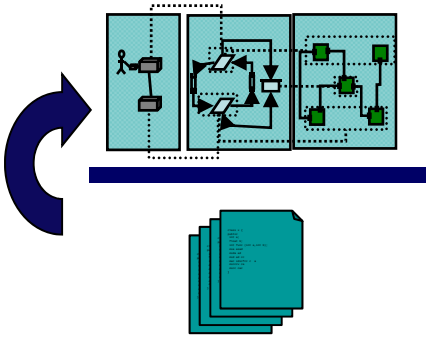
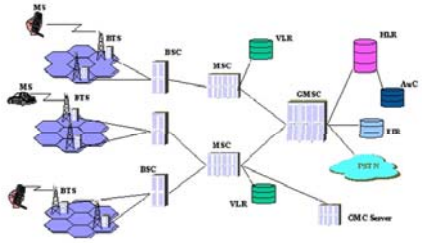
Functional Mockup Interface (FMI)
A General Standard for Model Exchange and Simulator Coupling

Linköping University
2011-02-10

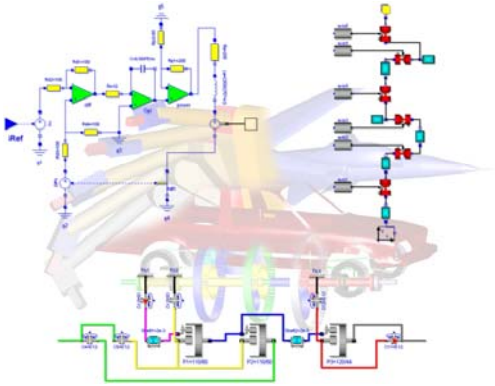
Peter Fritzson





Acknowledgements:

MODELISAR Project is developing FMI. Most slides in this presentation by Martin Otter, together with Torsten Blochwitz, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Hans Olsson
Also some slides by Azam Zia



FMI – Motivation 1

- Need to SOLVE **large integrated** modeling and simulation engineering **problems**
- Hundreds of simulation tools, different model formats
- Need to **integrate** multiple **tools** and **formats** for simulation

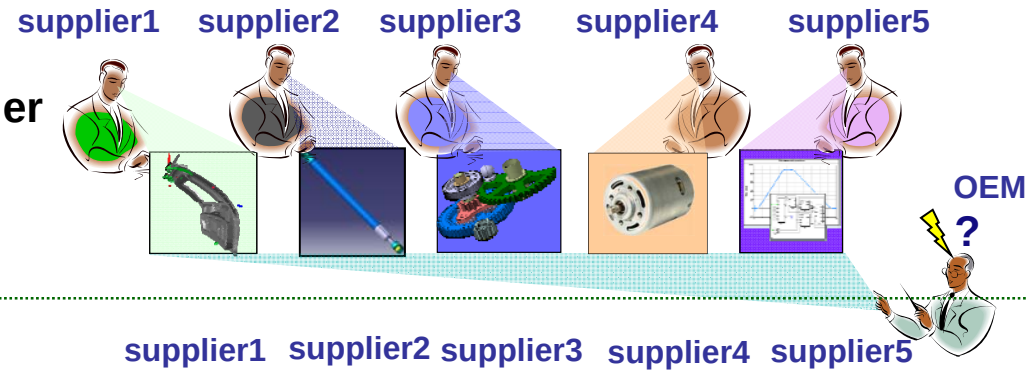
- Two main approaches:
  - 1. **Export** models from some tools, **import** into other tools for **simulation**
  - 2. **Co-simulation** of models in different tools
- Export/Import Model Format: **Functional Mockup Unit (FMU)**

- Solution: Functional Mockup Interface (FMI) standard **www.functional-mockup-interface.org**

FMI – Motivation 2

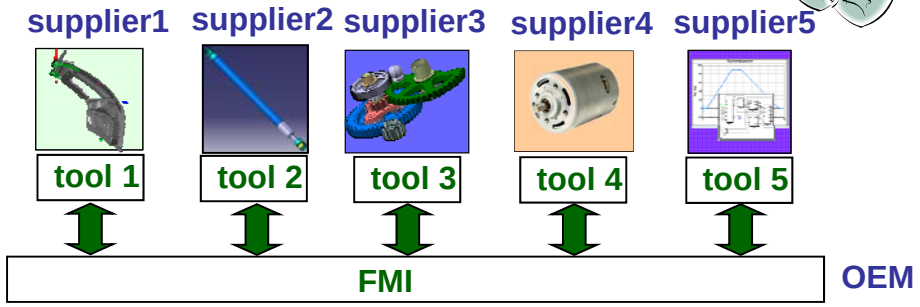
Problems / Needs

- Component development by supplier
- Integration by OEM
- Many different simulation tools



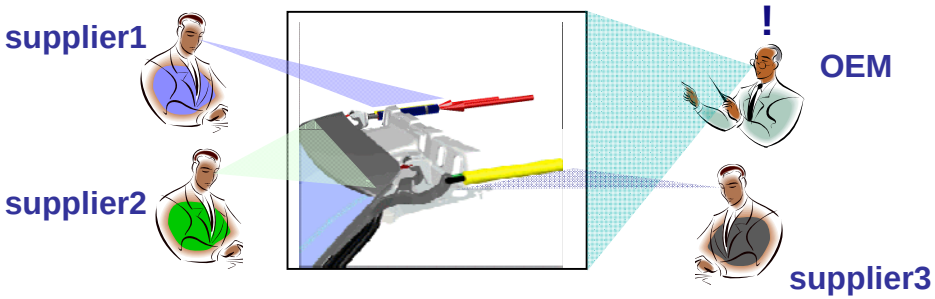
Solution

- Reuse of supplier models by OEM:
  - DLL (model import) and/or
  - Tool coupling (co-simulation)
- Protection of model IP of supplier



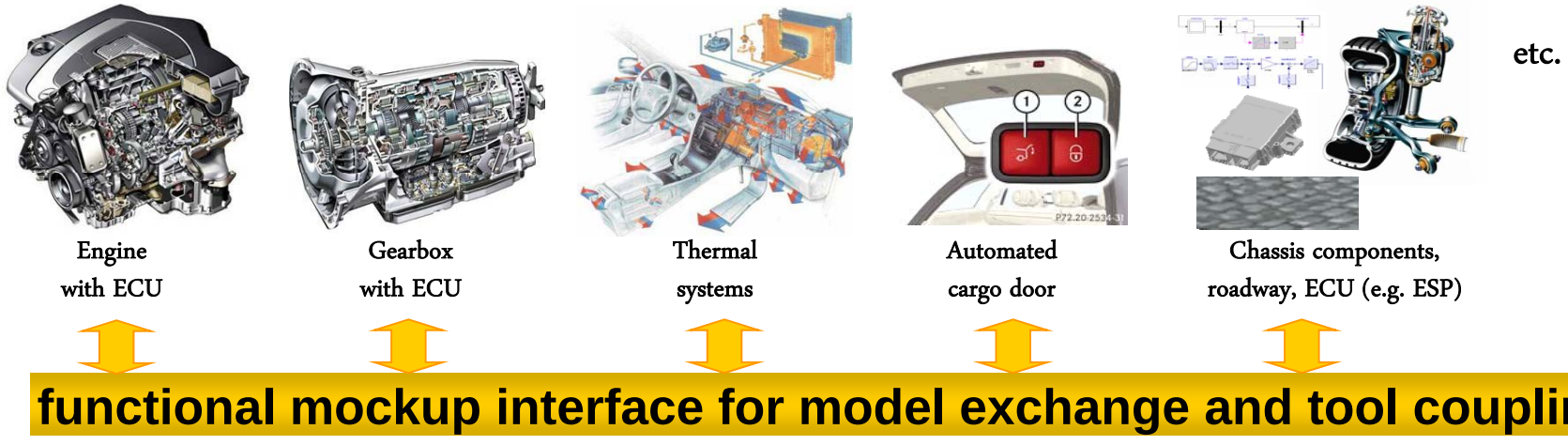
Added Value

- Early validation of design
- Increased process efficiency and quality



slide from Nick Suyam, Daimler (adapted)

