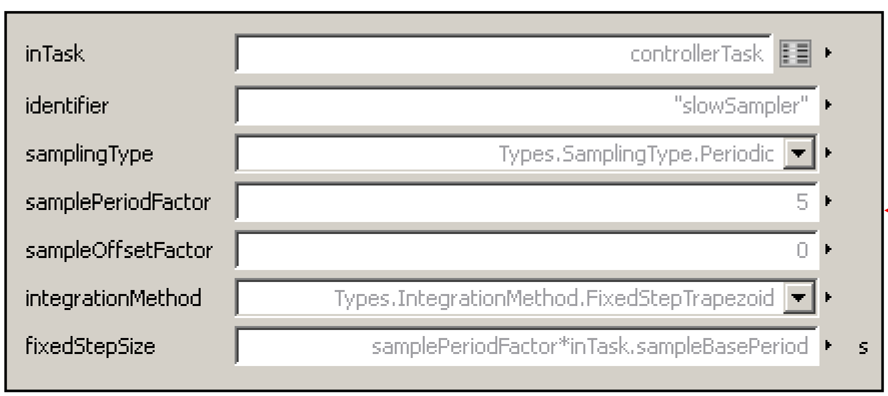
Configuration of architecture (here: multi-tasking on one machine)

Target machine

Task and processor (on target machine)

Sampling and integrator (in task)





4. Modelica Language Extensions

Modelica extensions developed for the mapping concept:

- `u = Subtask.decouple(y);`
  (a) u = y; (b) u and y are in different subtasks.

- ```
  parameter Modelica.SIunits.Time sampleBasePeriod;
  RealInput u annotation(mapping(
                target (identifier = "abc"),
                task   (identifier = "slowTask",
                        sampleBasePeriod = sampleBasePeriod),
                subtask(identifier = "reference",
                        samplingType=Subtask.SamplingType.Periodic)))
  ```

- Extension included in Modelica 3.1

- Partial prototype in Dymola

- Several device drivers (some will be made public)
  ```

5. Outlook

Near future:

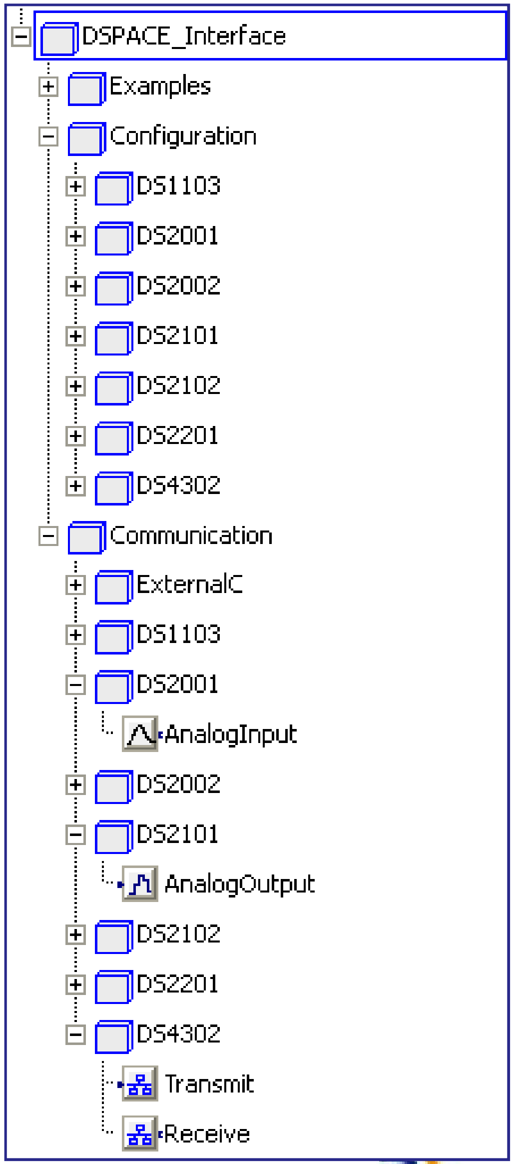
- Full support in Dymola in the near future.
- Release of Modelica_EmbeddedSystems, including some free device drivers (keyboard, game controller, PC speaker, ..)

Planned for Modelica 3.2:

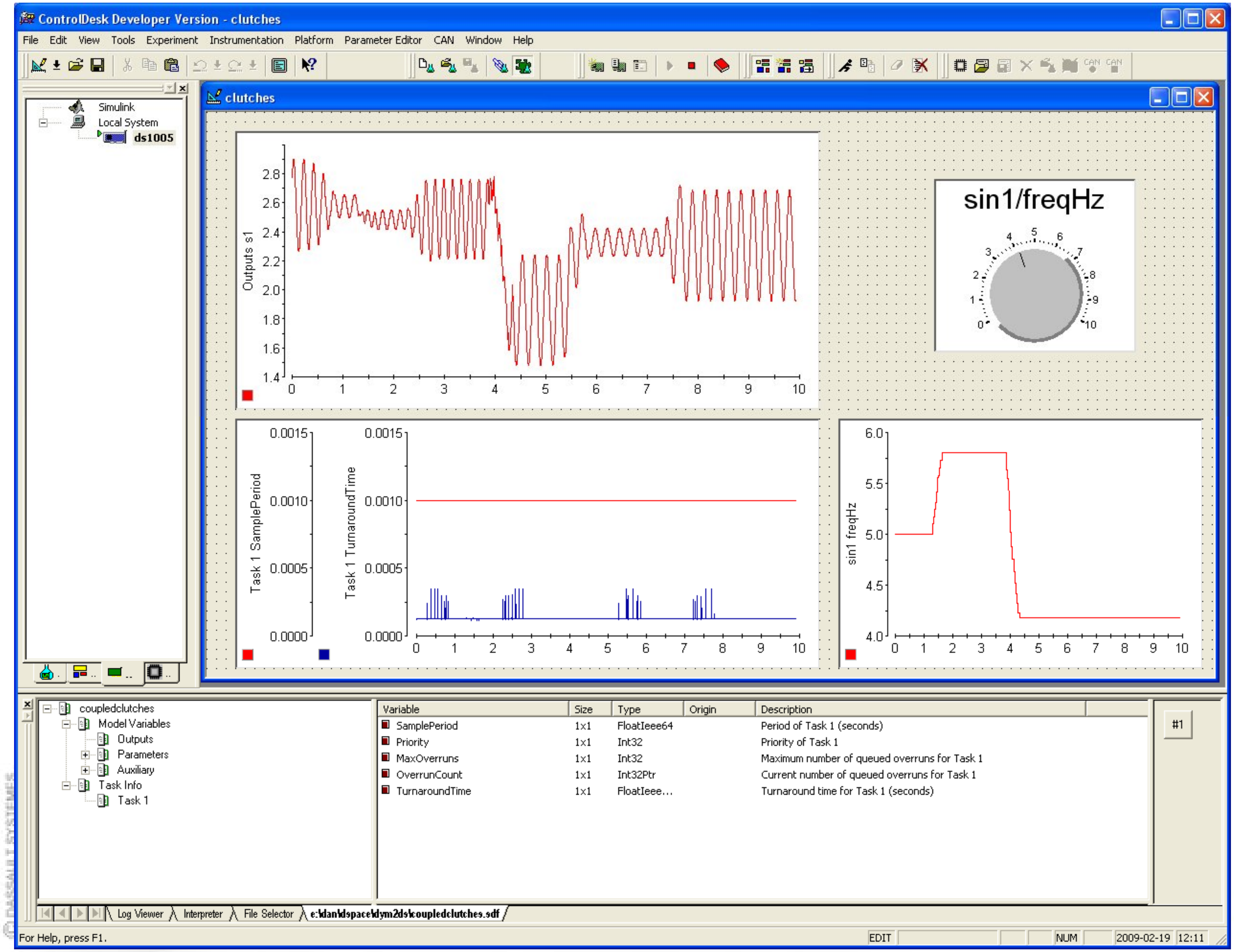
- Extension of concept for enabled/triggered tasks (currently only for continuous and for periodic tasks)
- Improving definition of "time" and of "event accuracy":
  "time" is an integer type of defineable precision per partition (e.g. 1ms)
  "events" occur only at multiples of base time.
- Built-in timer to simplify time event definitions
- Define mapping of "Real" to "Integer type" (if no floating point unit on target)

Interface to dSPACE Systems

- **Allow HILS and rapid prototyping of Dymola models on dSPACE targets *without* The Mathworks RTW**
- **Simulation framework and I/O drivers implemented from dSPACE real-time library APIs**
- **Configuration and communication blocks based on Modelica_EmbeddedSystems**

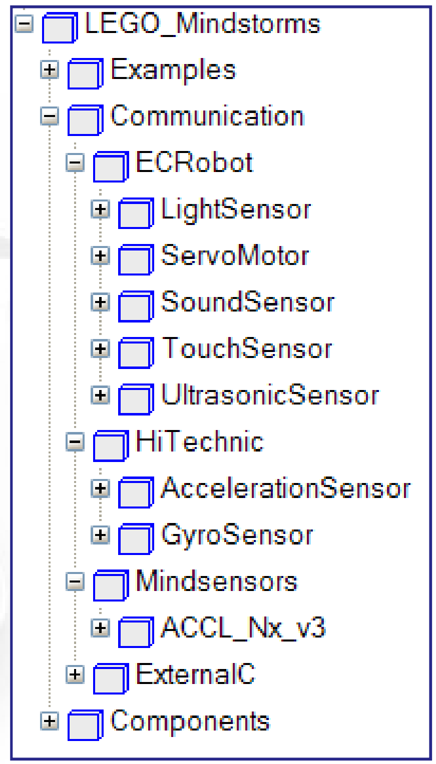


Interface to dSPACE Systems



Interface to LEGO Mindstorms NXT

- **Standard I/O components**
  - Light sensor
  - Sound sensor
  - Touch sensor
  - Ultrasonic sensor
  - Servo motor
- **Third party sensors**
  - Acceleration sensor (HiTechnic)
  - Gyro sensor (HiTechnic)
  - Acceleration sensor (Mindsensors)
- **Bluetooth communication**
- **Atmel ARM7 main processor**
- **Atmel ATmega48 microcontroller for A/D and PWM**



Code Generation for Fixed-point Arithmetics

- ## Declarations

