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
Content

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

Hybrid Differential Algebraic Equations (DAE)

- Algebraic equations
- Ordinary differential equations
- Event handling and sampled control

\[ 0 = f \left( \frac{dx}{dt}, x, w, p, u, y \right) \]

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

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

53’rd Modelica Design Meeting at Dynasim
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 Modelica.SIunits.Mass;
  equation
    \[ \frac{d}{dt}(m \cdot v) = \sum F \]
end Mass;

\[
\begin{align*}
  r_0 &= \text{frame}_0 \cdot r_0 \\
  \text{Frame}_a.R &= \text{Frame}_a.Q \left\{ \text{Frames} \right\} \cdot \text{Quaternions} \cdot \text{angularVelocity} \cdot \text{angularVelocity}^2 \left( \text{Frame}_a, \frac{d}{dt} \right) \\
  \{ \} &= \text{Frames} \cdot \text{Quaternions} \cdot \text{orientation} \cdot \text{constraint} \left( \text{Frame}_a \right) \\
  \{ \} &= \text{world} \cdot \text{gravity} \cdot \text{acceleration} \left( \text{Frame}_a, r_0 \right) + \text{Frame}_a \cdot \text{resolve} \left( \text{Frame}_a, R, \text{cm} \right) \\
  \dot{v}_0 &= \frac{d}{dt} \text{frame}_0 \cdot \text{r}_0 \\
  \dot{x}_0 &= \frac{d}{dt} x_0 \\
  \dot{w}_0 &= \frac{d}{dt} w_0 \\
  \text{Frame}_a.R &= \text{Frame}_a.Q \cdot \text{frames} \cdot \text{resolve} \cdot \left( \text{Frame}_a, R, \text{cm} \right) + \text{frame}_a \cdot \text{cm} \cdot \dot{w}_a + \text{frame}_a \cdot \text{cm} \\
  \text{Frame}_a.t &= \dot{x}_a + \dot{w}_a \cdot \dot{x}_a + r_{\text{cm}} \cdot \text{frame}_a.f
\end{align*}
\]
Model Knowledge

Thermo and fluid dynamics

```
model SimOrinNoStates "Simple orifice model for turbulent flow."
  extends HyLib.Restrictions.Basic.OrificePartial;
  parameter Model
    parameter Parameter.Rea Modelica.Stm
      q_unsigned = d
      q = noEvent if
      end SimOrinNoStates;

q_unsigned = a

q = n
```

```
model SimOrinNoStates "Simple orifice model for turbulent flow."
  extends HyLib.Restrictions.Basic.OrificePartial;
  parameter Model
    parameter Modelica.Rea
      q_unsigned = d
      q = noEvent if
      end SimOrinNoStates;

q_unsigned = a

q = n
```
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)
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

- Model Parameters
- Experimental Data

Graphical Editor

Plot and Animation

Symbolic Kernel

Graphical Editor

Dymola Kernel

Simulation

Optimization

Modelica Libraries

User Models

Modeling

Visualization and Analysis

- Scripting
  - Modelica
  - C Functions
  - LAPACK

Plot and Animation

Reporting

External Graphics (bitmap)

CAD (DXF, STL)

dxSPACE

xPC

Simulink

MATLAB

Model doc. and Experiment log (HTML, VRML, PNG, …)
Virtual testing of system behavior
Modelica for Embedded Systems

H. Elmqvist\textsuperscript{1}, M. Otter\textsuperscript{2}, D. Henriksson\textsuperscript{1}, B. Thiele\textsuperscript{2}, S.E. Mattsson\textsuperscript{1}

\textsuperscript{1}\textit{Dassault Systèmes (Dynasim), Lund, Sweden}
\textsuperscript{2}\textit{DLR - Institut for Robotics and Mechatronics, Oberpfaffenhofen, Germany}
Outline

1. Overview
2. Basic Idea
3. Modelica_EmbeddedSystems Library
4. Modelica Language Extensions
5. Outlook
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)
Complex control systems in cars