Functional Mock-up Interface (FMI) – Overview



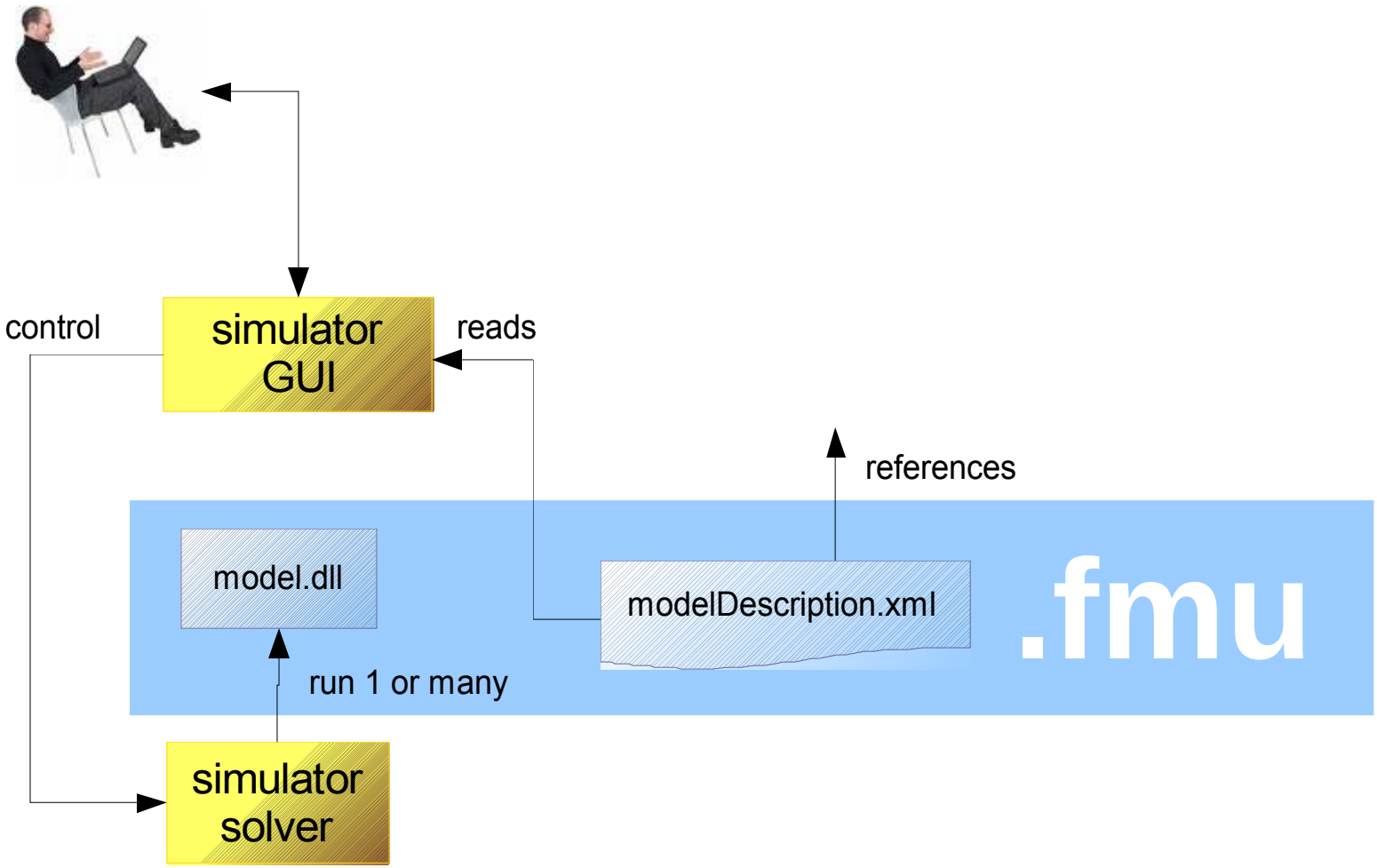
courtesy Daimler

The FMI development is part of the ITEA2 MODELISAR project.

- FMI development is initiated, organized and headed by **Daimler** AG
- Improved Software/Model/Hardware-in-the-Loop Simulation, of **physical** models and of **AUTOSAR** controller models from **different vendors** for automotive applications with **different levels of detail**.
- **Open Standard**
- **14 automotive use cases** for evaluation in MODELISAR
- **> 10 tools** plan to support it (mostly within MODELISAR)

Simulator with GUI and Solver

Executing Imported Model = FMU (Functional Mockup Unit)



Peter Fritzson presenter. Courtesy MODELISAR project, Martin Otter et al

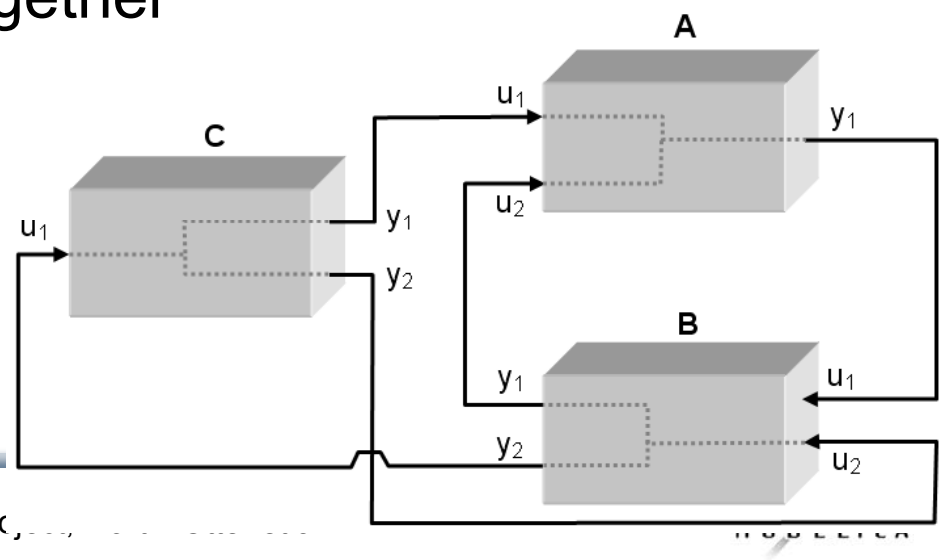
Exported Model in (Functional Mockup Unit) FMU Form

A model is distributed in one zip-file that contains several files:

- XML file of model variable information.
- C file with model equations converted into causal form
- Further data , documentation, maps, icon