```
/* output Modelica.Blocks.Interfaces.RealOutput ramp.y(
   min = 0.0, max = 100.0) annotation(mapping(
   resolution = 0.01)); */
int ramp_yFP = 0;   /* Q[7, 0] */

/* parameter Modelica.SIunits.Time ramp.duration(
   min = 0.0, max = 50.0) = 10
   annotation(mapping(resolution = 0.001)); */
int ramp_durationFP = 320;  /* Q[6, 5] */

/* parameter Real ramp.height(min = 0.0,
   max = 100.0) = 100  annotation(mapping(
   resolution = 0.001)); */
int ramp_heightFP = 1600;   /* Q[7, 4] */
```

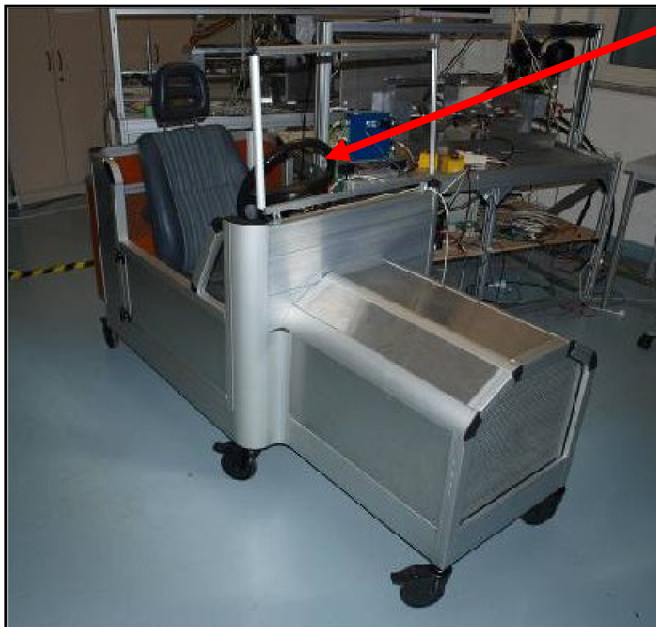
- ## Equations

