Functional Mockup Interface for Model Exchange and Co-Simulation

T. Blochwitz, ITI, Dresden
M. Otter, DLR, Oberpfaffenhofen
J. Akesson, Modelon, Lund,
M. Arnold, University of Halle
C. Clauß, Fraunhofer IIS EAS, Dresden
H. Elmqvist, H. Olsson, Dassault Systèmes, Lund
M. Friedrich, Simpack AG, Gilching
A. Junghanns, J. Mauss, QTronic, Berlin
D. Neumerkel, Daimler AG, Stuttgart
A. Viel, LMS Imagine, Roanne
Contents

- Motivation
- Main Design Idea
- FMI for Model Exchange and Co-Simulation
- New Features of FMI 2.0
  - Unification
  - Classification of Interface Variables
  - Save and Restore FMU State
  - Dependency Information
  - Partial Derivatives, Jacobian Matrices
- Tools supporting FMI
- FMI Modelica Association Project
- Conclusion
- Outlook
Modeling and Simulation are applied in all stages of system design

**Concept**
- Requirements
- Executable Specification

**Design and Optimization**
- Component
  - FEM, CFD
  - MBS
  - Modelica
  - VHDL-AMS
  - Signal based

**Implementation**
- Model Predictive Control
- Observers

**Test and Verification**
- MiL
- SiL
- HiL
- VHiL
Motivation

Challenges for Functional Mockup:
- Different tools and languages are involved
- No standards for model interface and co-simulation available
- Protection of model IP and know-how of supplier

Modelisar project:
- Functional Mockup Interface for Model Exchange and Co-Simulation
Functional Mockup Interface

EU project Modelisar (2008 – 2011, 26 Mill. €, 178 my)

- Initiated by Daimler AG, 28 European partners
  - Tool vendors
  - Users
  - Research organizations
- Proof of concept in industrial use cases

After 2011

- Continuation as Modelica Association Project
- Modelica Association changed its bylaws to become an umbrella organization for projects related to model based system design

MODELISAR (ITEA 2 ~ 07006)

- Partners
  - ARMINES
  - Arsenal Research
  - ATB
  - AVL
  - Berata
  - Daimler
  - Dassault Systèmes
  - David
  - DLR
  - Dynasim
  - Exlessy
  - FhG First, IIS EAS, SCAI
  - Geensys
  - Halle University
  - IFP
  - Imagine
  - INSPIRE
  - SIMPACK AG
  - ITI
  - LMS International
  - QTronic
  - Schneider Electric
  - Trialog
  - Triphase
  - TWT
  - Verhaert
  - Volkswagen
  - Volvo
FMI – Main Design Idea

- **FMI for Model Exchange**

  ![Diagram of FMI for Model Exchange]

- **FMI for Co-Simulation**

  ![Diagram of FMI for Co-Simulation]
FMI – Main Design Idea

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

- Separation of:
  - Description of interface data (XML file)
  - Functionality (API in C)

- An FMU is a zipped file (*.fmu) containing:
  - modelDescription.xml
  - Implementation in source and/or binary form
  - Additional data and functionality

- One FMU can contain implementations of both interfaces
XML Model Description

Interface definition is stored in one xml-file:

- Implementation and capability flags
- Definition of units
- Definition of variable types
- Variables and their attributes
- Dependency information
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="fmiModelDescription.xsd"
 fmiVersion="2.0"
 modelName="FMU_Coupling.DriveTrain_TorqueAtEnd"
 guid="{a4976b5c-b9f7-432a-9dd3-e80bafaac060}"
 ...>
 <ModelExchange
   modelIdentifier="FMU_0Coupling...
   canGetAndSetFMUstate="true"
   providesPartialDerivativesOf_DerivativeFunction_wrt_States="true"
   providesDirectionalDerivatives="true"/>
 <CoSimulation
   modelIdentifier="FMU_0Coupling...
   canHandleVariableCommunicationStepSize="true"
   canInterpolateInputs="true"
   ...
 <UnitDefinitions>
   <Unit name="N.m">
     <BaseUnit kg="1" m="2" s="-2"/>
   </Unit>
 </UnitDefinitions>
 <TypeDefinitions>
   <SimpleType
     name="Modelica.SIunits.Torque">
     <Real quantity="Torque" unit="N.m"/>
   </SimpleType>
   ...
 </TypeDefinitions>
 <DefaultExperiment startTime="0.0"
     stopTime="1.0" tolerance="0.0001"/>
...</fmiModelDescription>
Example

```xml
...<ModelVariables>
  <ScalarVariable
    name="torque"
    valueReference="335544320"
    description="Torque in flange"
    causality="output">
    <Real
      declaredType="Modelica.Blocks.Interfaces.RealOutput"
      unit="N.m"/>
  </ScalarVariable>
...</ModelVariables>

<ModelStructure>
  <Inputs>
    <Input name="phi"/>
    <Input name="w" derivative="1"/>
  </Inputs>
  <Derivatives>
    <Derivative
      name="der(inertia.phi)"
      state="inertia.phi"
      stateDependencies="2"
      inputDependencies=""/>
    <Derivative
      name="der(inertia.w)"
      state="inertia.w"/>
  </Derivatives>
  <Outputs>
    <Output name="torque"
      inputDependencies="1 2"
      inputFactorKinds="fixed fixed"/>
  </Outputs>
</ModelStructure>
</fmiModelDescription>
```
C-Interface