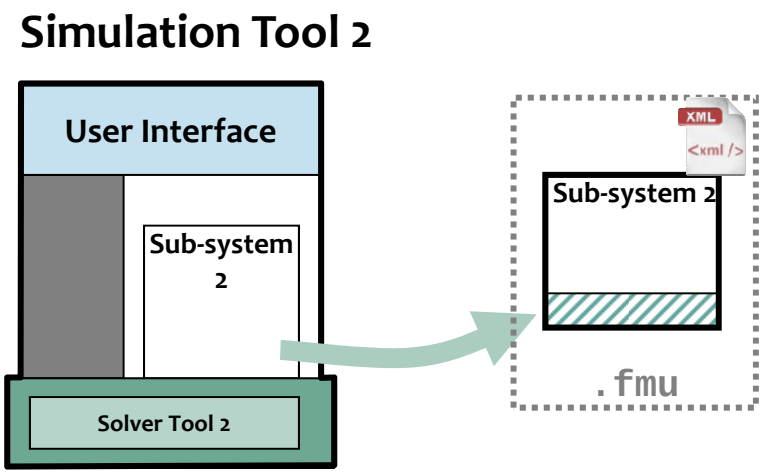
FMI for Model Exchange – Functional Mockup Units

- Import and export of input/output blocks – **Functional Mock-Up Units – FMU**s
- described by
  - differential-, algebraic-, discrete equations,
  - with time-, state, and step-events
- An FMU can be large (e.g. 100 000 variables)
- An FMU can be used in an embedded system (small overhead)
- FMUs can be connected together

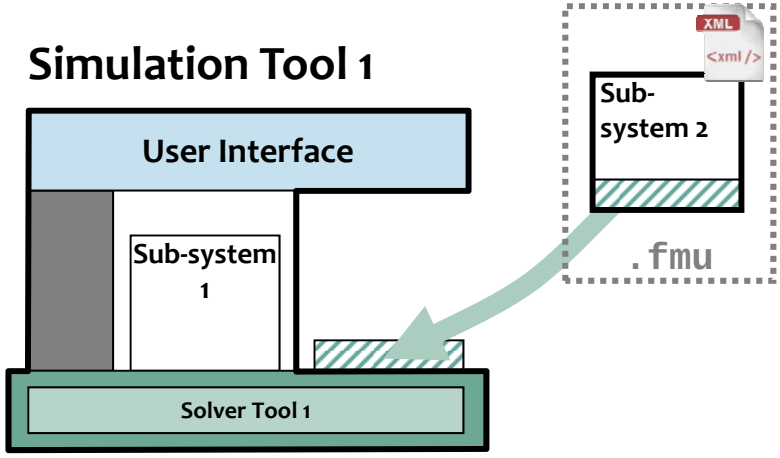


FMI for Model Exchange: Export/Import/Simulation

- Step 1 – **Export**: **Subsystem model is exported** from its **simulation tool**
  - Preparation as FMU-archive containing
    - model description (xml-file)
    - executable dll-file containing model equations
    - optionally C source code



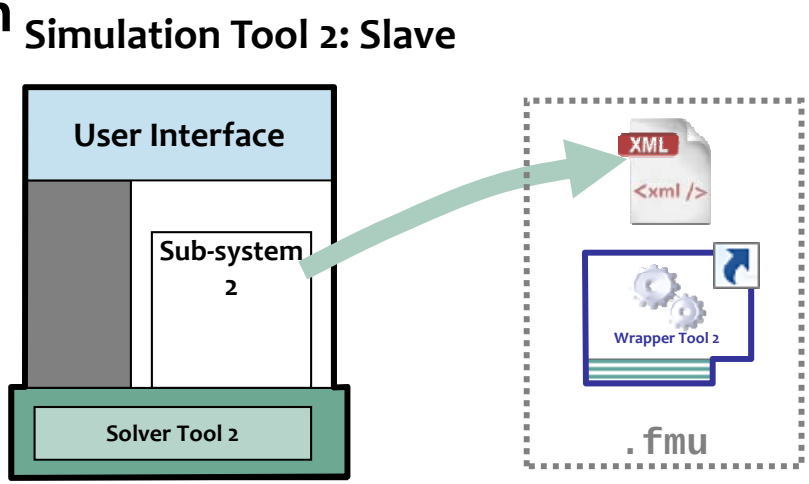
- Step 2 – **Import**: **Subsystem model is imported** into simulation system for **system simulation**
  - Reading FMU-archive
    - model information from xml-file
    - connecting subsystem variables
    - running system simulation



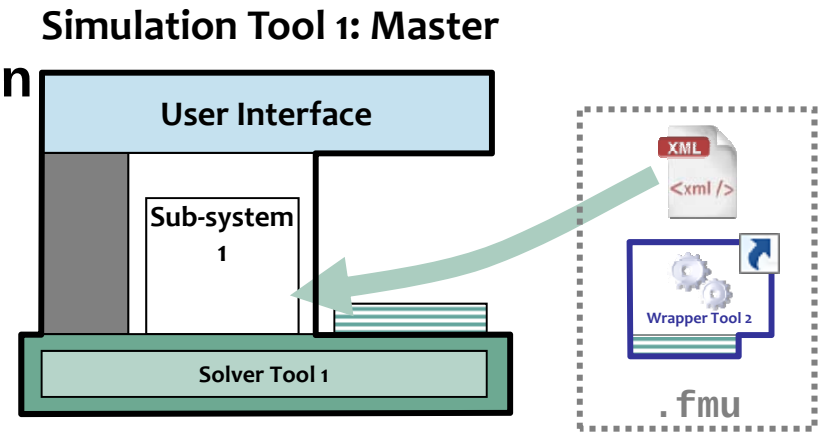
FMI for Co-Simulation (tool coupling)

Use case schematic 1/2

- Step 1 – **Export: Subsystem description** is exported from its simulation tool
  - Preparation as FMU-archive containing
    - model description (xml-file), describes also solver/tool capabilities
    - reference to executable dll-file as, wrapper which provides a tool specific implementation of the co-simulation slave interface



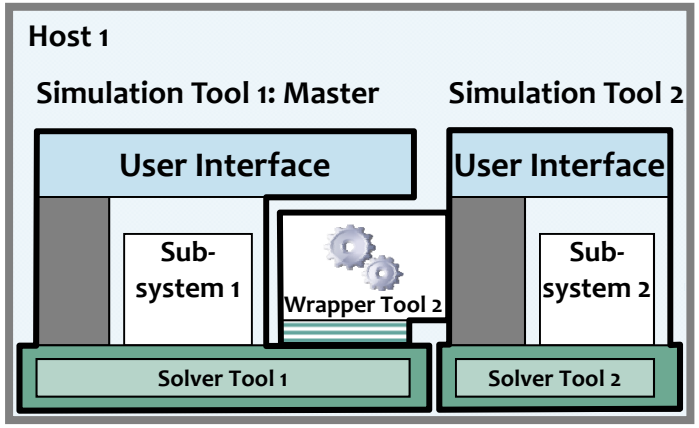
- Step 2 – **Import: Subsystem description** is imported into simulation system for system simulation
  - Reading FMU-archive
    - model information from xml-file
    - connecting subsystem variables