```
/* ramp.y = ramp.offset+(if time < ramp.startTime
  then 0 else (if time < ramp.startTime+ramp.duration
  then (time-ramp.startTime)*ramp.height/
  ramp.duration else ramp.height)); */
ramp_yFP = (((ramp_offsetFP << 4) + ((((timeFP0_0 << 4)
  < ramp_startTimeFP) ? (0 << 4) : ((((timeFP0_0 << 4)
  < (ramp_startTimeFP + (ramp_durationFP << 9))) ?
  ((((((timeFP0_0 << 4) - ramp_startTimeFP) / 32) *
  (ramp_heightFP / 32)) / ramp_durationFP) << 1) :
  ramp_heightFP)))))) / 16;
```

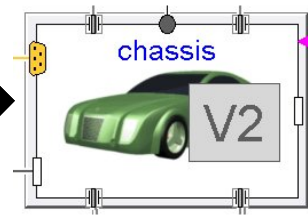
Lego Segway control

Modelica external device applications at DLR

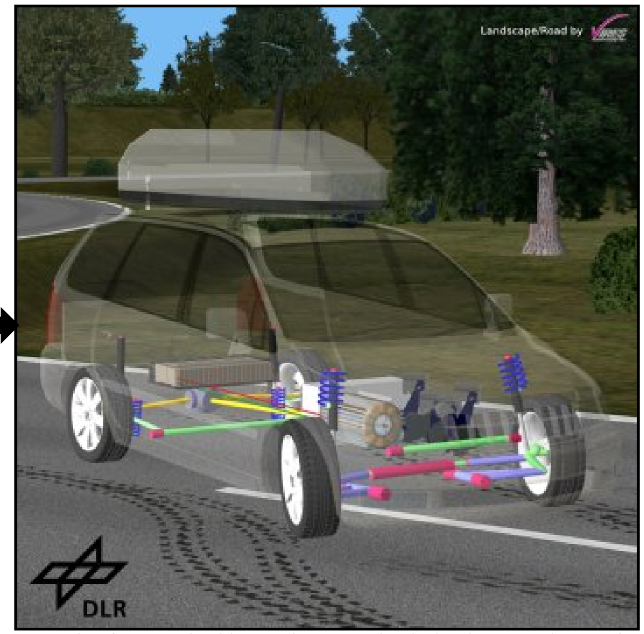
- Extensive use of human interface devices for interactive (real-time) simulations in two major use cases
  1. To help *simulation developers* to quickly get interactive feedback about the reaction of their model to certain stimula
  2. For real-time simulators including prototype hardware components (e.g. Force-feedback steering wheel for Steer-by-Wire application)



Vehicle Mockup with force-feedback from (Modelica) simulation



Modelica model



Dymola/Modelica based driving simulation (visualized with DLR Visualization library)

Use cases in the simulator development process

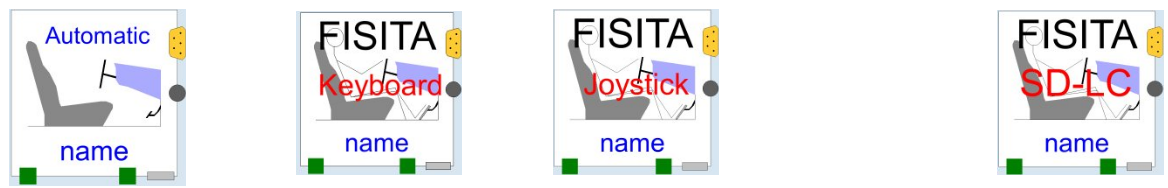
Non-interactive simulation

Interactive model stimula for simulation developers

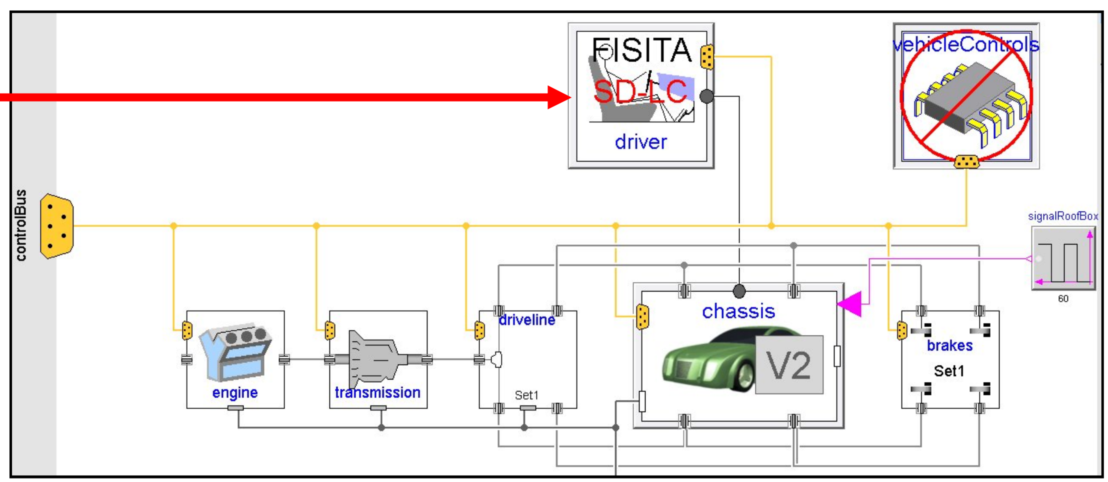
Force-feedback steering wheel for real-time driving simulation

Hardware device input Blocks

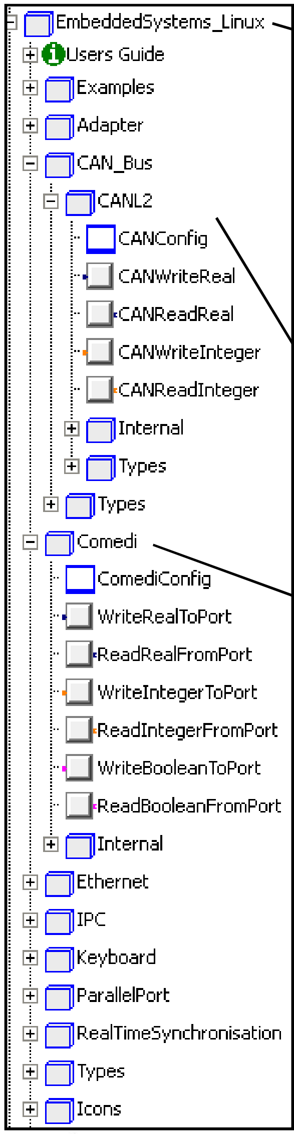
Several driver implementation



Replaceable driver model



Overview of device libraries used at DLR

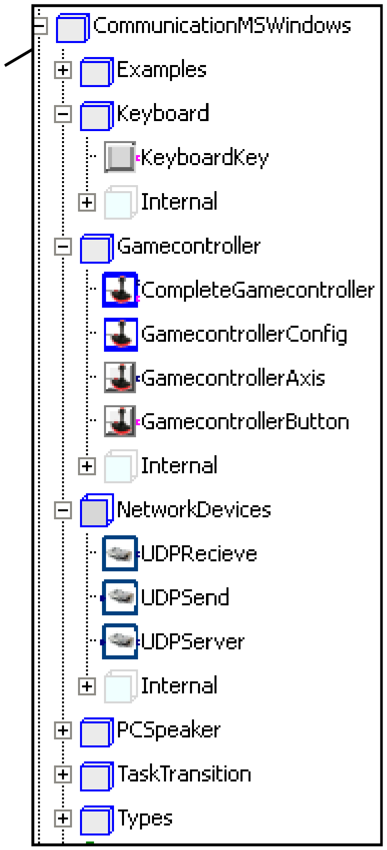


(Real-time) Linux devices (based on *Modelica_EmbeddedSystems*)

Support for Softing‘s **CAN-Bus** interface

Support for the **Comedi** linux control and measurment device interface

Windows device support



A New Formalism for Modeling of Reactive and Hybrid Systems

Martin Otter

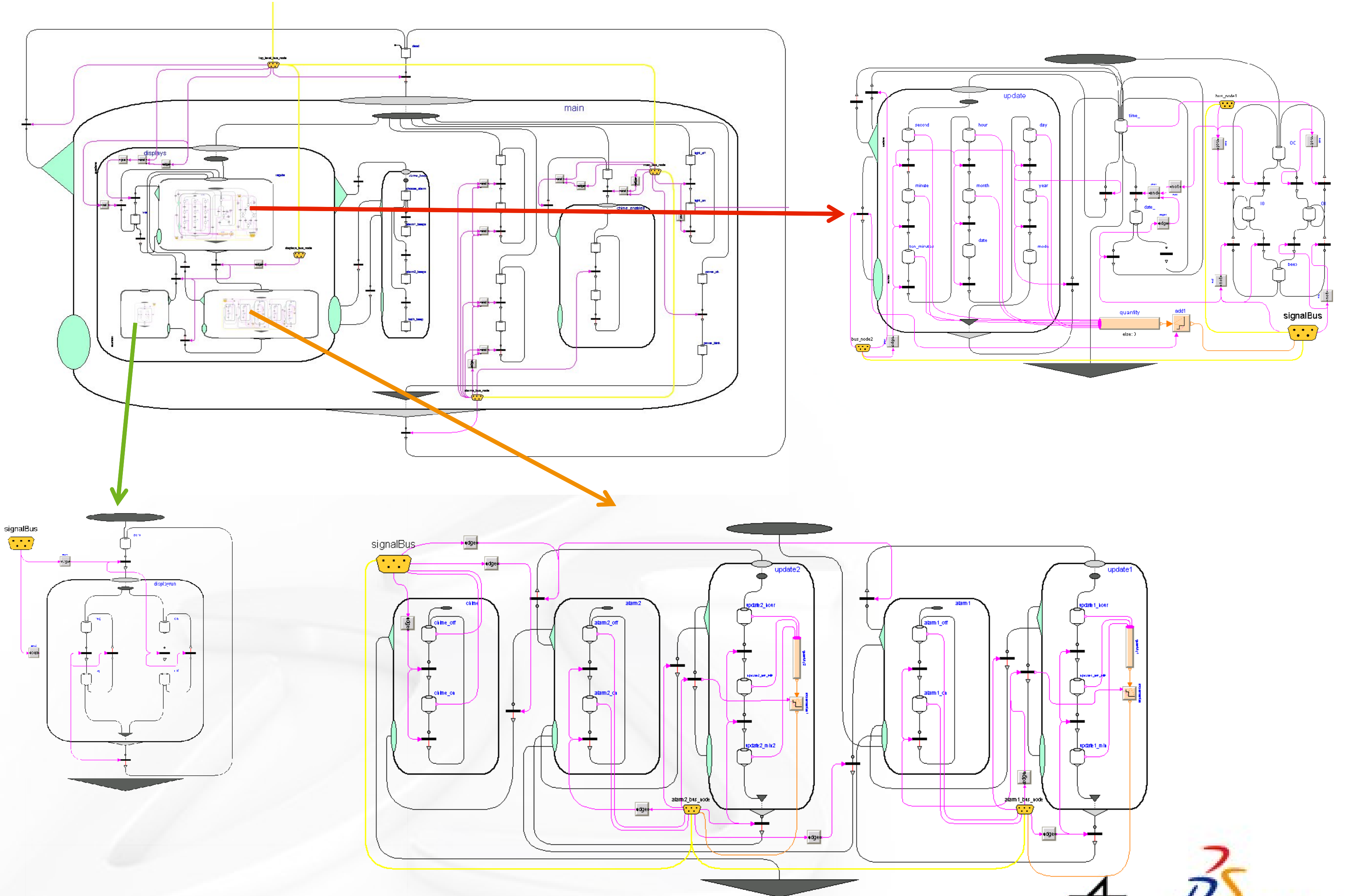
Martin Malmheden

Hilding Elmqvist

Sven Erik Mattsson

Charlotta Johnsson

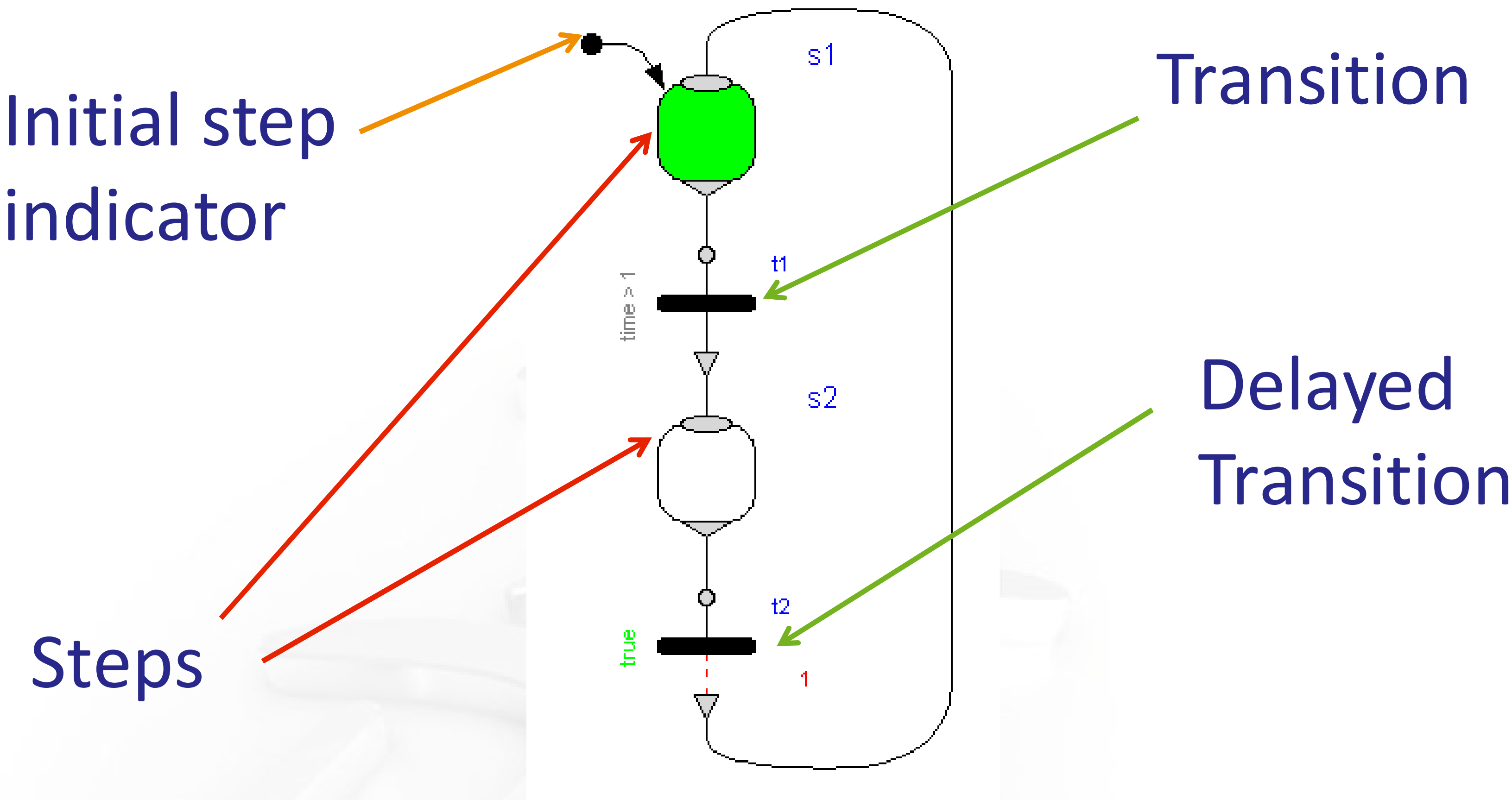
Modeling large reactive systems made easy



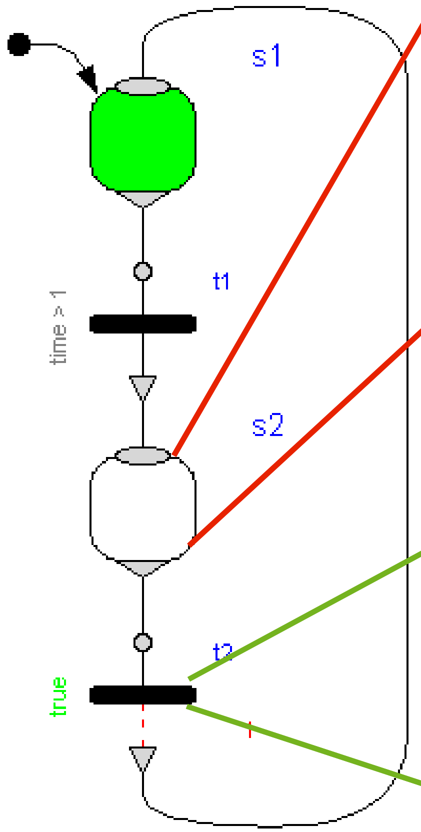
StateGraph2 - Advantages

- **Further improved usability**
  - Aggregations with open Icon layer
  - Automatic connector sizing
- **New graphical approach to Mode-Automata**
  - Same basic idea as Mode-Automata
  - Purely graphical approach gives easy overview
- **Safer graphs**
  - Guaranteed convergence of event iterations
  - Not possible to build unsafe graphs
- **SMV-output**
  - Allows analysis with external tool
- **Formal definition**

Steps & Transitions



Steps & Transitions are implemented in pure Modelica




