The Functional Mockup Interface for Tool independent Exchange of Simulation Models

Torsten Blochwitz (ITI), Martin Otter (DLR-RM)

M. Arnold
C. Bausch, M. Monteiro
C. Clauß, S. Wolf
H. Elmqvist, H. Olsson
A. Junghanns, J. Mauss
T. Neidhold
D. Neumerkel
J.-V. Peetz

University of Halle
Atego Systems GmbH
Fraunhofer IIS EAS, Dresden
Dassault Systèmes, Lund
QTronic, Berlin
ITI, Dresden,
Daimler AG, Stuttgart
Fraunhofer SCAI, St. Augustin

Modelica 2011, 21st, 22nd March
Functional Mock-up Interface (FMI) - Motivation (1)

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)
FMI - Motivation (2)

- No standards available for:
  - Model interface based on C or binaries
  - Co-simulation between simulation tools

- Lots of proprietary interfaces:
  - Simulink: S-function
  - Modelica: external function, external object interface
  - QTronic Silver: Silver-Module API
  - SimulationX: External Model Interface
  - NI LabVIEW: External Model Interface, Simulation Interface Toolkit
  - Simpack: uforce routines
  - ADAMS: user routines
  - ...
The FMI development is part of the ITEA2 MODELISAR project (2008 - 2011; 29 partners, Budget: 30 Mill. €)

- FMI development initiated, organized and headed by Daimler AG
- Improved Software/Model/Hardware-in-the-Loop Simulation, of physical models from different vendors.
- Open Standard
- 14 Automotive Use-Cases to evaluate FMI.

Engine with ECU
Gearbox with ECU
Thermal systems
Automated cargo door
Chassis components, roadway, ECU (e.g. ESP)

functional mockup interface for model exchange and tool coupling
courtesy Daimler
FMI - Main Design Idea (1)

- FMI for Model Exchange:
  - Version 1.0 released in January 2010

- FMI for Co-Simulation:
  - Reuses as much as possible from FMI for Model Exchange standard
  - Version 1.0 released in October 2010
FMI - Main Design Idea (2)

- A component which implements the interface is called *Functional Mockup Unit (FMU)*

- Separation of
  - Description of interface data (XML file)
  - Functionality (C code or binary)

- A FMU is a zipped file (*.fmu) containing the XML description file and the implementation in source or binary form

- Additional data and functionality can be included

XML schema (.xsd) defined by the FMI specification

control

simulator GUI

reads

model.dll

run 1 or many

simulator solver

references

fmiModelDescription

attributes

UnitDefinitions

TypeDefinistions

DefaultExperiment

VendorAnnotations

Exposed variables of the model

Tool specific data (ignored by other tools)

modelDescription.xml

.fmu
FMI XML Schema

- **Information** not needed for execution is stored in one xml-file:
  - Complex data structures give still simple interface.
  - Reduced overhead in terms of memory.

```
- attributes
  - UnitDefinitions
  - TypeDefinitions
- DefaultExperiment

- attributes
  - startTime
  - stopTime
  - tolerance

- VendorAnnotations
  - Tool specific data (ignored by other tools)

- ModelVariables
  - Exposed variables of the model

- Implementation
  - type fmiImplementation
    - CoSimulation_StandAlone
    - CoSimulation_Tool

- fmiModelDescription
```

Definition of display units
Definition of type defaults

Default stop time, tol. etc.

Tool specific data

Variable names and attributes
Example

```xml
<?xml version="1.0" encoding="UTF8"?>
<fmiModelDescription
  fmiVersion="1.0"
  modelIdentifier="Modelica_Mechanics_Rotational_Examples_Friction"
  guid="{8c4e810f-3df3-4a00-8276-176fa3c9f9e0}"
...
nNumberOfContinuousStates="6"
nNumberOfEventIndicators="34"/>
<UnitDefinitions>
  <BaseUnit unit="rad">
    <DisplayUnitDefinition displayUnit="deg" gain="57.295795130823"/>
  </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>
```
C-Interface

