Functional Mockup Interface – Overview

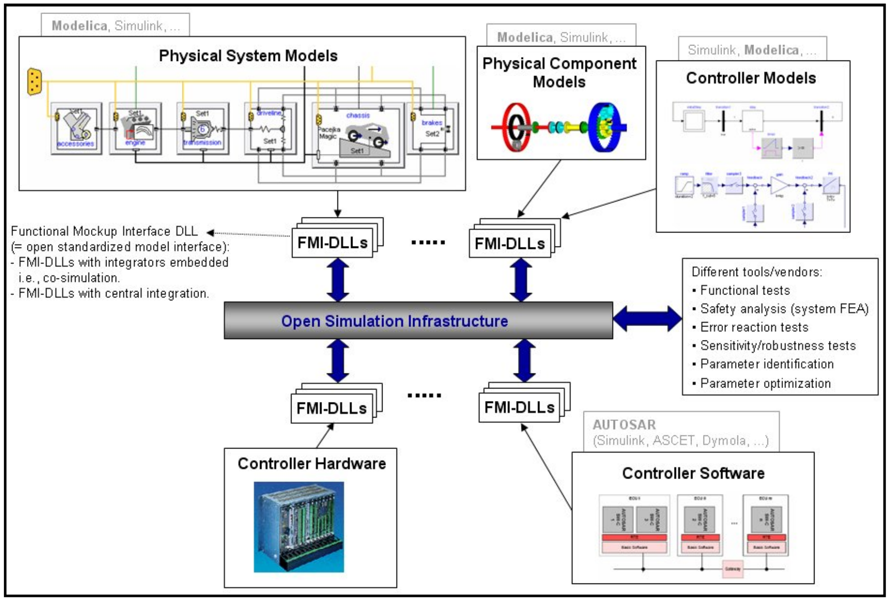
Martin Otter (DLR-RM)
Torsten Blochwitz (ITI)
Hilding Elmqvist (Dassault Systèmes – Dynasim)
Andreas Junghanns (QTronic)
Jakob Mauss (QTronic)
Hans Olsson (Dassault Systèmes – Dynasim)

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 (no resources yet planned)

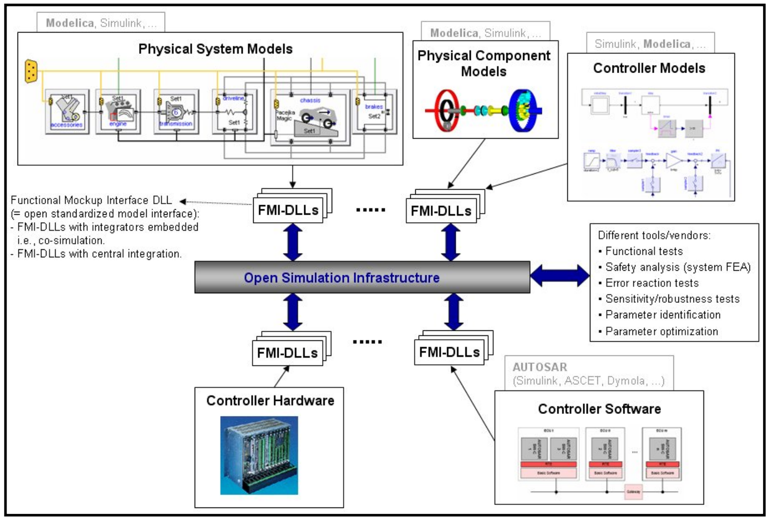
Open Standard

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.