```
equation
  // set active state
  inport_fire  = Modelica_StateGraph2.Blocks.BooleanFunctions.anyTrue(inPort.fire);
  outport_fire = Modelica_StateGraph2.Blocks.BooleanFunctions.anyTrue(outPort.fire);

  newActive = if node.resume then oldActive else
                inport_fire or (active and not outport_fire) and not node.suspend;
  active = pre(newActive);

  // Remember state for suspend action
  when node.suspend then
    oldActive = active;
  end when;
```

```
equation

  // Determine firing condition
  enableFire = localCondition and inPort.available;

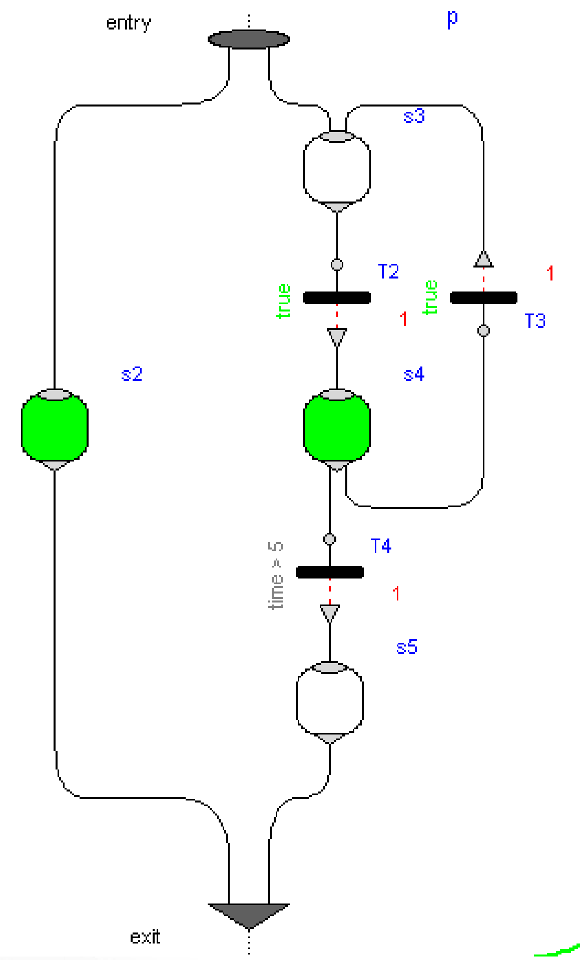
  if delayedTransition then
     when enableFire then
        t_start = time;
     end when;
     fire = enableFire and time >= t_start + waitTime;
  else
     t_start = 0;
     fire = enableFire;
  end if;