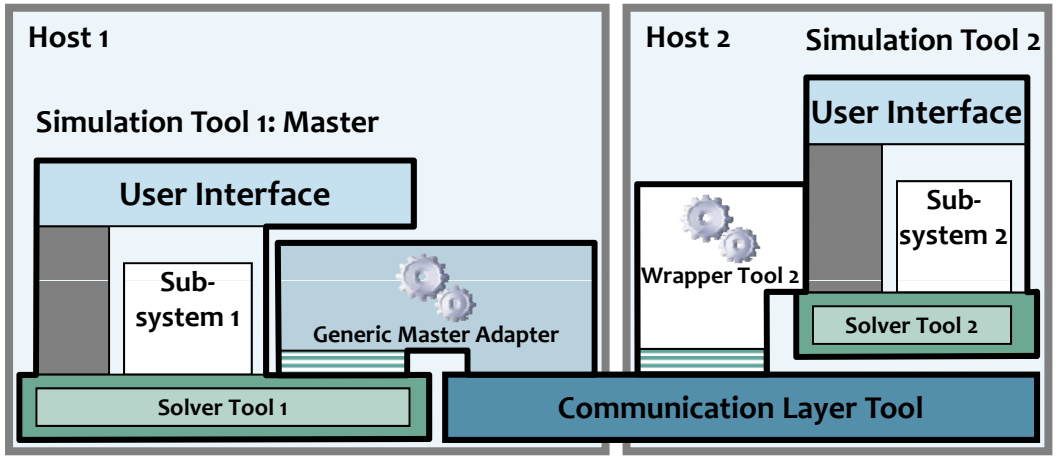
FMI for Co-Simulation (tool coupling)

Use case schematic 2/2

- Step 3a – **Run simulation on same host**
  - Master subsystem is connected with wrapper dll via co-simulation interface
  - Subsystem 2 is called via wrapper of tool 2 as if it would have been directly imported into master simulation tool



- Step 3b – **Run simulation on different hosts**
  - Master subsystem is connected via a generic adapter with a communication tool
    - Adapter provides co-simulation slave interface
  - Communication tool uses wrapper dlls of slave tools



 Peter Fritzson presenter. Courtesy MODELISAR project, Martin Otter et al

Complex Task to Develop FMI, Many Aspects

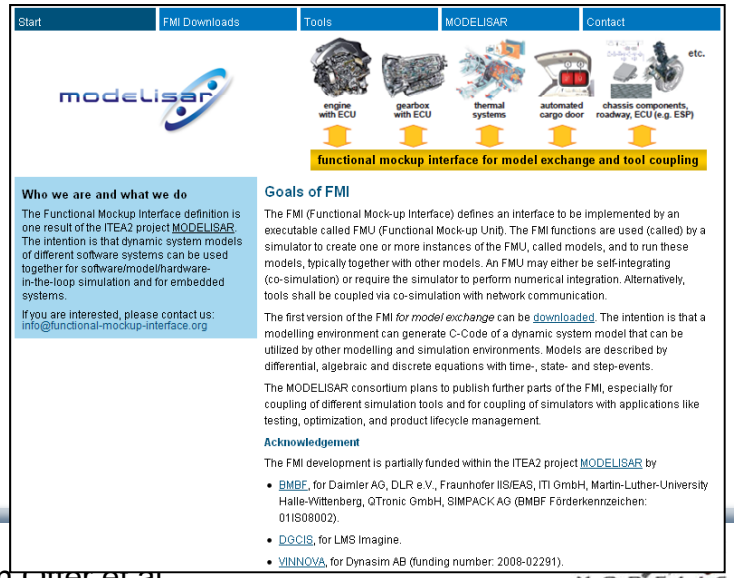
- **Model Exchange** (ODE/DAE components without integrators)
- **Co-Simulation** (ODE/DAE components with integrators)
- **Co-Simulation** with **PDE solver** (MpCCI)
- **AUTOSAR** (discrete components with complex communication)
- **Simulation Backplane**

In Jan. 2010, the first version of "FMI for Model Exchange" was released.
It was mainly developed by Dassault Systèmes (Dynasim), DLR, ITI, QTronic.

On Oct. 12, 2010, the first version of “FMI for Co-Simulation” was released.

www.functional-mockup-interface.org

- **specification**
- **xml schema files**
- **C header files**
- **software development kit (QTronic)**



MODELISAR Project

- ITEA2 project
- 3 years (2008 – 2011)
- 29 project partners
- Coordinators
  - Dassault Systèmes
  - Daimler AG
- **Budget** / Funding
  30M€ / 10M€
- Funded by
  Germany (BMBF)
  France (DGCIS)
  Sweden (VINNOVA)
  Belgium
  Austria

Model Distribution as a Zip-file (.fmu file)

A model is distributed as one zip-file with extension ".fmu“, containing:

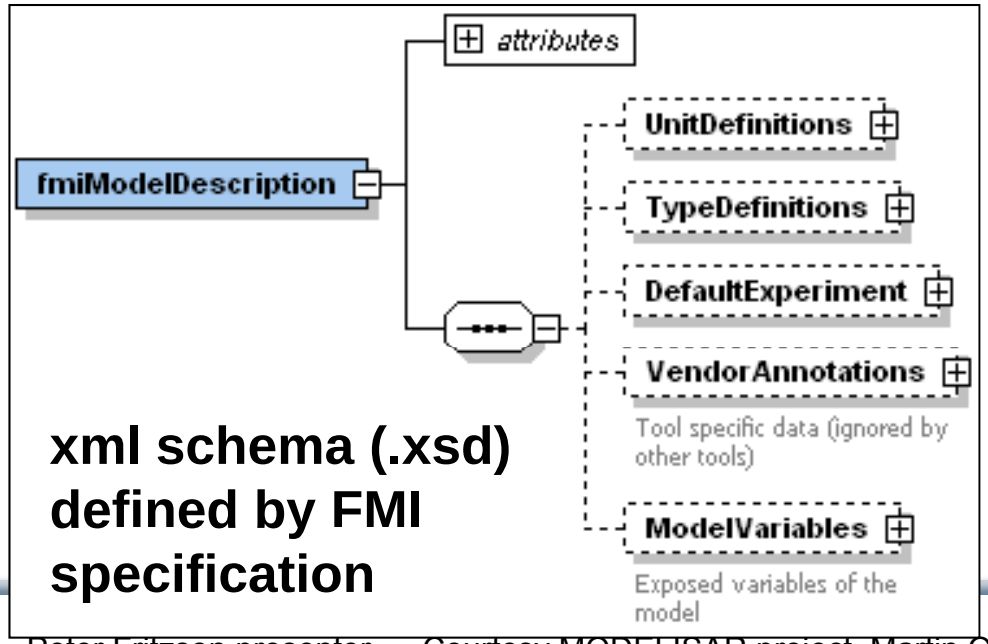
- **XML model description file**
  All model information that is not needed during integration of model, e.g., signal names and attributes. Advantage:
  - No overhead for model execution.
  - Tools can read this information (= complicated data structure) with their prefered language (C++, C#, Java, ...)
- **Model equations** defined by a small set of **C-functions**. In zip-file:
  - **C source code** and/or
  - **Binary code** (DLL) for one or more platforms (Windows, Linux, ...)
- **Resources**
  - Documentation (html files)
  - Model icon (bitmap file)
  - Maps and tables (read by model during initialization)

Structure of an FMU zip-file

```
1. modelDescription.xml     // Description of model (required file)
2. model.png                // Optional image file of model icon
3. documentation            // Optional directory containing the model
                           // documentation
   _main.html              // Entry point of the documentation
   <other documentation files>
4. 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
5. 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
      ...
6. resources  // Optional resources needed by the model
   < data in model specific files which will be read during initialization >
```