- Instantiation:
  ```c
  fmComponent fmiInstantiateModel(fmString instanceName, ...)
  fmComponent fmiInstantiateSlave(fmString instanceName, ...)
  ```
  Returns an instance of the FMU. Returned `fmComponent` is an argument of the other interface functions.

- Functions for initialization, termination, destruction

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

- Set and Get functions for each type:
  ```c
  fmStatus fmiSetReal(fmComponent c,
                        const fmValueReference vr[], size_t nvr,
                        const fmReal value[])
  ```
  ```c
  fmStatus fmiSetInteger(fmComponent c,
                         const fmValueReference vr[], size_t nvr,
                         const fmInteger value[])
  ```

- Identification by `valueReference`, defined in the XML description file for each variable
FMI for Model Exchange
Features

- Functionality of state of the art modeling methods can be expressed
- Support of continuous-time and discrete-time systems
- Model is described by differential, algebraic, discrete equations

- Interface for solution of Ordinary Differential Equations (ODE)
- Handling of time, state and step events, event iteration

- Discarding of invalid inputs, state variables
- No explicit function call for computation of model algorithm
  - FMU decides which part is to be computed, when a `fmiGetXXX` function is called
  - Allows for efficient caching algorithms
FMI for Model Exchange
Signals

Enclosing Model

\[ t_0, p, \text{initial values (a subset of } v(t_0)) \]

External Model (FMU Instance)

Solver

- \( t \): time
- \( m \): discrete states (constant between events)
- \( p \): parameters of type Real, Integer, Boolean, String
- \( u \): inputs of type Real, Integer, Boolean, String
- \( v \): all exposed variables
- \( x \): continuous states (continuous between events)
- \( y \): outputs of type Real, Integer, Boolean, String
- \( z \): event indicators
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

Features

- **State-of-the-Art Co-Simulation:**
  - Fixed communication step size

- **To improve accuracy and robustness:**
  - Optional variable communication step size
  - Optional higher order approximation of inputs and outputs
  - Optional repetition of communication steps

- **Capabilities of the slave are contained in the XML-file, for example:**
  - canHandleVariableCommunicationStepSize
  - canInterpolateInputs
  - canGetAndSetFMUState

- **Master can decide which coupling algorithm is applicable**
- **Asynchronous execution (allows parallel execution)**
FMI for Co-Simulation Signals

Additional:
- Status information
- Derivatives of inputs, outputs w.r.t. time for support of higher order approximation
FMI for Model Exchange and Co-Simulation
Sample Code

- Model Exchange: (One model evaluation)

```c
/* Set inputs*/
fmiSetReal(m, id_u1, u1, nul);
fmiSetTime(m, tC);
fmiSetContinuousStates(m, x, nx);
/* Get results */
fmiGetDerivatives(m, derx, nx);
fmiGetEventIndicators (m, z, nz);
fmiGetReal(m, id_u1, u1, nul);
```

- Co-Simulation: (One communication step)

```c
/* Set inputs*/
fmiSetReal(s, id_u1, u1, nul);
/* Do computation*/
fmiDoStep(s, tC, hC, fmiTrue);
/* Get results */
fmiGetReal(s, id_u1, u1, nul);
```
Development Process