Extension for co-simulation under development (Uni Halle, ITI, Fraunhofer)



2. FMI - Distribution of Model

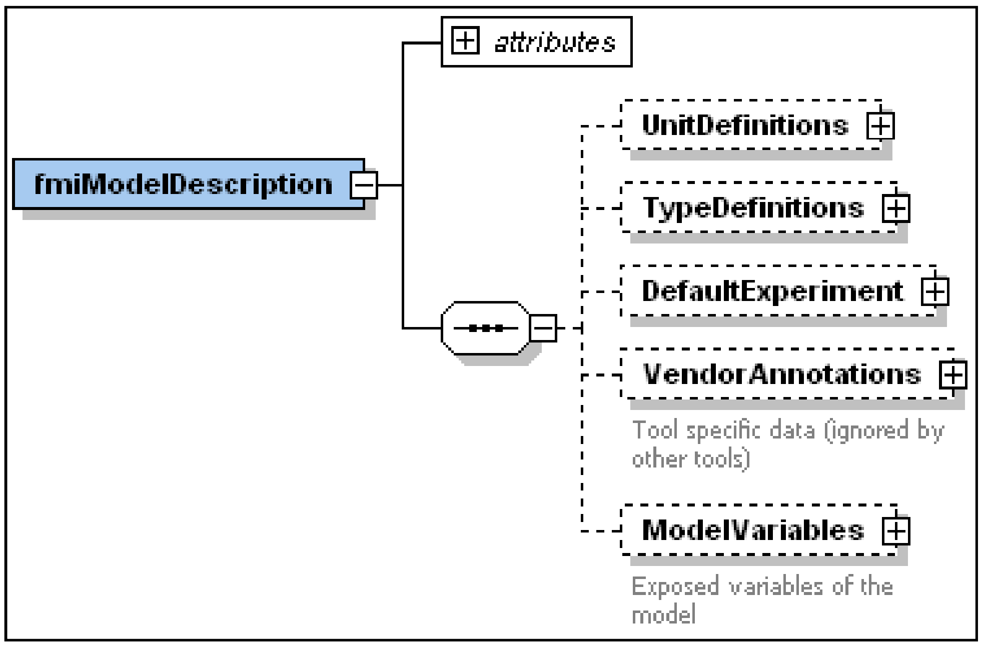
A model is distributed as one zip-file with extension "**.fmu**". Content:

```
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)

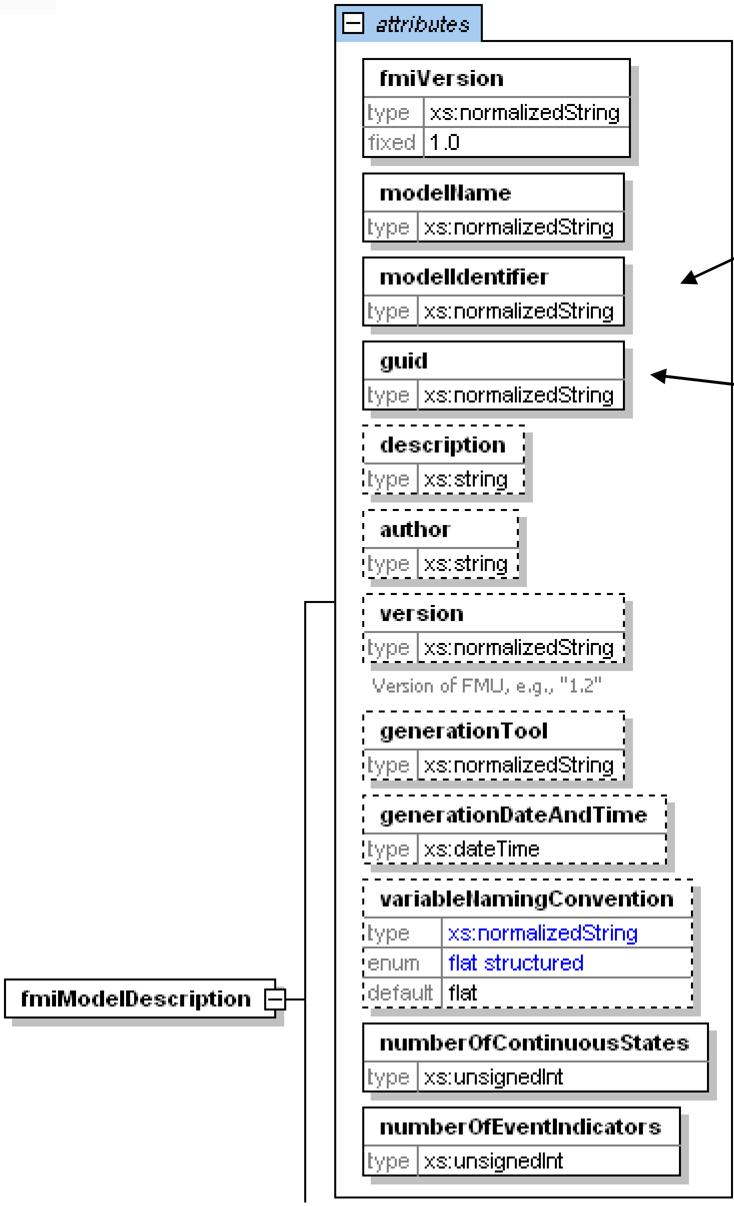
Advantage:
Complex data structures give still simple interface, and tool can use its favorite programming language for reading (e.g., C++, C#, Java).