Model Description Schema

- **Model information** not needed for execution is stored in one **xml-file** (modelVariables.xml in zip-file) defined by **xml schema (.xsd) files**.
- Advantage: Complex data structures give still simple interface, and tool can use its favorite programming language for reading (e.g., C++, C#, Java).



xml schema (.xsd) defined by FMI specification

Definition of display units

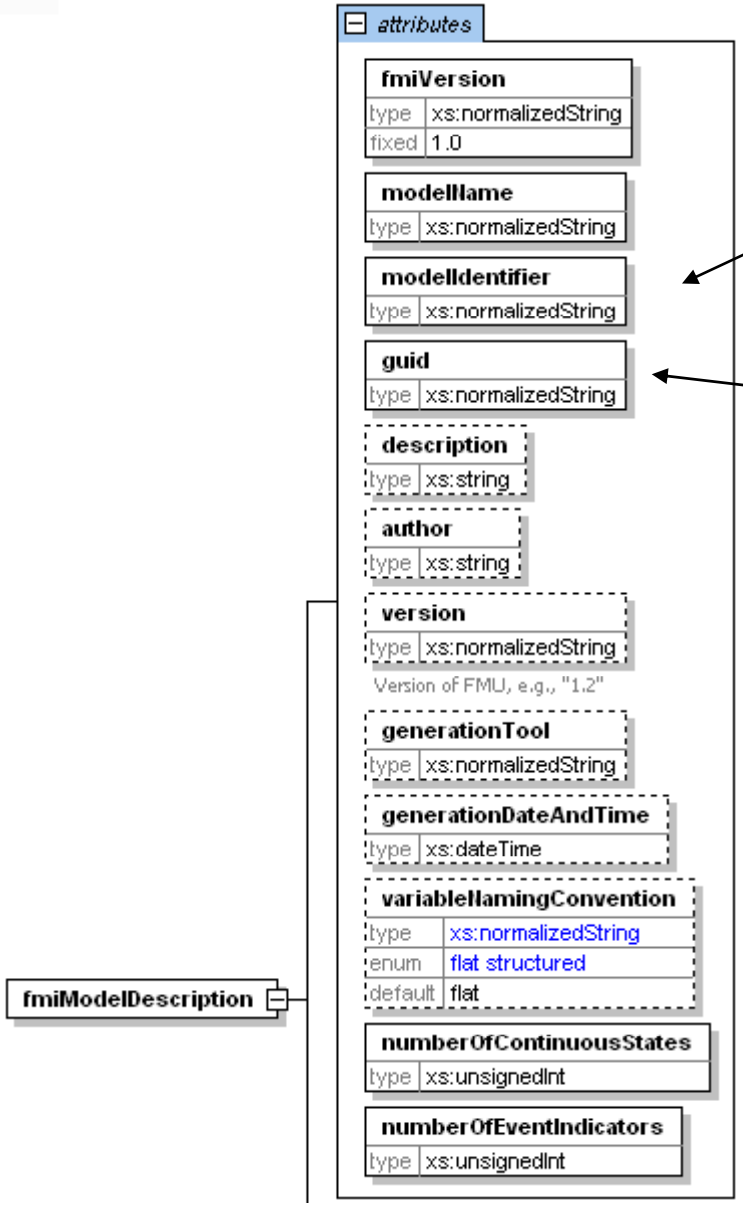
Definition of type defaults

Default stop time, tol. etc.

Tool specific data

Variable names and attributes

Model Attributes

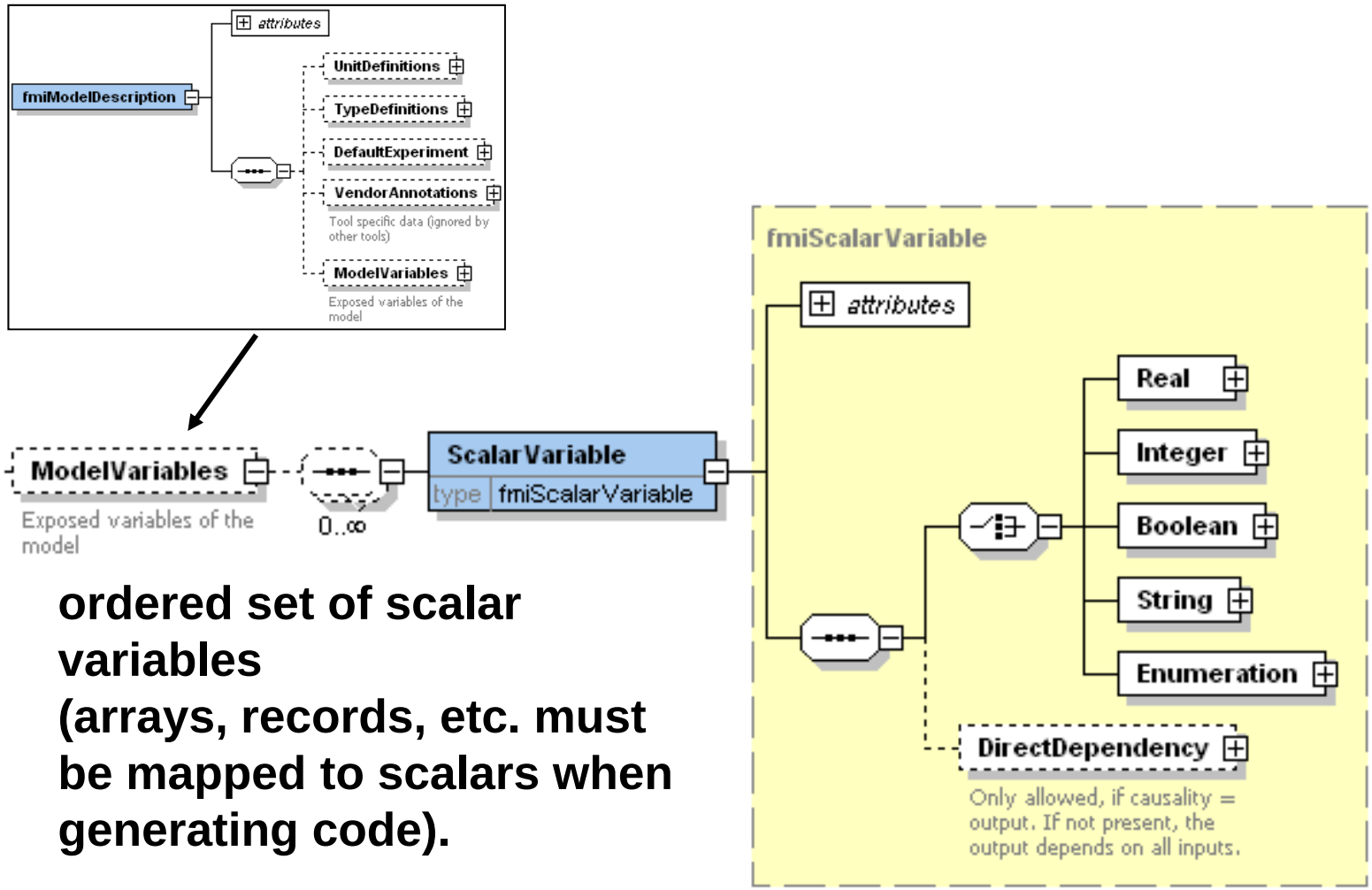


modelIdentifier is a C-name that is used as prefix for the C-functions (model interface)

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 dynamica during simulation)