- Two C-header files:
  - **Platform dependent definitions** (basic types):

```c
/* 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-functions:**
  - 18 core functions
  - 6 utility functions
  - no macros
  - C-function name: `<ModelIdentifier>_ <name>`, e.g. `Drive_fmiSetTime`
C-Interface

- Instantiation:
  
  \[
  \text{fmiComponent fmiInstantiateXXX}(\text{fmiString instanceName}, \ldots)\]

- Returns an instance of the FMU. Returned \text{fmiComponent} is a parameter of the other interface functions. It is of type \text{void*} for the master. The FMU uses it to hold all necessary information.

- Functions for initialization, termination, destruction

- Support of real, integer, boolean, and string inputs, outputs, parameters

- Set and Get functions for each type:
  
  \[
  \begin{align*}
  \text{fmiStatus fmiSetReal} & \quad (\text{fmiComponent c},) \\
  & \quad \text{const fmiValueReference vr[]}, \text{size_t nvr,} \\
  & \quad \text{const fmiReal value[]} \\
  \text{fmiStatus fmiSetInteger} & \quad (\text{fmiComponent c},) \\
  & \quad \text{const fmiValueReference vr[]}, \text{size_t nvr,} \\
  & \quad \text{const fmiInteger value[]} \\
  \end{align*}
  \]

- Identification by \text{valueReference}, defined in the XML description file for each variable
FMI for Model Exchange (1)

- Import and export of input/output blocks (FMU – Functional Mock-up Unit)
- described by
  - differential-, algebraic-, discrete equations,
  - with time-, state, and step-events
- FMU can be large (e.g. 100000 variables)
- FMU can be used in an embedded system (small overhead)
- FMUs can be connected together
FMI for Model Exchange

Signals of an FMU

For example: 10 input/output signals (u/y) for connection and 100000 internal variables (v) for plotting
### Modelica 2011: Functional Mockup Interface

#### Slide 14

<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, \text{subset of } {p, \dot{x}_0, x_0, y_0, v_0, m_0}))</td>
<td>fmiInitialize, fmiGetReal/Integer/Boolean/String, fmiGetContinuousStates, fmiGetNominalContinuousStates</td>
</tr>
<tr>
<td>Derivatives</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</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</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</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, fmiGetReal/Integer/Boolean/String, fmiGetContinuousStates, fmiGetNominalStates, fmiGetStateValueReferences</td>
</tr>
</tbody>
</table>

**Example:**