Definition of display units

Definition of type defaults

Variable names and attributes



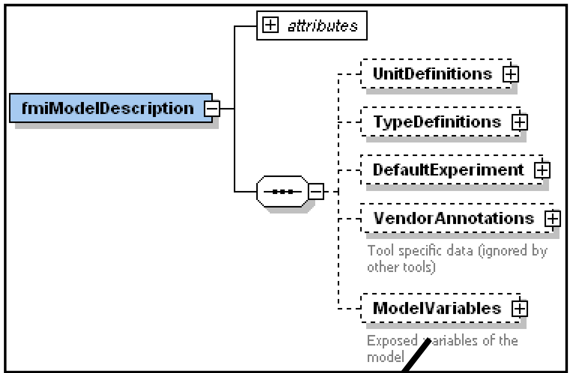
Model attributes. Most important

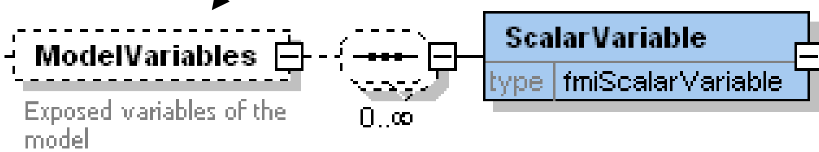
modelIndentifier is a C-name that is used as prefix for the C-functions (model interface)

guid is a globally unique identifier ("fingerprint" of all releveant information in the xml file) that