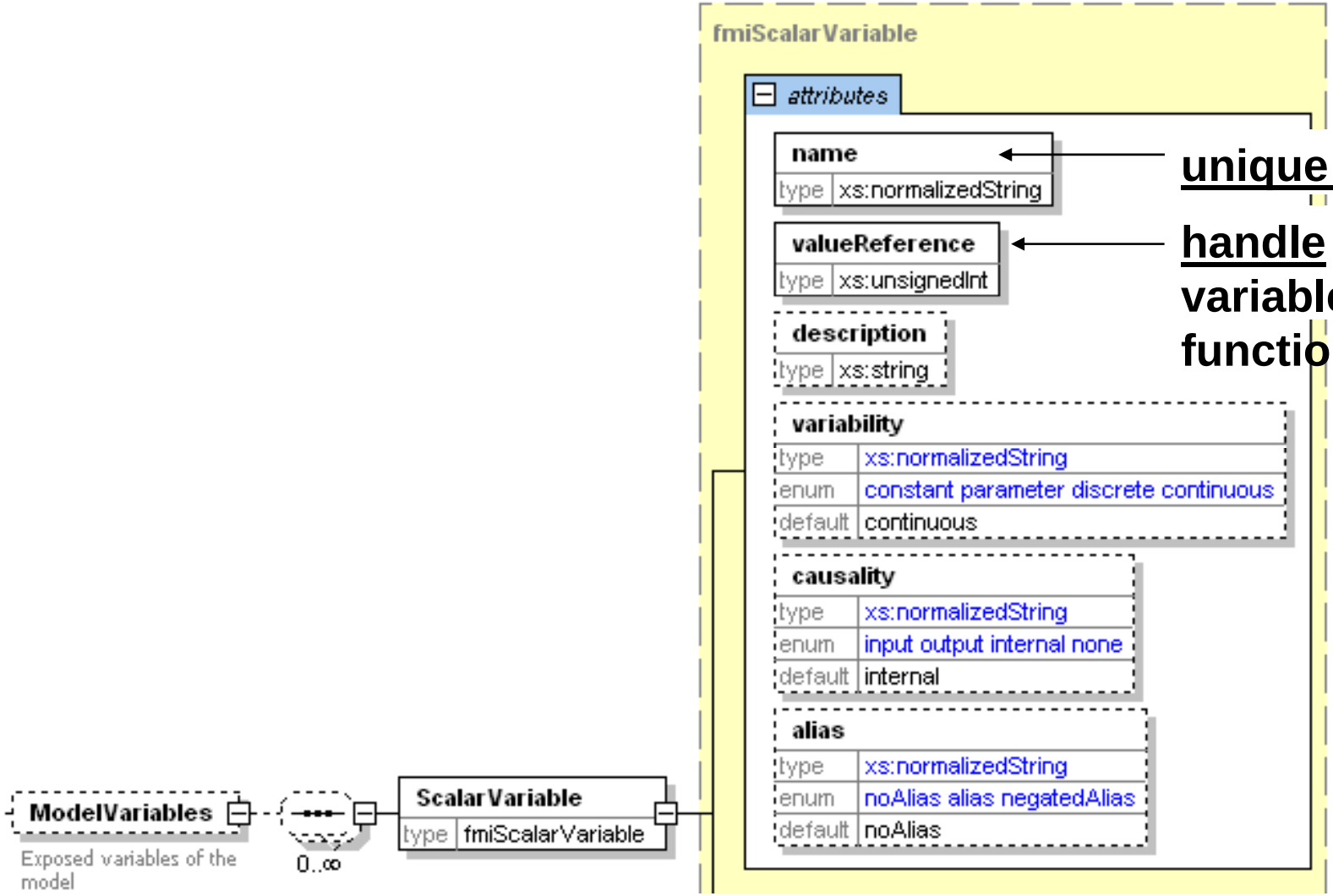
Model Variables



data types

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

Attributes of Model Variables



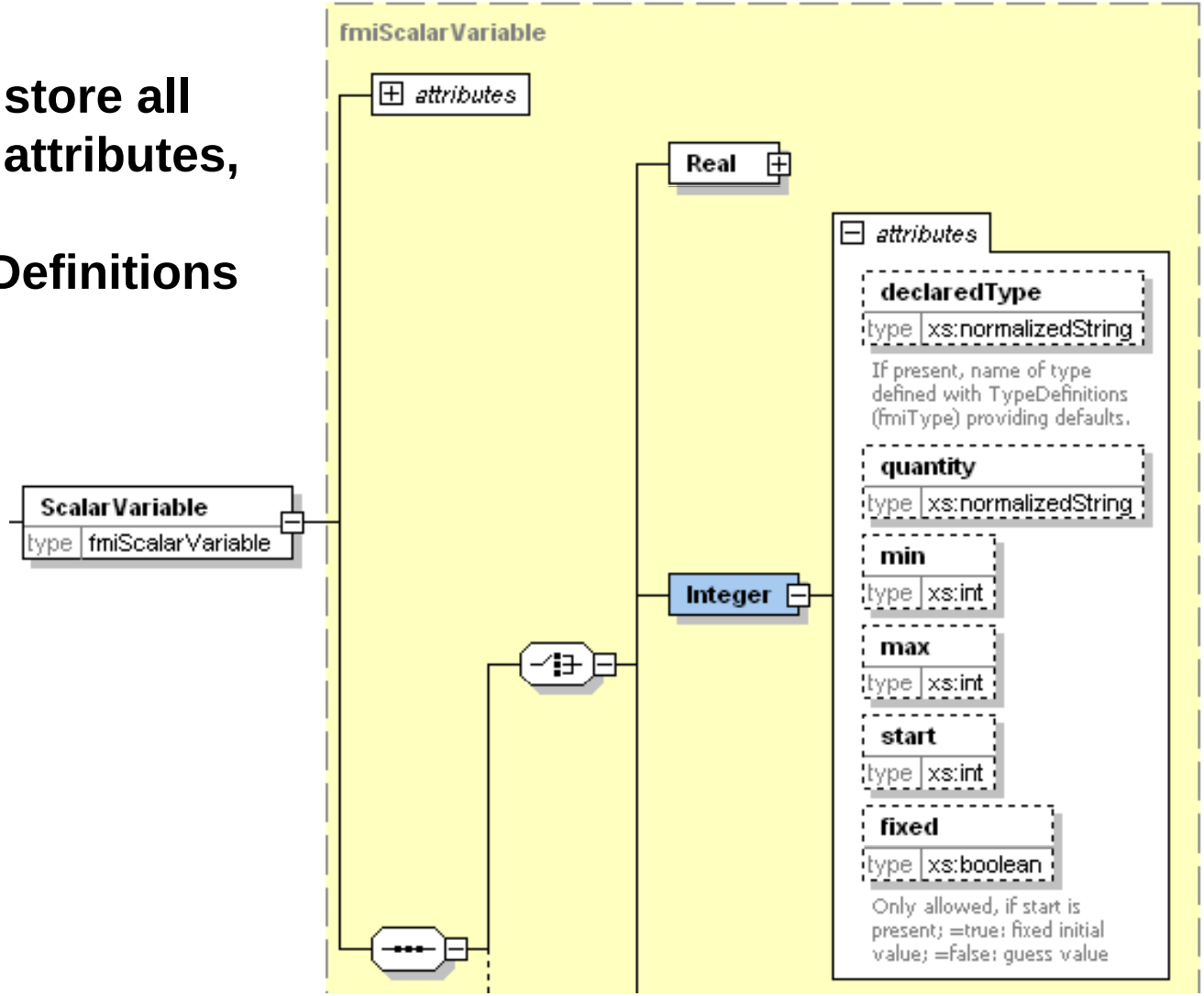
unique name

handle to identify variable in C-functions

...