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

But:
The logical model needs to be partitioned for different applications
Controlled Systems

- System Model
- Flat Logical Model
- Controlled Subsystem
- Physical Subsystem
- Control Subsystem
X-in-the-loop Simulation

- Logical Model
- Model-in-the-Loop
- Software-in-the-Loop
- Physical model
- Control model
- Interface

Hardware-in-the-Loop

Rapid Prototyping

Production Code

Slide 28
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

physical model without controller

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

Inherit from logical model and define mapping to target system (no change of logical system)

local controllers are in corresponding submodel

global controller (e.g. path planning)
Important:
Mark submodels to be specially handled
2. Basic Idea

1. Mark boundaries → Logical model is partitioned into 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, ...; asynchronous equations)
3. Modelica_EmbeddedSystems Library

Free library to define the mapping for embedded systems

- Replaceable blocks for direct communication (for simulation)
- Signal to hardware driver
- Signal from hardware driver

Actual blocks for communication

Defining task, subtask, target
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.

- Direct Communication (same model/subtask)
- Communication between two subtasks
- Communication between two tasks
- Communication to a port
- Communication from a port

**Communication between input and output**

- **communicationType**: Simulated DA or AD converter block
- **fromInputToOutput**: Simulated DA or AD converter block
- **toPort**: Periodic subtask sample PeriodFactor if useToPort
- **fromPort**: Start of subtask sample PeriodFactor if useFromPort

**Sampling and other configurations of subtask to which input and/or output belongs (if de-activated, the information is defined somewhere else)**

- **defineInSubtask**
- **defineOutSubtask**
- **inSubtask**
- **outSubtask**

**Sampling and noise**

- **sampled**: True
- **baseSampleRate**: 0.002
- **sampleFactor**: 1
- **computationalDelay**: 1
- **communicationDelay**: 0
- **noisy**: False
- **noise**: redeclare Modelica

**Limiting and quantization**

- **limited**: True
- **quantized**: True
- **max**: 10
- **min**: -10
- **bits**: 8

Here:
simulated communication with effects like delay, noise, limitation, quantization, ....
Sampled Controller
Sampled Controller with Delay
Sampled Controller with Input Noise
Sampled Controller with Output Quantization
here:

communication between two tasks
input: Writes to hardware (comedi driver)
output: 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)

<table>
<thead>
<tr>
<th>inTask</th>
<th>controllerTask</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>&quot;slowSampler&quot;</td>
</tr>
<tr>
<td>samplingType</td>
<td>Types.SamplingType.Periodic</td>
</tr>
<tr>
<td>samplePeriodFactor</td>
<td>5</td>
</tr>
<tr>
<td>sampleOffsetFactor</td>
<td>0</td>
</tr>
<tr>
<td>integrationMethod</td>
<td>Types.IntegrationMethod.FixedStepTrapezoid</td>
</tr>
<tr>
<td>FixedStepSize</td>
<td>samplePeriodFactor*inTask.sampleBasePeriod</td>
</tr>
</tbody>
</table>
4. Modelica Language Extensions

Modelica extensions developed for the mapping concept:

\[ u = \text{Subtask}.\text{decouple}(y); \]
(a) \( u = y \); (b) \( u \) and \( y \) are in different subtasks.

\[ \text{parameter} \ \text{Modelica.SIunits.Time} \ \text{sampleBasePeriod}; \]
\[ \text{RealInput} \ u \ \text{annotation}(\text{mapping}(\]
  \[ \text{target} \ (\text{identifier} = "abc"), \]
  \[ \text{task} \ (\text{identifier} = "slowTask"," \]
  \[ \text{sampleBasePeriod} = \text{sampleBasePeriod}), \]
  \[ \text{subtask} \ (\text{identifier} = "reference"," \]
  \[ \text{samplingType=}\text{Subtask.SamplingType.Periodic})) \]