FMI for Model Exchange
- Berlin
- Munich
- Berlin
- Munich
- V 1.0
- Prototypes
- FMI for Co-Simulation
- Berlin
- Bonn
- Munich
- Halle
- Dresden
- Munich
- Stuttgart
- Leuven
- Munich
- Dresden
- V 1.0
- Prototypes
- FMI for Model Exchange and Co-Simulation
- Berlin
- Dresden
- Munich
- Lund
- Munich
- Munich
- Munich
- Munich
- Munich
- V 2.0 β 3
- Prototypes
- Use Cases
- Modelisar
- FMI MAP
Development Process

FMI for Model Exchange and Co-Simulation

FMI 2.0 specification:
- Release December 2012
- Valid for several years
- Backwards-compatible enhancements in minor releases
FMI 2.0
New Features

- Motivation for FMI 2.0
  - Clarification of specification document
  - Ease usability
  - Increase performance for large models

- Unification of Model Exchange and Co-Simulation Standard
  - FMU can contain implementations of both interfaces
  - Distributed and tool based use cases now also for Model Exchange

- Many minor changes
  - Definition of log categories
  - Removement of alias and anti alias variables to ease usage
  - Continuous state variables are named and ordered
  - Improved unit handling
Current status of FMI 2.0

Clarification of specification:
- Instantiation
- Classification of variables
- Calling sequence

Features:
- Tunable parameters
- Improved unit handling
- Save and restore FMU state
- Detailed dependency information (inputs, outputs, derivatives)
- Efficient interface to partial derivatives
- Improved handling of time events

*contained in public Beta 4*

*under discussion*
FMI 2.0
Classification of interface variables

causality

- parameter
- input: output of another model
- output: input for another model
- local: not to be used by other models

variability

- constant
- fixed: constant after initialization
- tunable: constant between events
- discrete: changes at event instances
- continuous

Combination of causality and variability allows clear classification of all kinds of variables

New: distinction between tunable and fixed parameters

- Stop simulation, set tunable parameters, resume simulation
FMI 2.0
Save and Restore FMU State

- FMI 1.0: implicite save and restore depending on arguments of `fmiDoStep`
- FMI 2.0: explicite function calls
  ```c
  fmiStatus fmiGetFMUstate(fmiComponent c, fmiFMUstate* FMUstate)
  fmiStatus fmiSetFMUstate(fmiComponent c, fmiFMUstate FMUstate)
  ```
- Iterative co-simulation algorithms
  - Repeat more than one communication step

- Model Predictive Control
  - Simulate some steps starting from the same state with different sets of input values
  - Use the optimal set as control value for the real system

- FMU state can be serialized into a byte vector
  - Usage: start a training simulator from a certain scenario
FMI 2.0
Dependency Information

- FMI 1.0:
  - Only dependencies of outputs on inputs can be indicated

- FMI 2.0:
  - Dependencies of outputs on continuous states
  - Dependencies of derivatives on continuous states and inputs

- Usage:
  - Detection of algebraic loops
  - Definition of sparsity pattern of Jacobian matrices
FMI 2.0
Dependency Information

- Kind of dependency is also defined:
  - nonlinear: Jacobian entry is not constant
  - fixed: Jacobian entry is constant
  - discrete: Jacobian entry may change after events

- Allows optimizations:
  - Generate linear systems of equations for solution of algebraic loops if possible
  - Reduce number of Jacobian computations
FMI 2.0
Directional Derivatives (Jacobian Matrices)

- Jacobians are needed for:
  - Implicit integration methods
  - Solution of systems of equations resulting from algebraic loops
  - Linearization of FMU
  - Extended Kalman filters

- Numerical computation is expensive for large models

- Optional function for providing directional derivatives
  
  \[
  \text{fmiStatus fmiGetDirectionalDerivative(fmiComponent c,..)}
  \]

- Arguments define which derivative(s) w.r.t. which variable(s) are to be retrieved
FMI 2.0
Time Event Handling (under Development)

Requirements:

- Guarantee synchronicity of time events
- Support a subset of the synchronous extensions from Modelica 3.3 (time triggered clocks with constant and variable period)
- Allow backward compatible extensions
- Usable for tools without synchronous features

Main design idea:

- FMU exposes base rates and clocks in the XML model description
- Clock ticking is signaled by `fmiSetClock(..)` before `fmiEventUpdate(..)`
- Discrete variables can be associated with clocks (optional) in XML model description
FMI 2.1
Hierarchical Data, Buses, Physical Connectors (planned)

Requirements:
- Group variables to hierarchical structures, connectors
- Signal based tools must not be excluded
- Keep type information of connectors (e.g. `Modelica.Electrical.Analog.Interfaces.Pin`)
- Add connector type definition for reconstruction of connector type or mapping to existing types

Main design idea:
- Additional “layer” in XML model description
- Mark input/output variables as flow or across quantities
- Causality (input, output) is fixed
Roadmap

2012:
- Finalize time event handling
- October: FMI Meeting
- November: Release of public beta 5
- December: Release of FMI 2.0
- Coordinated prototype implementations by tool vendors

2013:
- Backwards-compatible extensions
- Support of arrays and hierarchical data
- Bus and physical connectors
- Graphical appearance
- ...