  inPort.fire  = fire;
  outPort.fire = fire;
```


DLR DASSAULT SYSTEMES

The Parallel Component cont'd

- **Parallel branching**
  - Every branch must be connected to the entry port
  - Synchronization by connecting branches to the exit port
- Requires inport and outport



DLR DASSAULT SYSTEMES

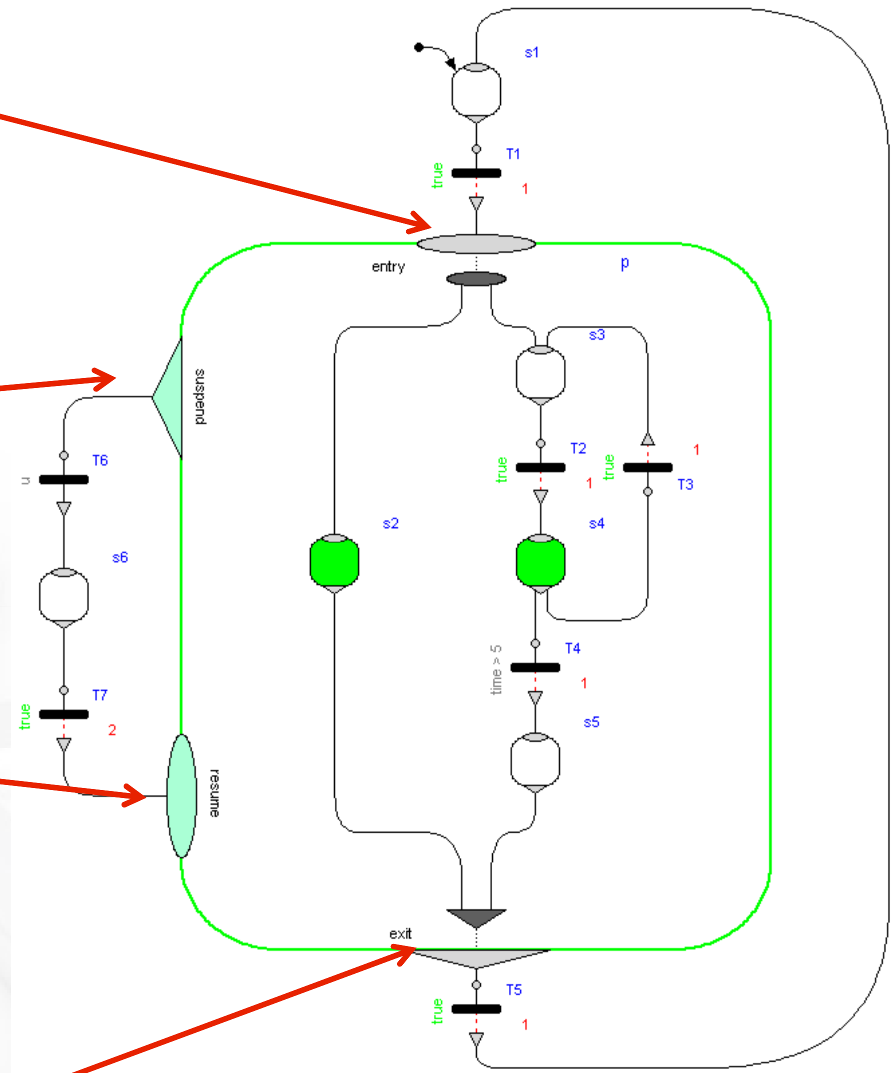
The Parallel Component cont'd

Inport

Suspend port

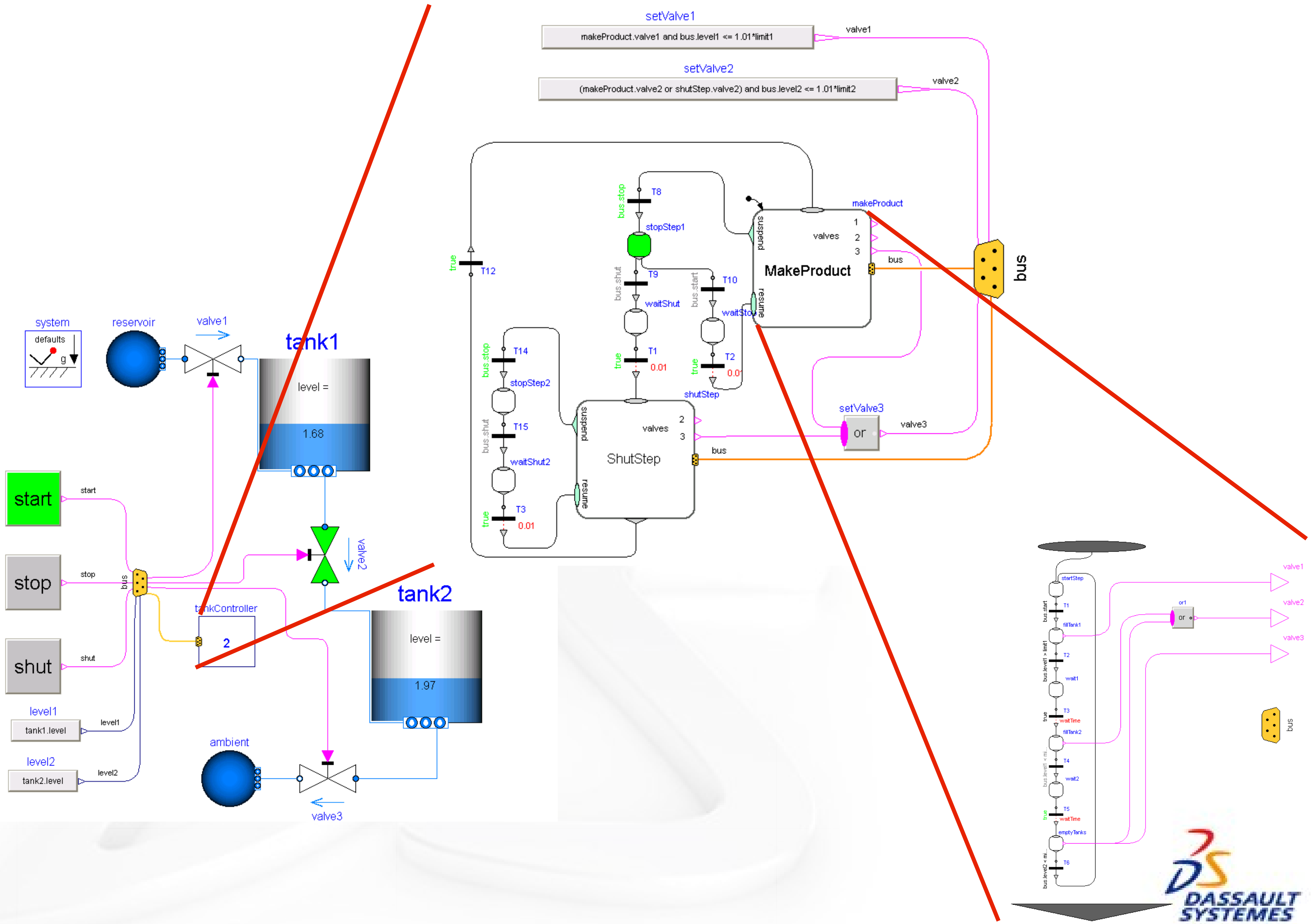
Resume port

Outport



DLR DASSAULT SYSTEMES

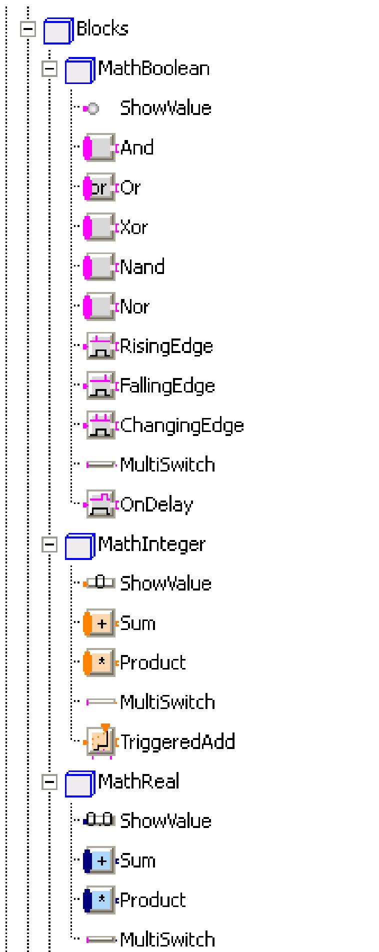
Hierarchical Hybrid System – Tank example

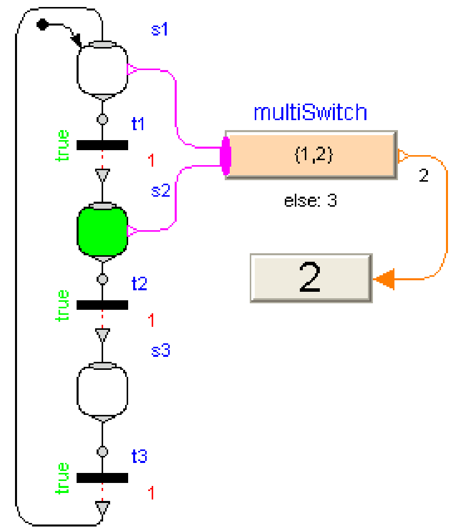


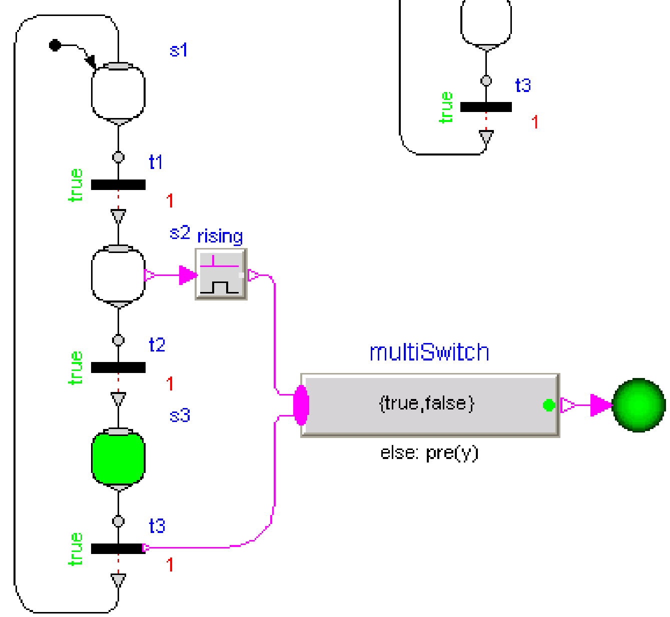
DASSAULT SYSTEMES

Graphical assignment of actions

Introducing MultiSwitches

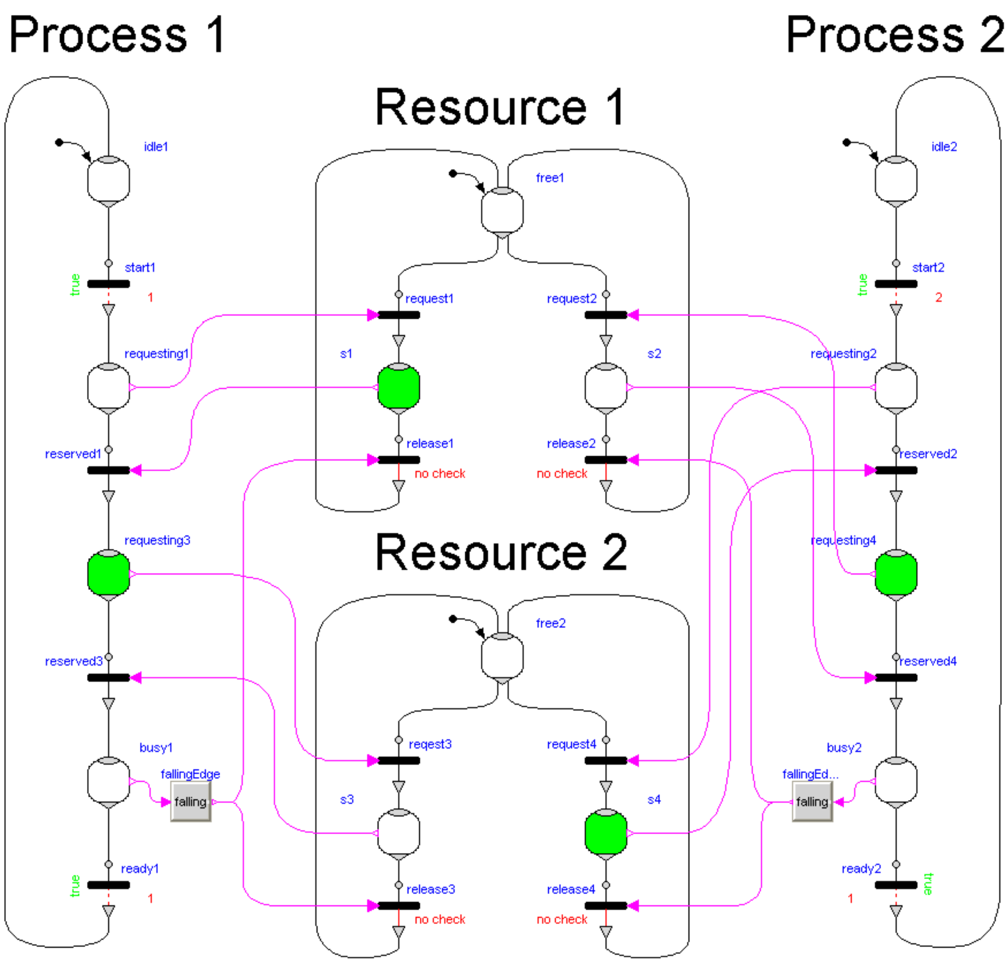






Verification of reactive systems

- **Export to SMV code for external analysis**



Summary

- User friendly
- Improved graphical approach to variable assignment
- Safe – not possible to make dangerous graphs
- Flexible
- Allows external analysis of graph structure

Functional Mockup Interface – Overview

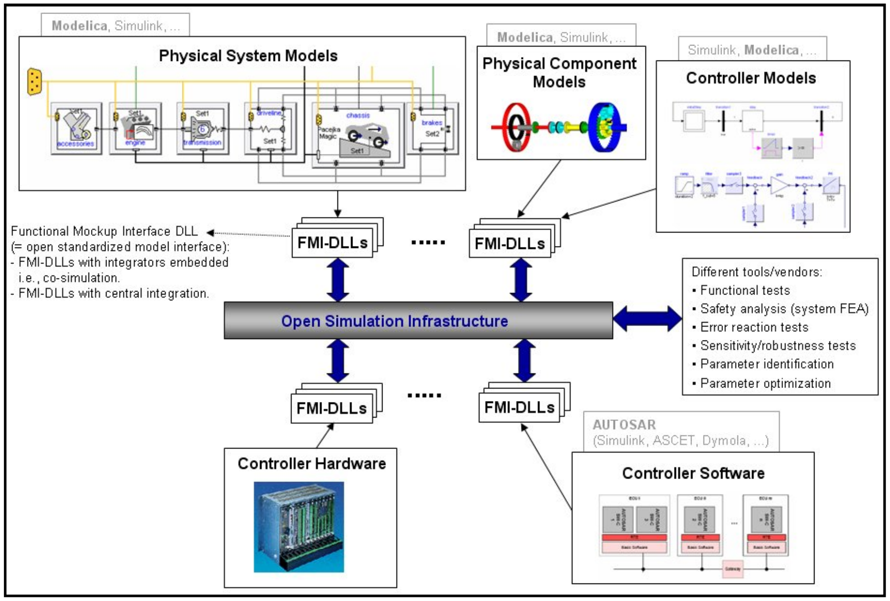
Martin Otter (DLR-RM)
Torsten Blochwitz (ITI)
Hilding Elmqvist (Dassault Systèmes – Dynasim)
Andreas Junghanns (QTronic)
Jakob Mauss (QTronic)
Hans Olsson (Dassault Systèmes – Dynasim)

Contents

1. Functional Mockup Interface (FMI) — Goals

Overall goal of FMI in MODELISAR

Software/Model/Hardware-in-the-Loop,
of **physical** models and
of **AUTOSAR** controller models
from **different vendors** for
automotive applications with
different levels of detail.



Concrete goal of FMI in MODELISAR

... for (alphabetically ordered)

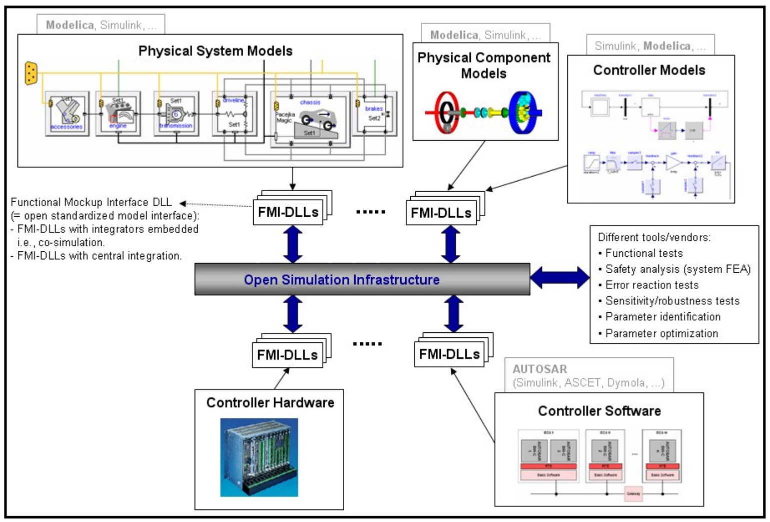
AMESim (Modelica, hydraulic)
Dymola (Modelica)
EXITE (co-simulation environment)
Silver (co-simulation environment)
SIMPACK (multi-body)
SimulationX (Modelica)
SIMULINK

Open Standard

Task is complex since the different parts are complex by themselves:

- **Model Exchange** (ODE/DAE components without integrators)
- **Co-Simulation** (ODE/DAE components with integrators)
- **Co-Simulation** with **PDE solver** (MpCCI)
- **AUTOSAR** (discrete components with complex communication)
- **Simulation Backplane**

"**Model Exchange**" is **most reliable** due to central step-size control.
Released January 2010.
Extension for co-simulation under development (Uni Halle, ITI, Fraunhofer)



2. FMI - Distribution of Model

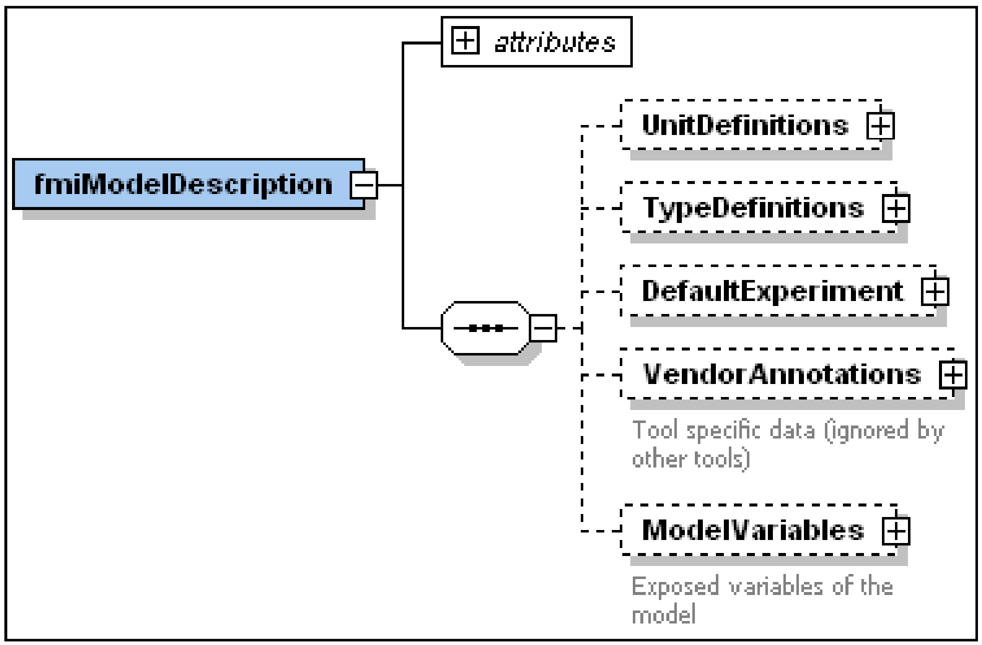
A model is distributed as one zip-file with extension "**.fmu**". Content:

```
modelDescription.xml          // Description of model (required file)
model.png                     // Optional image file of model icon
documentation                 // Optional directory containing the model
documentation
   _main.html                 // Entry point of the documentation
   <other documentation files>
sources                       // Optional directory containing all C-sources
   // all needed C-sources and C-header files to compile and link the model
   // with exception of: fmiModelTypes.h and fmiModelFunctions.h
binaries                      // Optional directory containing the binaries
   win32 // Optional binaries for 32-bit Windows
      <modelIdentifier>.dll   // DLL of the model interface implementation
      VisualStudio8           // Microsoft Visual Studio 8 (2005)
        <modelIdentifier>.lib   // Binary libraries
      gcc3.1               // Binaries for gcc 3.1.
   win64    // Optional binaries for 64-bit Windows
      ...
   linux32  // Optional binaries for 32-bit Linux
      ...
resources  // Optional resources needed by the model
   < data in model specific files which will be read during initialization >
```

3. FMI - Model Description Schema

All model information not needed for execution is stored in one xml-file (modelVariables.xml in zip-file)

Advantage:
Complex data structures give still simple interface, and tool can use its favorite programming language for reading (e.g., C++, C#, Java).

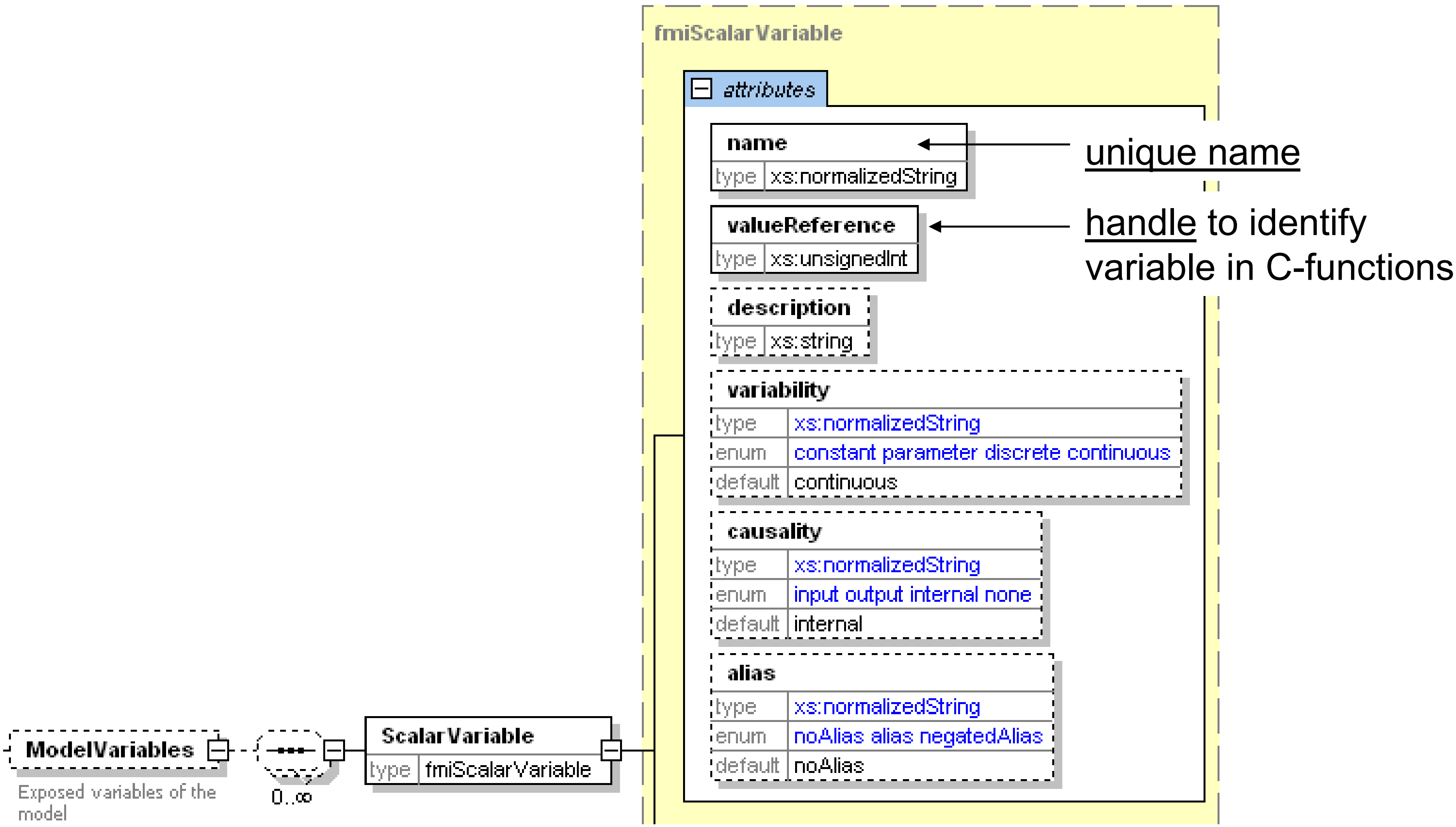


Definition of display units

Definition of type defaults

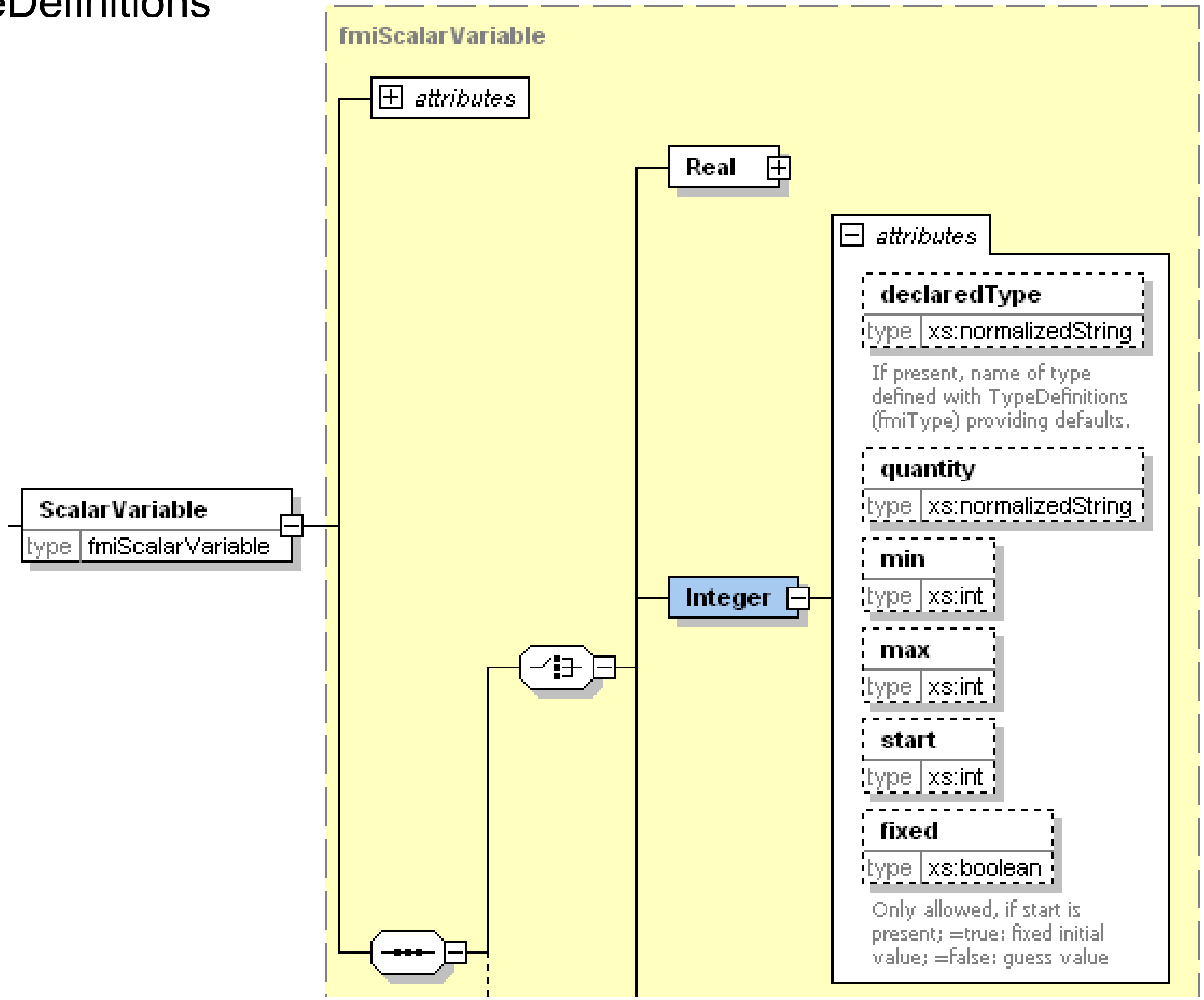
Variable names and attributes

Attributes of ModelVariables



...

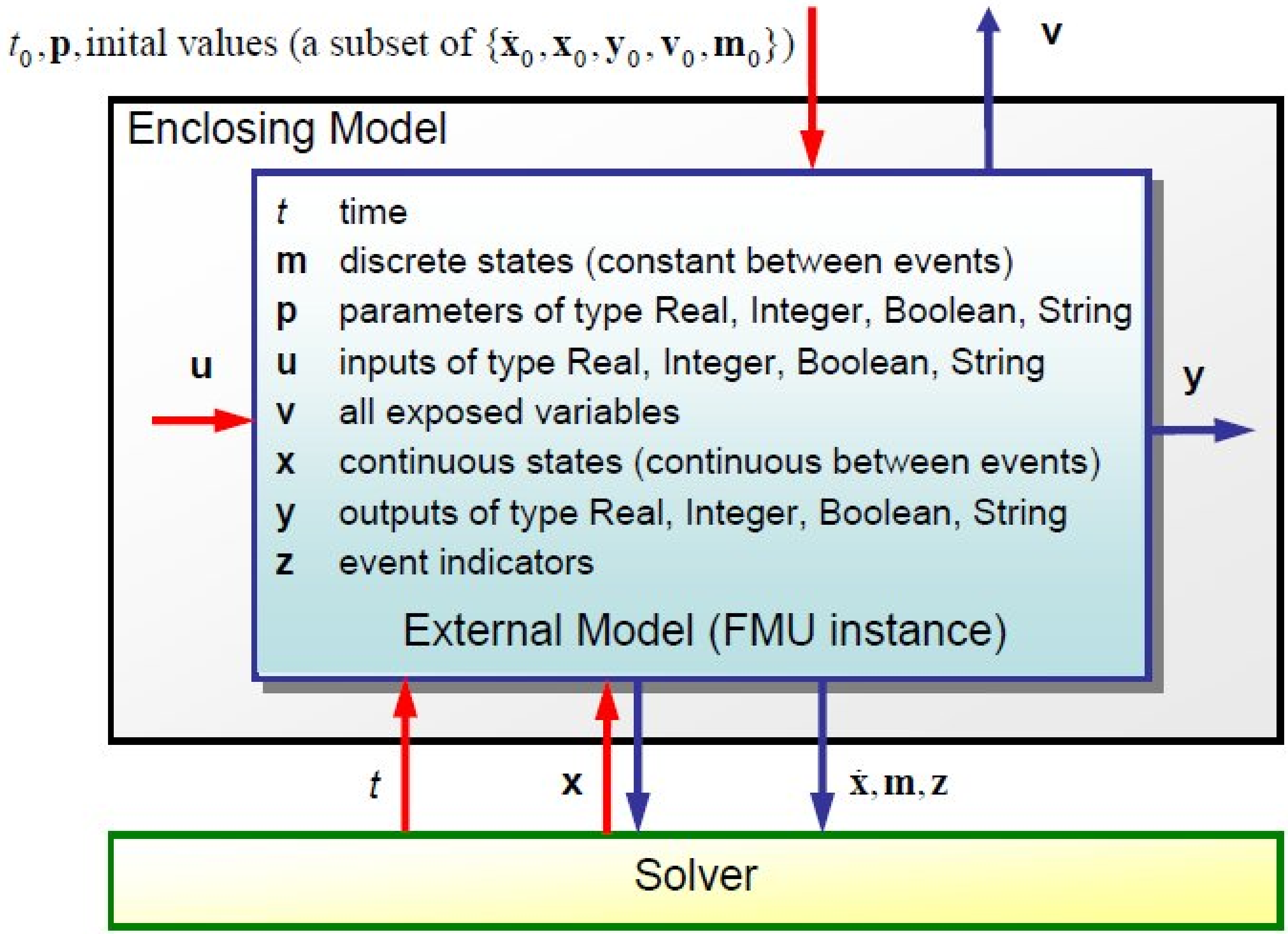
Data types allow to store all (relevant) Modelica attributes.
Defaults from TypeDefinitions



Example