is also stored in the C-functions to gurantee consisteny

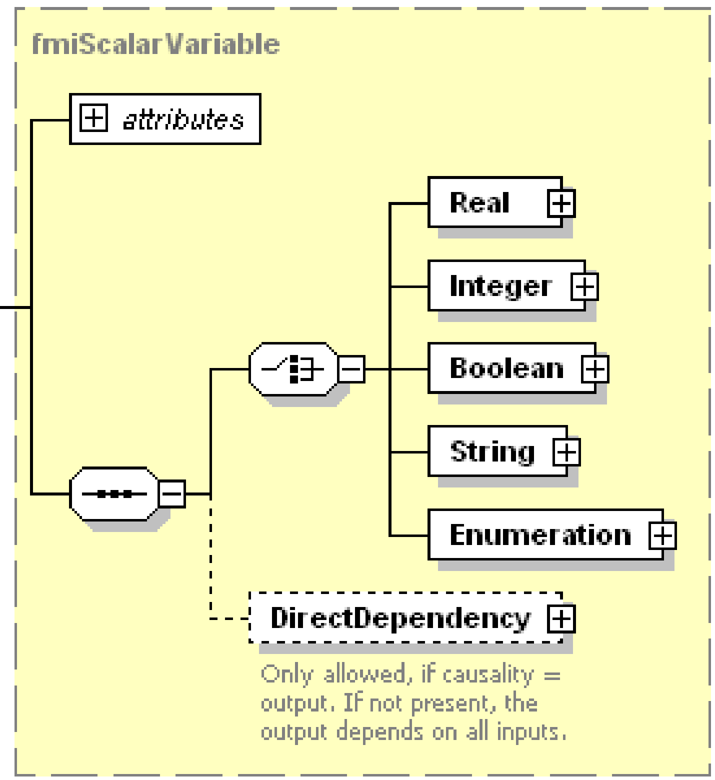
Number of continuous states and of event indicators; numbers are fixed (meaning of states can change dynamically during simulation)

ModelVariables



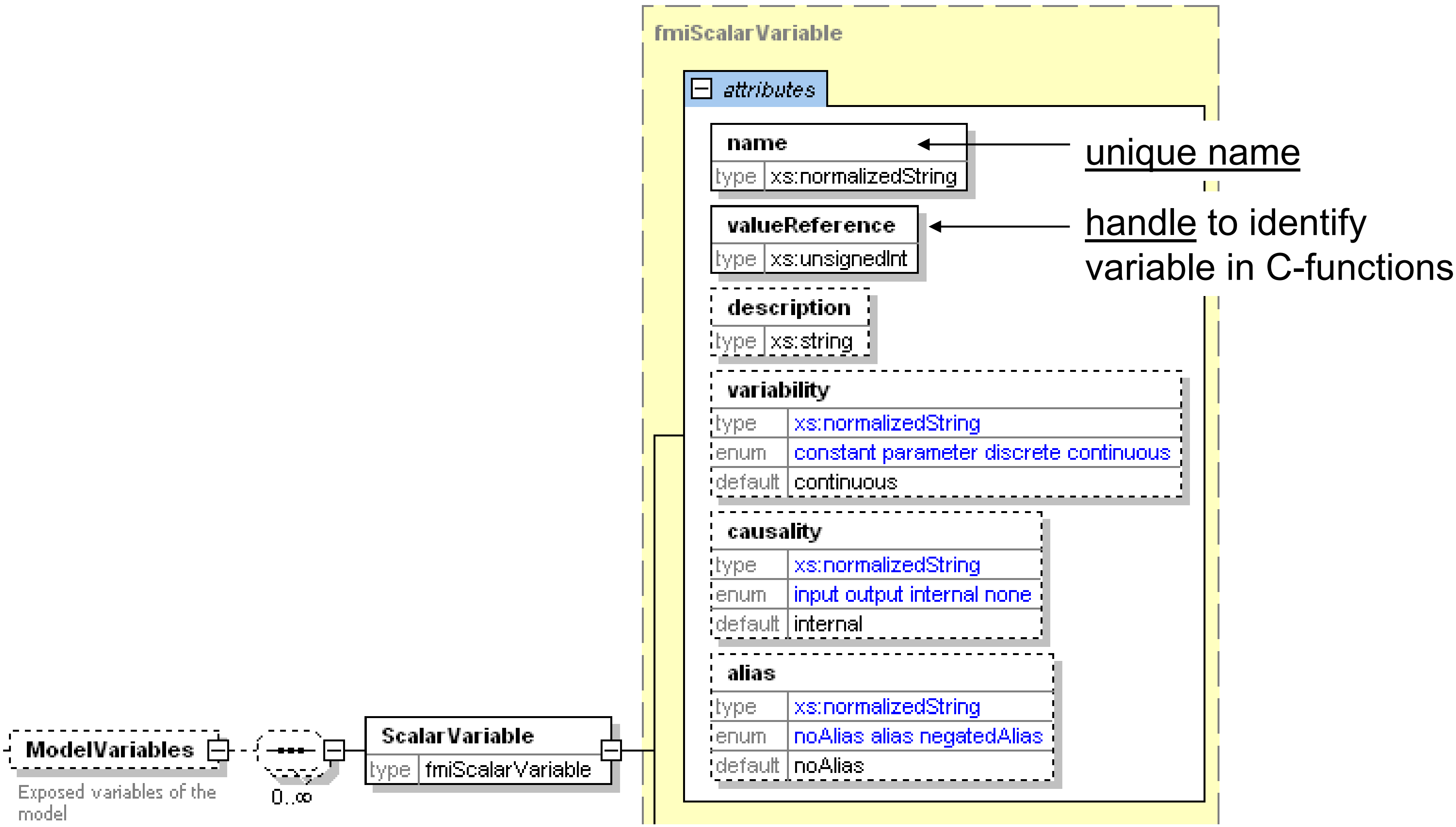


ordered set of scalar variables (arrays, records, etc. must be mapped to scalars when generating code).