```plaintext
// 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);
```
Co-Simulation

Definition:
- Coupling of several simulation tools
- Each tool treats one part of a modular coupled problem
- Data exchange is restricted to discrete communication points
- Subsystems are solved independently between communication points

Motivation:
- Simulation of heterogeneous systems
- Partitioning and parallelization of large systems
- Multirate integration
- Hardware-in-the-loop simulation
FMI for Co-Simulation

- Master/slave architecture
- Considers different capabilities of simulation tools
- Support of simple and sophisticated coupling algorithms:
  - Iterative and straightforward algorithms
  - Constant and variable communication step size
- Allows (higher order) interpolation of continuous inputs
- Support of local and distributed co-simulation scenarios

- FMI for Co-Simulation does not define:
  - Co-simulation algorithms
  - Communication technology for distributed scenarios
FMI for Co-Simulation

Signals of an FMU for Co-Simulation

- Inputs, outputs, and parameters, status information
- Derivatives of inputs, outputs w.r.t. time can be set/retrieved for supporting of higher order approximation
FMI for Co-Simulation
C-Interface

→ Execution of a time step:
  fmiStatus fmiDoStep(fmiComponent c,
                     fmiReal currentCommunicationPoint,
                     fmiReal communicationStepSize, fmiBoolean newStep)

→ communicationStepSize can be zero in case of event iteration
→ newStep = fmiTrue if last step was accepted by the master
→ It depends on the capabilities of the slave which parameter constellations and calling sequences are allowed
→ Depending on internal state of the slave and the function parameters, slave can decide which action is to be done before the computation
→ Return values are fmiOK, fmiDiscard, fmiError, fmiPending
→ Asynchronous execution is possible
FMI for Co-Simulation
Use Case

Co-Simulation stand alone:

- Executable
  - Master
  -Slave
  -Model
  -Solver

Co-Simulation tool:

- Executable
  - Master
  -Slave
  -Model
  -Solver
- Library (DLL)
  - FMI Wrapper
  - Process 1
  - Process 2
- Simulation tool
FMI for Co-Simulation
Use Case

- Distributed co-simulation scenario

- Data exchange is handled by a communication layer which is implemented by a special FMI wrapper

- Master and slave utilize FMI for Co-Simulation only
## Tools supporting FMI (from FMI web site, March 2011)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMESim</td>
<td>(FMU export and import)</td>
</tr>
<tr>
<td>Dymola 7.4</td>
<td>(FMU export and import; now available)</td>
</tr>
<tr>
<td>EXITE ACE</td>
<td>(FMU export and import)</td>
</tr>
<tr>
<td>EXITE</td>
<td>(FMU import)</td>
</tr>
<tr>
<td>FMU SDK</td>
<td>(FMU export and import; now available)</td>
</tr>
<tr>
<td>JModelica.org</td>
<td>(FMU export and import; now available)</td>
</tr>
<tr>
<td>NI VeriStand</td>
<td>(FMU Co-Simulation)</td>
</tr>
<tr>
<td>NI LabVIEW</td>
<td>(FMU import)</td>
</tr>
<tr>
<td>Silver 2.0</td>
<td>(FMU import; now available)</td>
</tr>
<tr>
<td>SIMPACK</td>
<td>(FMU import)</td>
</tr>
<tr>
<td>SimulationX 3.4</td>
<td>(FMU export and import; FMU Co-Simulation; now available)</td>
</tr>
<tr>
<td>Simulink</td>
<td>(FMU export now available by Dymola 7.4 via Real-Time Workshop)</td>
</tr>
<tr>
<td>Simulink</td>
<td>(FMU export now available via @Source)</td>
</tr>
<tr>
<td>TISC</td>
<td>(FMU import)</td>
</tr>
</tbody>
</table>
Conclusions and Outlook

- FMI has a high potential being widely accepted in the CAE world:
  - Initiated, organized and pushed by Daimler to significantly improve the exchange of simulation models between suppliers and OEMs.
  - Defined in close collaboration of different tool vendors.
  - Industrial users were involved in the proof of concept.
  - FMI can already be used with several Modelica tools, Simulink, multi-body and other tools.

- FMI is maintained and further developed:
  - Unification and harmonization of FMI for Model Exchange and Co-Simulation (FMI 2.0) within Modelisar.
  - Improved handling of time events.
  - Clean handling of changeable parameters.
  - Efficient interface to Jacobian matrices.
Acknowledgments

FMI initiated and organized: Daimler AG (Bernd Relovsky, ....)

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

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

**FMI for Co-Simulation**
- Martin Arnold, University Halle, Germany

**Core-Design by:**
- Constanze Bausch, Atego Systems GmbH, Wolfsburg, Germany
- Torsten Blochwitz, ITI GmbH, Dresden, Germany
- Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany
- Manuel Monteiro, Atego Systems GmbH, Wolfsburg, Germany
- Thomas Neidhold, ITI GmbH, Dresden, Germany
- Jörg-Volker Peetz, Fraunhofer SCAI, St. Augustin, Germany
- Susann Wolf, Fraunhofer IIS EAS, Dresden, Germany

**Contributors:**
- Jens Bastian, Fraunhofer IIS EAS, Dresden, Germany
- Christoph Clauß, Fraunhofer IIS EAS, Dresden, Germany
- Dietmar Neumerkel, Daimler AG, Böblingen, Germany
- Martin Otter, DLR, Oberpfaffenhofen, Germany
- Tom Schierz, University Halle, Germany
- Wolfgang Trautenberg, Simpack AG, Germany
- Klaus Wolf, Fraunhofer SCAI, St. Augustin, Germany

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