© 2012 FMI Modelica Association Project | www.fmi-standard.org
FMI Support in Tools
fmi-standard.org/tools

- Tool support started immediately after release of FMI 1.0
- 32 tools support FMI, 9 intend to
- Within Modelisar project: 15
### FMI Support in Tools

- **Authoring Tools:** 12
  - (Co-Simulation master, HiL, optimization, control, analyses)

- **Integration Tools:** 20
  - (Co-Simulation master, HiL, optimization, control, analyses)

- **Software Development Kits:** 3
  - (C, Python, Java)

```plaintext
<table>
<thead>
<tr>
<th>Tools supporting FMI</th>
<th>Modal-Exchange</th>
<th>Co-Simulation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autodriver</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Bridge</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>CATIA/Simulia</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Cybermatic CEMT</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Cybermatic ModifEX</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>ControlEight</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Dataflow</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Dymola</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>Enterprise</td>
<td>available</td>
<td>available</td>
<td>planned</td>
</tr>
<tr>
<td>FMI Support in Tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2012 FMI Modelica Association Project | www.fmi-standard.org

```
Quality of FMI Implementations

FMI Compliance Checker
- Open source implementation under contract of MA
- Checks XML model description
- Simulates single FMUs for Model Exchange and Co-Simulation
- https://svn.fmi-standard.org/fmi/branches/public/Test_FMUs/FMI_1.0/Compliance-Checker/

Repository of FMUs, generated by different tools

Public Error Tracking System
- https://trac.fmi-standard.org/
Applications outside of Automotive

Power plant simulation and control
- Siemens, ABB, EDF
- EU Project MODRIO (19 Mill. €, 150 man-years, 2012 – 2015)

Building simulation
- Situation is similar to automotive industry:
  - Heterogeneous systems (building, heating, air conditioning, …)
  - Components of different nature and from several suppliers

Research
- Co-Simulation master algorithms
- Model based control
FMI Modelica Association Project (MAP)

General conditions
- FMI project members need not to be Modelica Association (MA) members
- Project results are owned by the MA
- Project results are freely usable under copyleft license
- Meetings are open to the public

FMI Steering Committee
- Defines FMI policy, strategy, feature roadmap, releases
- Voting rights

FMI Advisory Board
- Contribute to FMI design
- Access to FMI infrastructure (svn, trac, meeting minutes)
FMI Project Rules
How to participate

Steering Committee
- Prove active FMI support by participation at 2 meetings in the last 24 months
- Support FMI or part of it in a commercial or open source tool, and/or active FMI usage in industrial projects
- Be accepted by Steering Committee with qualified majority

Advisory Board
- Prove active FMI support by participation at 2 meetings in the last 24 months

Guests
- Send e-mail to contact@fmi-standard.org for registration in mailing list
FMI MAP Members

Steering Committee

- Atego, Daimler, Dassault Systèmes, IFP EN, ITI, LMS, Modelon, QTronic, Siemens, SIMPACK

Advisory Board

- Armines, DLR, Fraunhofer (IIS/EAS, First, SCAI), Open Modelica Consortium, TWT, University of Halle

Guests

- Altair Engineering, Berkeley University, Bosch, ETAS, Equa Simulation, IBM Research
Conclusions

FMI for Model Exchange and Co-Simulation is an established standard
- 32 tools currently support FMI 1.0, 9 intend to
- Is used in industrial and research applications
- Is maintained as Modelica Association Project

FMI project is open for non Modelica tool vendors and organizations

FMI 2.0 improves:
- Compatibility of implementations (clarified specification)
- Usability (tunable parameters, unit handling)
- Efficiency and robustness for large models (dependency information, directional derivatives)
Outlook

FMI 2.0 Release planned for December 2012

Current tasks:
- Precise handling of time events for periodic and aperiodic sampled data systems

Ideas for FMI 2.1
- Arrays, hierarchical data, buses, physical ports
- Graphical appearance, connector placement