Data Types

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



Example Model Description XML File

modelDescription.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>
```

Model C-Interface – Two Header Files

- **Platform dependent definitions (basic types)**

```
/* Platform (combination of machine, compiler, operating system) */
#define fmiModelTypesPlatform "standard32"

/* Type definitions of variables passed as arguments */
   typedef void*        fmiComponent;
   typedef unsigned int fmiValueReference;
   typedef double       fmiReal    ;
   typedef int          fmiInteger;
   typedef char         fmiBoolean;
   typedef const char*  fmiString ;

/* Values for fmiBoolean  */
#define fmiTrue  1
#define fmiFalse 0

/* Undefined value for fmiValueReference (largest unsigned int value) */
#define fmiUndefinedValueReference (fmiValueReference)(-1)
```

- **C-function interfaces**
  - **18 core functions**
  - **6 utility functions**
  - **no macros**
  - **C-function name:** <ModelIdentifier>_<name>**, e.g. "**Drive_fmiSetTime**"**

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`

Example Use:

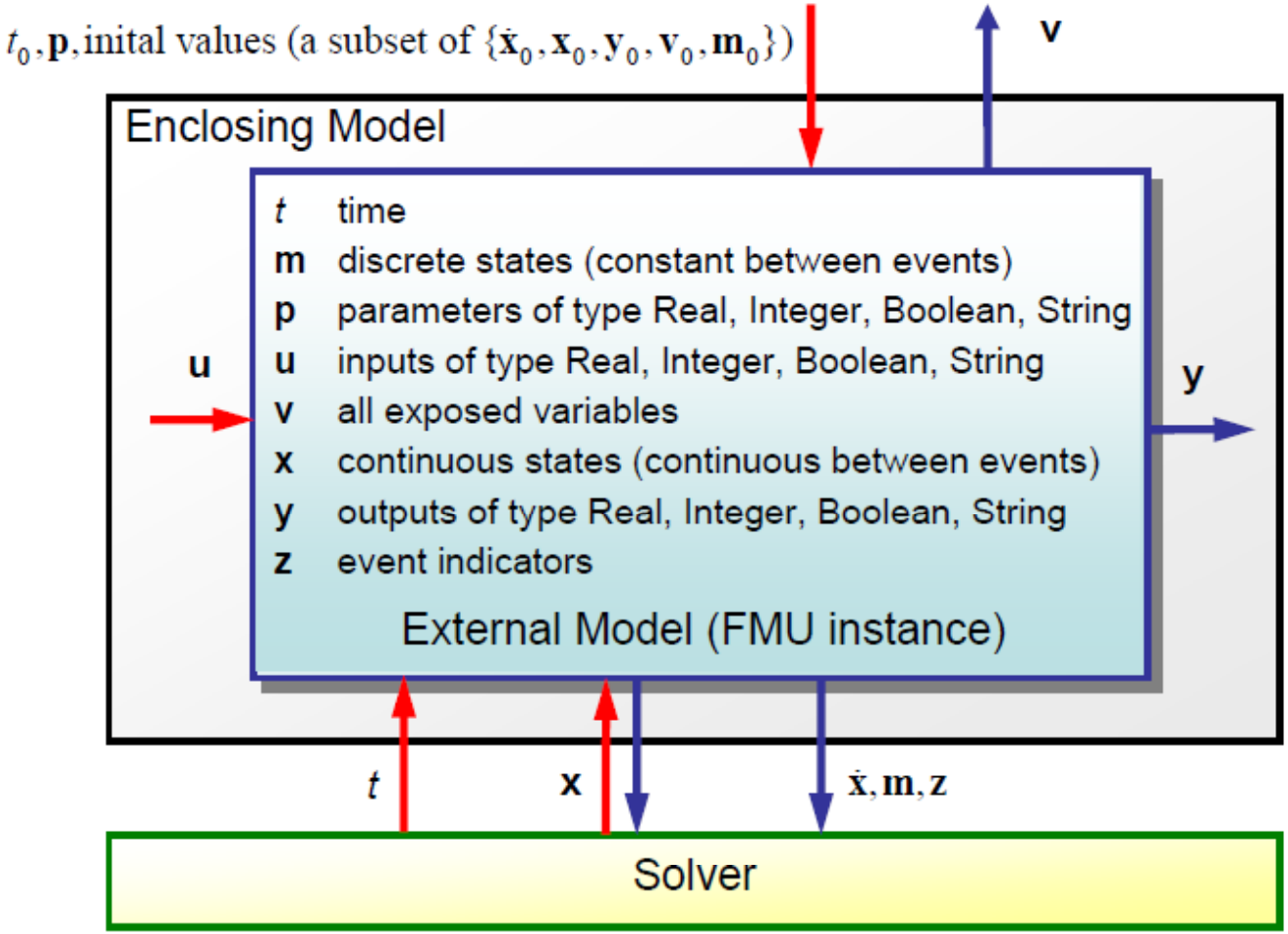
```
// Set input arguments
fmiSetTime(m, time);
fmiSetReal(m, id_u1, u1, nu1);
fmiSetContinuousStates(m, x, nx);

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

FMI Functions and their use

Mathematical description of an FMU

description	range of t	equation
initialization	$t = t_0$	$(\mathbf{m}, \mathbf{x}, \mathbf{p}, T_{next}) = \mathbf{f}_0(\mathbf{u}, t_0,$ subset of $\{\mathbf{p}, \dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{v}_0, \mathbf{m}_0\})$
derivatives $\dot{\mathbf{x}}(t)$	$t_i \le t < t_{i+1}$	$\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
outputs $\mathbf{y}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{y} = \mathbf{f}_y(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
internal variables $\mathbf{v}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{v} = \mathbf{f}_v(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
event indicators $\mathbf{z}(t)$	$t_i \le t < t_{i+1}$	$\mathbf{z} = \mathbf{f}_z(\mathbf{x}, \mathbf{m}, \mathbf{u}, \mathbf{p}, t)$
event update	$t = t_{i+1}$	$(\mathbf{x}, \mathbf{m}, T_{next}) = \mathbf{f}_m(\mathbf{x}^-, \mathbf{m}^-, \mathbf{u}, \mathbf{p}, t_{i+1})$
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

Signals of an FMU (Functional Mockup Unit)



For example: 10 input/output signals (u/y) for connection and 100 000 internal variables (v) for plotting

Ongoing Tool Support for FMI
Model Exchange, Export and/or Import

- **AMESim**. For Modelica. By LMS
- **CATIA Systems**. For CAD/Modelica. By Dassault Systèmes.
- **Dymola 7.4**. Available for Modelica. By Dassault Systèmes.
- **EXITE ACE** (co-simulation of software/AUTOSAR). By Extessy
- **Simulation X**. For Modelica. By ITI Systems.
- **SIMULINK**. Export avail. in Dymola 7.4. Dassault Systèmes.
- **JModelica.org**. Beta version of import. By Modelon.
- **OpenModelica**. ”beta version” export. Import started. By OSMC
- **EXITE**. (Import, co-simulation), By Extessy
- **Silver/TestWeaver**. (FMU integration,cosimulation, testing). Qtronic
- **SIMPACK**. (Import, Multibody). By SIMPACK