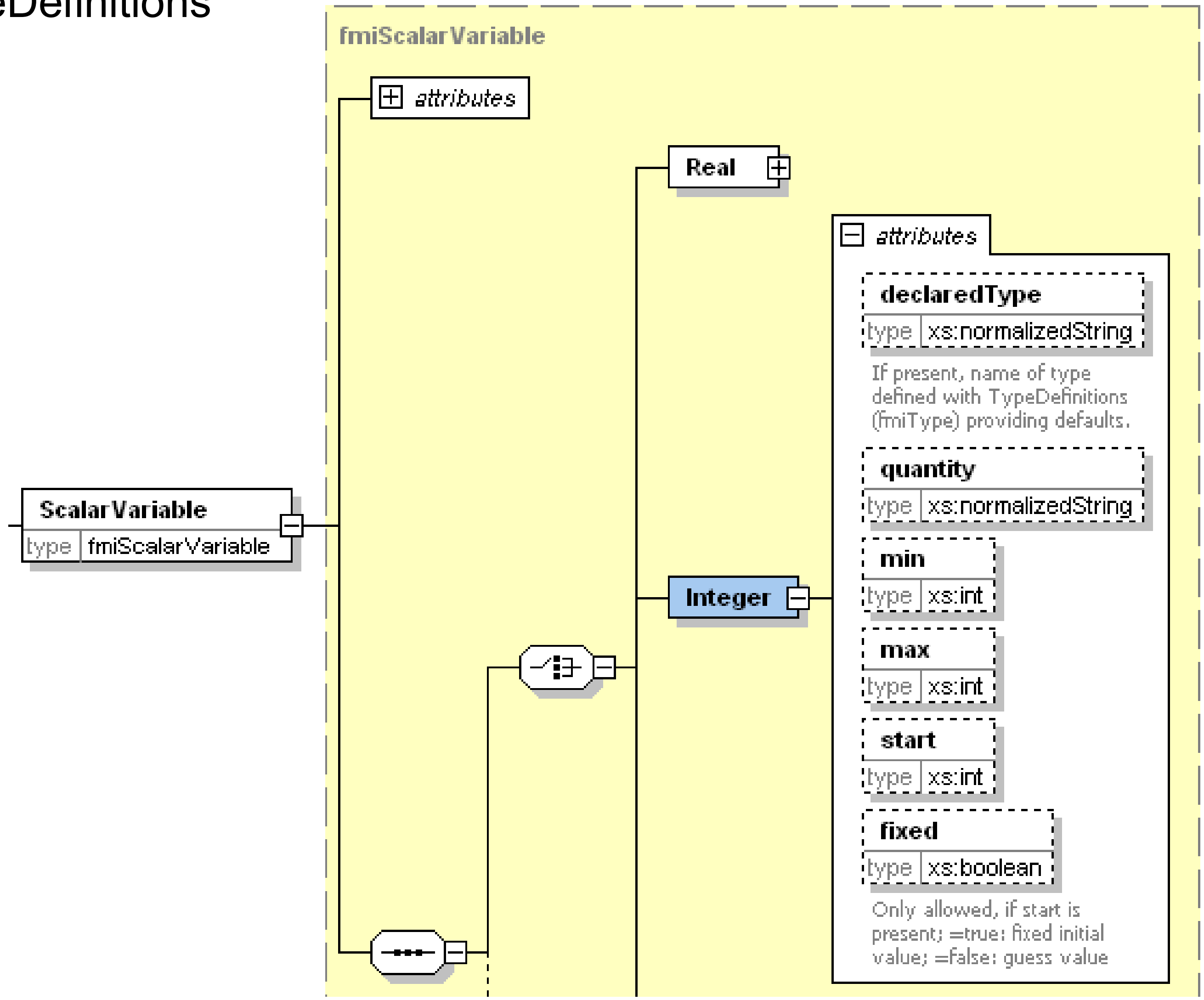
data types

Attributes of ModelVariables



...