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 \emph{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

```c
/* 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

```c
/* 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 stimuli

2. For real-time simulators including prototype hardware components (e.g. Force-feedback steering wheel for Steer-by-Wire application)
Use cases in the simulator development process

Non-interactive simulation

Interactive model stimulus 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 measurement device interface

Windows device support
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

Initial step indicator

Steps

Transition

Delayed Transition
Steps & Transitions are implemented in pure Modelica

```modelica
equation
    // Set active state
    import_fire = Modelica.StateGraph2.Blocks.BooleanFunctions.anyTrue(inPort.fire);
    output_fire = Modelica.StateGraph2.Blocks.BooleanFunctions.anyTrue(outPort.fire);
    newActive = if node.resume then oldActive else
                import_fire or (active and not output_fire) and not node.suspend;
    active = pre(newActive);

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

    // 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;
```
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
The Parallel Component cont’d

Inport

Suspend port

Resume port

Outport
Hierarchical Hybrid System – Tank example
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 – Goals
2. FMI - Distribution of Model
3. FMI - Model Description Schema
4. FMI - Model Interface
5. Tool Support for FMI
6. Comparison with SIMULINK S-Function Interface
7. Outlook
8. Acknowledgements
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
    - <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

- unique name
- handle to identify variable in C-functions
Data types allow to store all (relevant) Modelica attributes.
Defaults from TypeDefinitions
Example

```xml
<?xml version="1.0" encoding="UTF8"?>
<fmiModelDescription
  fmiVersion="1.0"
  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

$t_0, p, \text{iniital values (a subset of } \{\dot{x}_0, x_0, y_0, v_0, m_0\})$
<table>
<thead>
<tr>
<th>description</th>
<th>range of t</th>
<th>equation</th>
<th>function names</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialization</td>
<td>$t = t_0$</td>
<td>$(m, x, p, T_{next}) = f_0(u, t_0,$</td>
<td>fmiInitialize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subset of ${p, \dot{x}_0, x_0, y_0, v_0, m_0}$)</td>
<td>fmiGetReal/Integer/Boolean/String</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetContinuousStates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetNominalContinuousStates</td>
</tr>
<tr>
<td>derivatives $\dot{x}(t)$</td>
<td>$t_i \leq t &lt; t_{i+1}$</td>
<td>$\dot{x} = f_x(x, m, u, p, t)$</td>
<td>fmiGetDerivatives</td>
</tr>
<tr>
<td>outputs $y(t)$</td>
<td>$t_i \leq t &lt; t_{i+1}$</td>
<td>$y = f_y(x, m, u, p, t)$</td>
<td>fmiGetReal/Integer/Boolean/String</td>
</tr>
<tr>
<td>internal variables $v(t)$</td>
<td>$t_i \leq t &lt; t_{i+1}$</td>
<td>$v = f_v(x, m, u, p, t)$</td>
<td>fmiGetReal/Integer/Boolean/String</td>
</tr>
<tr>
<td>event indicators $z(t)$</td>
<td>$t_i \leq t &lt; t_{i+1}$</td>
<td>$z = f_z(x, m, u, p, t)$</td>
<td>fmiGetEventIndicators</td>
</tr>
<tr>
<td>event update</td>
<td>$t = t_{i+1}$</td>
<td>$(x, m, T_{next}) = f_m(x^{-}, m^{-}, u, p, t_{i+1})$</td>
<td>fmiEventUpdate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetReal/Integer/Boolean/String</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetContinuousStates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetNominalStates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fmiGetStateValueReferences</td>
</tr>
</tbody>
</table>

event $t = t_{i+1}$ is triggered if

$t = T_{next}(t_i)$ or $\min_{t > t_i} t : (z_j(t) > 0) \neq (z_j(t_i) > 0)$ or step event

Example:

```c
// 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
  » ...

Functional Mockup Interface - Overview > Jan. 25 2010 > Slide 95
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