```xml
<?xml version="1.0" encoding="UTF8"?>
<fmiModelDescription
  fmiVersion="1.0"
  modelName="Modelica.Mechanics.Rotational.Examples.Friction"
  modelIdentifier="Modelica_Mechanics_Rotational_Examples_Friction"
  guid="{8c4e810f-3df3-4a00-8276-176fa3c9f9e0}"
  ...
  numberOfContinuousStates="6"
  numberOfEventIndicators="34"/>
  <UnitDefinitions>
    <BaseUnit unit="rad">
      <DisplayUnitDefinition displayUnit="deg" gain="57.2957795130823"/>
    </BaseUnit>
  </UnitDefinitions>
  <TypeDefinitions>
    <Type name="Modelica.SIunits.AngularVelocity">
      <RealType quantity="AngularVelocity" unit="rad/s"/>
    </Type>
  </TypeDefinitions>
  <ModelVariables>
    <ScalarVariable
      name="inertia1.J"
      valueReference="16777217"
      description="Moment of inertia"
      variability="parameter">
      <Real declaredType="Modelica.SIunits.Torque" start="1"/>
    </ScalarVariable>
    ...
  </ModelVariables>
</fmiModelDescription>
```

4. FMI - Model Interface



description	range of t	equation	function names

initialization	$t = t_0$	$(\mathbf{m}, \mathbf{x}, \mathbf{p}, T_{next}) = \mathbf{f}_0(\mathbf{u}, t_0,$ subset of $\{\mathbf{p}, \dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{v}_0, \mathbf{m}_0\})$	`fmiInitialize`
			`fmiGetReal/Integer/Boolean/String`
			`fmiGetContinuousStates`
			`fmiGetNominalContinuousStates`

derivatives $\dot{\mathbf{x}}(t)$	$t_i \le t < t_{i+1}$	$\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	`fmiGetDerivatives`
outputs $\mathbf{y}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{y} = \mathbf{f}_y(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	`fmiGetReal/Integer/Boolean/String`
internal variables $\mathbf{v}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{v} = \mathbf{f}_v(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	`fmiGetReal/Integer/Boolean/String`
event indicators $\mathbf{z}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{z} = \mathbf{f}_z(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	`fmiGetEventIndicators`

event update	$t = t_{i+1}$	$(\mathbf{x}, \mathbf{m}, T_{next}) = \mathbf{f}_m(\mathbf{x}^-, \mathbf{m}^-, \mathbf{u}, \mathbf{p}, t_{i+1})$	`fmiEventUpdate`
			`fmiGetReal/Integer/Boolean/String`
			`fmiGetContinuousStates`
			`fmiGetNominalStates`
			`fmiGetStateValueReferences`

event $t = t_{i+1}$ is triggered if		$t = T_{next}(t_i)$ **or** $\min_{t>t_i} t: \left(z_j(t) > 0\right) \neq \left(z_j(t_i) > 0\right)$ **or** step event	

Example:

```
// Set input arguments
fmiSetTime(m, time);
fmiSetReal(m, id_u1, u1, nu1);
fmiSetContinuousStates(m, x, nx);
```

```
// Get results
fmiGetContinuousStates(m, derx, nx);
fmiGetEventIndicators (m, z, nz);
```

5. Tool Support For FMI

In **Dymola 7.4**

- **Export** of any Modelica model as **FMU** (Functional Mock-up Unit)
- **Import** of a **FMU** into Dymola
  (Modelica model can be translated once-and-for-all to DLL and then reused in a Modelica model as compiled input/output block;
  afterwards code-generation and translation will be much faster for the Modelica models where the DLL is used. Example:
  Large vehicle model and design work is on a controller).
- **Import** of a **Simulink** model as FMU into Dymola
  (based on model code generated by Real-Time Workshop).

FMI support planned for the first half year of 2010

- **SimulationX** (**export** and **import** of **FMUs**)
- **Silver 2.0** (**import** of **FMUs**)
- **SIMPACK** (**import** of **FMUs**, i.e.,
  Modelica models as force elements in high-end multi-body program)

7. Outlook

- "FMI for Model Exchange" released
- "FMI for Co-Simulation" in a good stage. Will be released in first half year.
  (support for: extrapolation/interpolation of interface variables,
  variable communication step-size, re-doing a step
  → step-size control possible).
- "FMI for Model Exchange" will be further developer. A lot of requirements available, such as:
  - Sparse Jacobian
  - Direct support for arrays and records in xml schema
  - Improved sample time definition (for embedded systems)
  - Online changeable parameters
  - Saving/restoring model state
  - ...

8. Acknowledgments

FMI initiated : Volker May (Daimler AG)
Head of FMI development : Dietmar Neumerkel (Daimler AG)
Head of FMI-for-Model-Exchange: Martin Otter (DLR-RM)

FMI-for-Model-Exchange Core-Design by:
Torsten Blochwitz (ITI)
Hilding Elmqvist (Dassault Systèmes -Dynasim)
Andreas Junghanns (QTronic)
Jakob Mauss (QTronic)
Hans Olsson (Dassault Systèmes -Dynasim)
Martin Otter (DLR-RM)

Other MODELISAR contributors:
Ingrid Bausch-Gall, Bausch-Gall GmbH
Alex Eichberger, SIMPACK AG
Rainer Keppler, SIMPACK AG
Gerd Kurzbach, ITI GmbH
Carsten Kübler, TWT
Johannes Mezger, TWT
Thomas Neidhold, ITI GmbH
Dietmar Neumerkel, Daimler AG
Peter Nilsson, Dassault Systèmes-Dynasim
Antoine Viel, LMS International
Daniel Weil, Dassault Systèmes

Prototypes for FMI evaluation:
Dymola by Peter Nilsson, Sven Erik Mattsson, Carl Fredrik Abelson, Dan Henriksson (Dassault Systèmes, Dynasim)
JModelica.org by Tove Bergdahl (Modelon)
Silver by Andreas Junghanns, Jakob Mauss (QTronic)

Other contributors:
Johan Akesson, Lund University
Joel Andersson, KU Leuven
Roberto Parrotto, Politecnico di Milano

Partially funded by: BMBF, VINNOVA, DGCIS, organized by ITEA2

Outlook

- Modelica scope extended to code generation for embedded targets
- Designed to be flexible enough to allow reuse of logical model for many X-In-the-Loop scenarios
- Integrating the solutions
- Utilize in larger projects