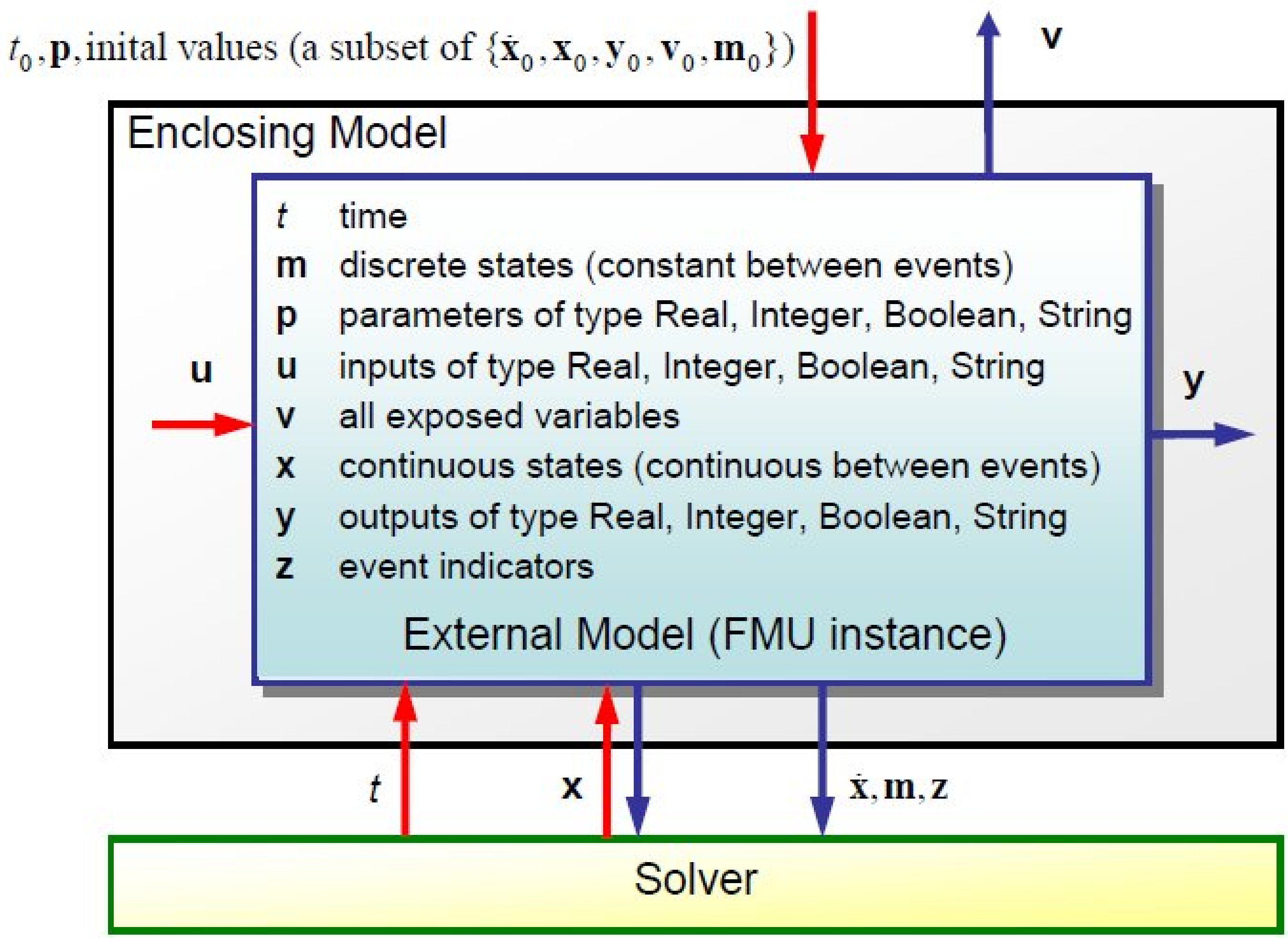
Data types allow to store all (relevant) Modelica attributes.