Comparison FMI – SIMULINK S-Function

- **S-function is designed to import models into SIMULINK**
  - Interface to construct a model (S-function) for import in SIMULINK.
  - No specification to call S-Function in another simulator.
  - **Model DLL depends on target simulator, needs one DLL per target**
    (SIMULINK/S-Function models can be imported in other simulation environments via the Target Language Compiler of Real-Time Workshop depends on target: Model used in 3 environments → 3 DLLs of same model)
  - Proprietary format. Might give legal problems if used in other simulators
- **FMI is designed to import models into many simulation environments**
  - No specification to construct a model.
  - Interface definition to call a FMI-model in a simulation environment.
  - **Model DLL does not depend on target simulator**
    (If 1 model is used in 3 simulation environ. on the same platform → 1 DLL)
  - Open source format (Wikipedia license for spec, BSD for schema/header)

Comparison: Differences in Design

- **S-function is complex**
  - **Hundreds of C-functions**/macros (e.g. simstruct.h: 9000 lines of code).
  - No simulator can import all S-function models since too complex (with exception of SIMULINK)
  - S-Function **distribution** is **inconvenient**, since **several files** (DLL, C-source, model specific maps, etc.)

- **FMI is simple**
  - **20 C-functions** (e.g. fmiModelFunctions.h: 200 lines of code).
  - Simulators can rather easily support the full interface, due to simplicity.
  - FMI-model **distribution** is **convenient**, since **one file** (all files are stored in one zip-file, including DLLs for several platforms).

Comparison: Better technical solutions in FMI
(not available in S-Function interface)

- **Advanced co-simulation interface**
  Support for higher order extrapolation/interpolation of interface variables, variable communication step-size, rejecting a communication step.
- **Reliable event handling for hybrid systems**
  Clean state event handling + event iteration over simulation model
- **Reliable and efficient handling of algebraic equations** in model.
  - If no convergence of equation solver, model can request step-size reduction in simulator.
  - Solution of equation systems are cached for reuse of subsequent calls.
- **Support for efficient storing results of internal variables**
  - Allows to store results of many (e.g.100 000) variables in a practical way.
  - Optimized if a = b or a = -b, then only time response of "a" needs to be stored in the result file.
    For Modelica models size of result files is typically reduced by factor 4-5.

Ongoing Work – OpenModelica FMI Export/Import

Current status, completed, FMI Export:

- XML Data export.
- C FMI interface functions.
- C export for variable definitions.
- C export for functions like model initialization, real variables, etc.
- FMU Makefile
- API for FMI translation translateModelFMU(ModelName)

Status, FMI Export/Import OpenModelica

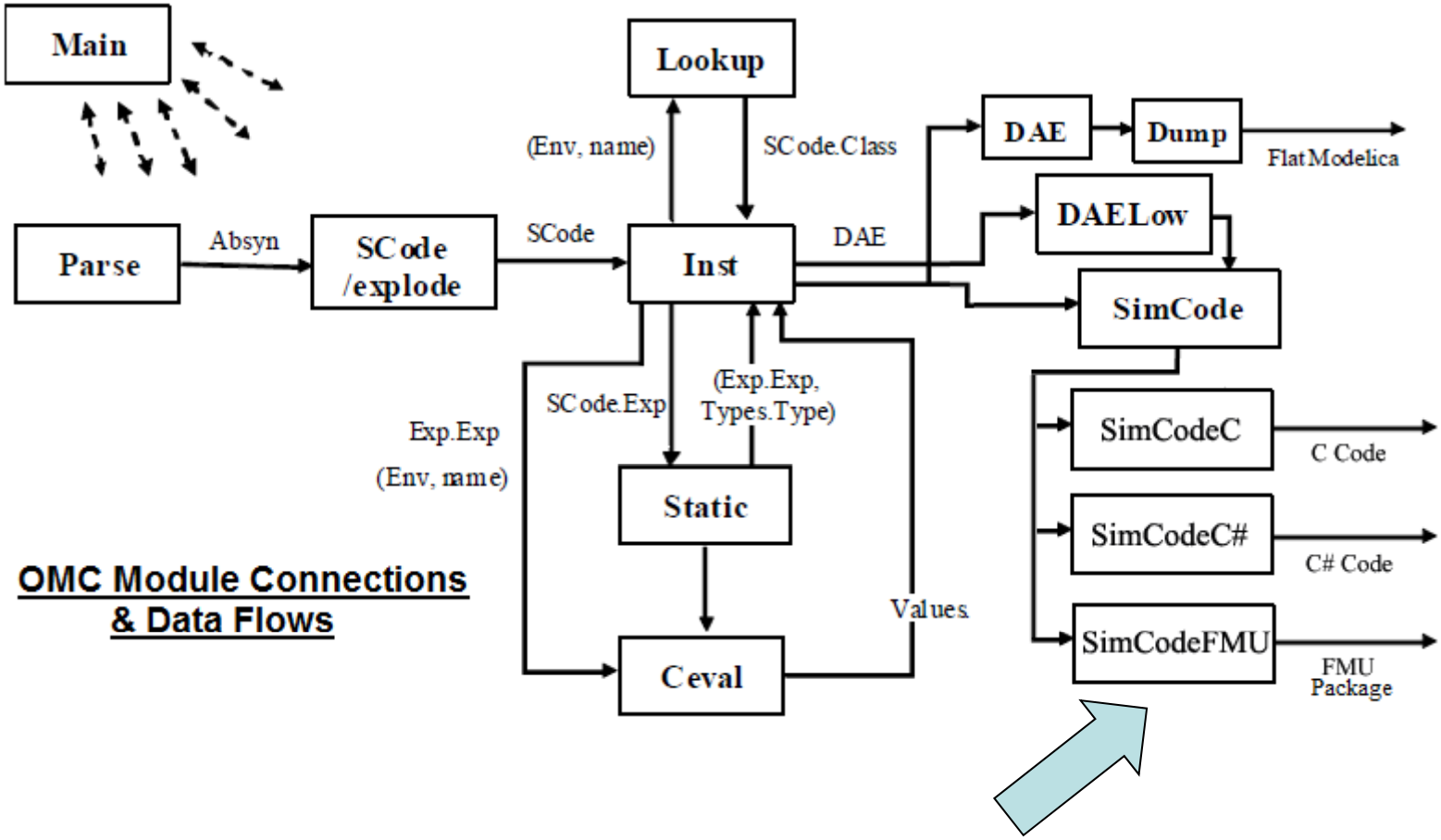
Left to Develop for FMI Export

- Getting event indicators/zero crossings.
- Event update function.
- Getting time events.
- Testing exported FMUs.

FMI Import

- Work recently started

OpenModelica Compiler and Code Generators Including New SimCodeFMU for FMI Export



 Peter Fritzson presenter. Courtesy MODELISAR project, Martin Otter et al

Outlook and Conclusion

- "**FMI for Model Exchange**" released at end of January 2010 (available from "**http://www.functional-mockup-interface.org**")
- "**FMI for Co-Simulation**" released on Oct. 12, 2010:
  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 developed**. A lot of requirements available, such as:
  - Sparse Jacobian
  - Direct support for arrays and records in xml schema
  - Improved sample time definition + more data types (for embedded systems)
  - Online changeable parameters
  - Saving/restoring model state
- **Other tool vendors** are encouraged to support FMI.