Defaults from TypeDefinitions



Example

```
<?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	$t = t_0$	$(\mathbf{m}, \mathbf{x}, \mathbf{p}, T_{next}) = \mathbf{f}_0(\mathbf{u}, t_0, \text{subset of } \{\mathbf{p}, \dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{v}_0, \mathbf{m}_0\})$	`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** $\min_{t>t_i} t: \left(z_j(t) > 0\right) \ne \left(z_j(t_i) > 0\right)$ **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);
```

Caching for efficient model evaluation

Model equations:

input: x_1, x_2

output: $y, \dot{x}_1, \dot{x}_2$

$$0 = f_1(\dot{x}_1, x_1, y)$$
$$0 = f_2(\dot{x}_1, x_1, y)$$
$$\dot{x}_2 = f_3(x_1, x_2, y)$$

Unefficient solution (algebraic system of equations is solved twice)

$\mathbf{y} = \mathbf{f}_y(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	$0 = f_1(\dot{x}_1, x_1, y)$ $0 = f_2(\dot{x}_1, x_1, y)$ // solve for $y, \dot{x}_1$ and return y
$\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$	$0 = f_1(\dot{x}_1, x_1, y)$ $0 = f_2(\dot{x}_1, x_1, y)$ // solve for $y, \dot{x}_1, \dot{x}_2$ and return $\dot{x}_1, \dot{x}_2$ $\dot{x}_2 = f_3(x_1, x_2, y)$

Efficient solution with caching

$\mathbf{y} = \mathbf{f}_y(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
→ call $\mathbf{f}_{int}$(...,compute_y)

$\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
→ call $\mathbf{f}_{int}$(...,compute_xd)

```
function fmiSetContinuousStates(..)
  ...
  y_computed  = false
  xd_computed = false


function f_int
  ....
  if (compute_y or compute_xd)
    and not y_computed then
      0 = f1(xd1, x1, y)
                          // solve for y, xd1
      0 = f2(xd1, x1, y)
    y_computed = true;
  end if;

  if compute_xd then
     xd2 = f3(x1, x2, y)  // compute xd2
     xd_computed = true
  end if;
```

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)

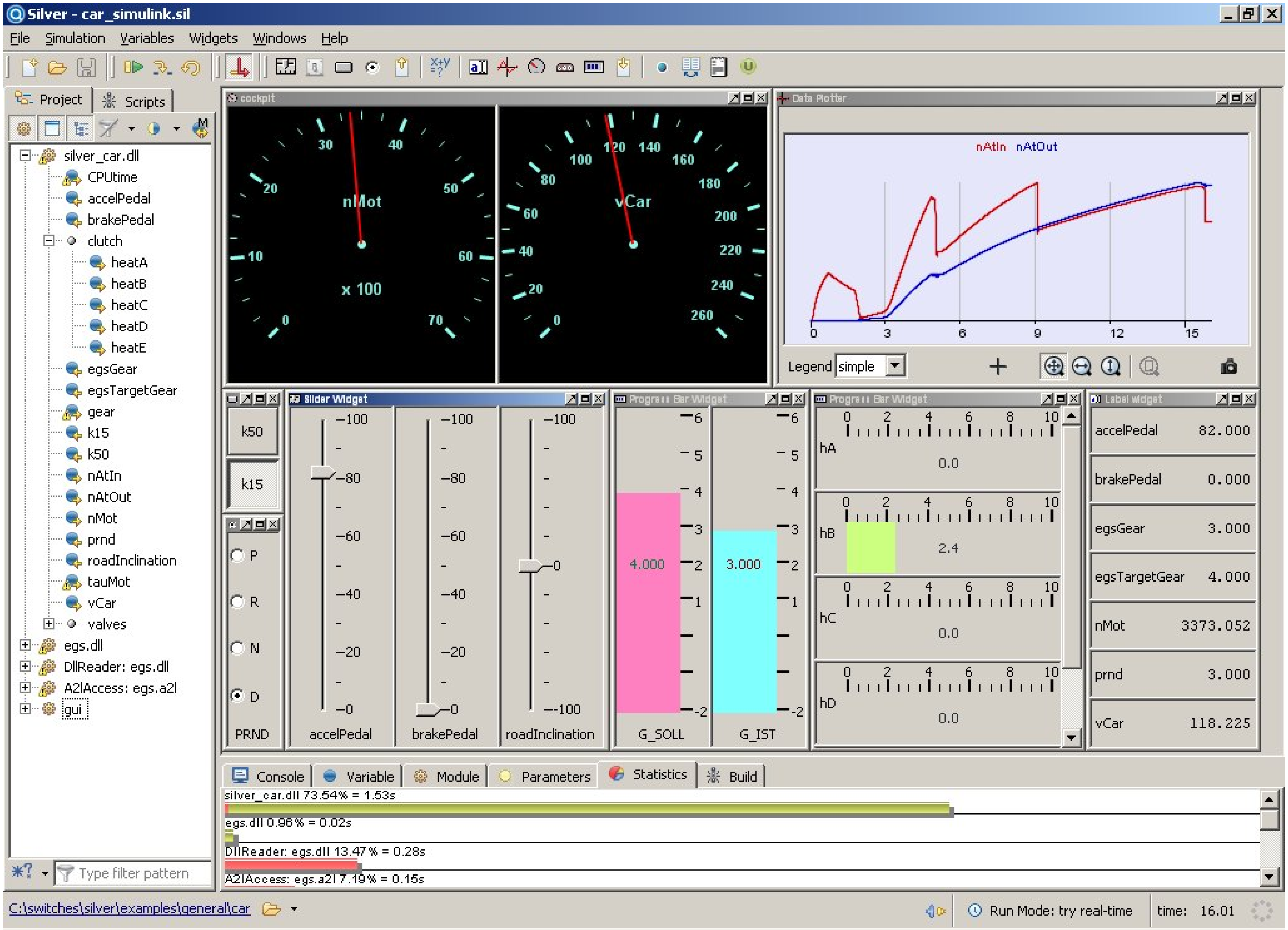
SimulationX

- **Export** of **FMUs** (March 2010)
- **Export** for FMI-for-**Co-Simulation** (April 2010)
- **Import** of **FMUs** (June 2010)

Silver 2.0 (March 2010)

- **Import** of **FMUs**
- **Connecting FMUs** in Silver (is treated as DAE)
- Together will all other Silver features, e.g., submodels can be provided in other formats:
  - Software of Electronic Control Units
  - Models of other tools (Dymola 6.x, 7.x, SimulationX, ...)
  - Configurable GUI to control inputs and outputs.
  - Automatic tests

Silver



courtesy: QTronic

6. Comparison with SIMULINK S-Function Interface

- S-function DLL is simulator-specific:
  Since model data structure is a "secret" of the simulation environment. E.g. for 3 simulation environments → 3 DLLs of the same model
  DLL needs to be newly generated for every new version of the S-Function header file (every SIMULINK version).
  FMI: Model DLL is specific to modeling environment, i.e., the same DLL can be used for all simulators on the same platform.
- S-function not suited for embedded systems, due to large memory overhead since all information of a model is stored in the Model DLL (therefore separate code generation for embedded systems via Realtime Workshop)
  FMI: Only the minimum necessary part is stored in C source code or in Model DLL. All information not needed for execution, is provided in an XML file (which is needed on host, but not on target microprocessor)
- S-function has very complex definition (> 100 C-functions/macros)
  Generating S-function is fine. However, there is no simulator that can import all S-function models (with exception of SIMULINK).
  FMI: Simple definition (20 C-functions, no macros, XML schema file)
- S-function proprietary format, gives legal problems if used in other simulators
  FMI: Wikipedia license for specification, BSD license for schema/header

Technical issues that are missing in S-Function interface and are available in FMI:

- Reliable **state event** handling
- **Event iteration** over simulation model (not only component model)
- Request from submodel to **reduce step-size**
  (for non-linear equations in model that do not converge)
- **Dynamic selection of states**
- **Alias** variables (FMI: alias variables are marked; need to be stored only once, not several times).
- **Caching** of computed results
  (FMI: more efficient solution)

7. Outlook

- "FMI for Model Exchange" shall be released this week
  (technical specification finalized; some discussion about precise license text)
- "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