The Functional Mockup Interface for Tool independent Exchange of Simulation Models

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

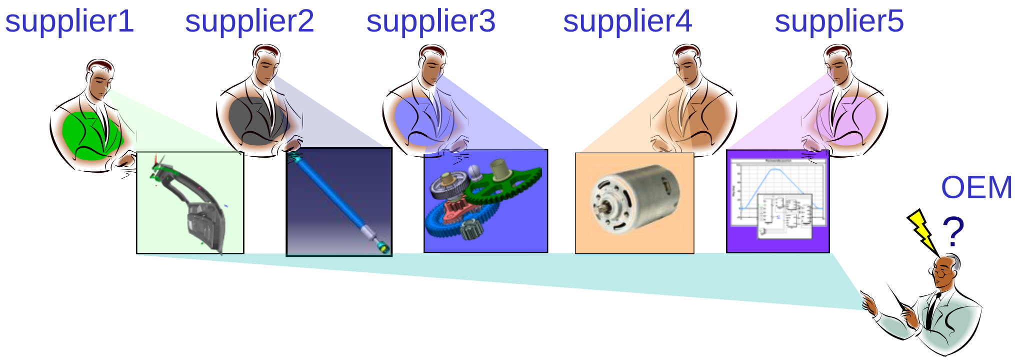
M. Arnold	University of Halle
C. Bausch, M. Monteiro	Atego Systems GmbH
C. Clauß, S. Wolf	Fraunhofer IIS EAS, Dresden
H. Elmqvist, H. Olsson	Dassault Systèmes, Lund
A. Junghanns, J. Mauss	QTronic, Berlin
T. Neidhold	ITI, Dresden,
D. Neumerkel	Daimler AG, Stuttgart
J.-V. Peetz	Fraunhofer SCAI, St. Augustin

Modelica 2011, 21st,22nd March

Functional Mock-up Interface (FMI) - Motivation (1)

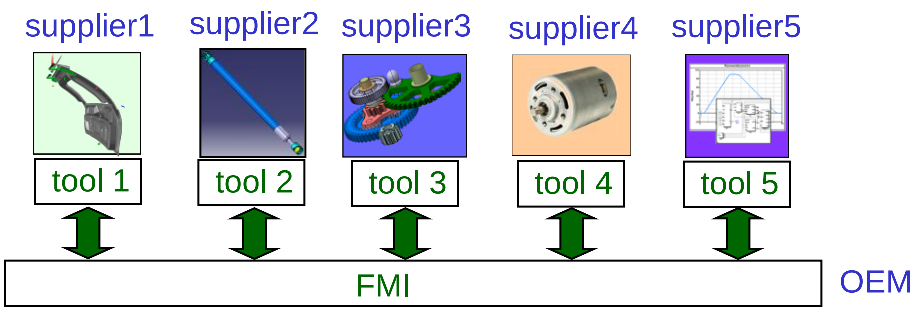
Problems / Needs

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



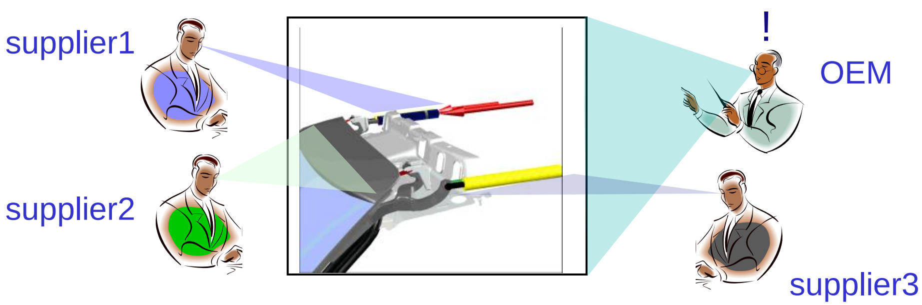
Solution

- 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

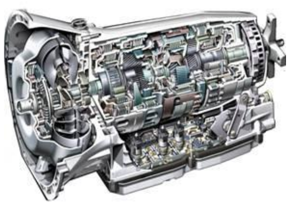
- Lots of proprietary interfaces:
  - Simulink: S-function
  - Modelica: external function, external object interface
  - QTronic Silver: Silver-Module API
  - SimulationX: External Model Interface
  - NI LabVIEW: External Model Interface, Simulation Interface Toolkit
  - Simpack: uforce routines
  - ADAMS: user routines
  - …

FMI – Overview

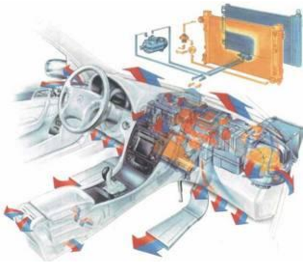
The FMI development is part of the ITEA2 MODELISAR project
(2008 - 2011; 29 partners, Budget: 30 Mill. €)

- 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

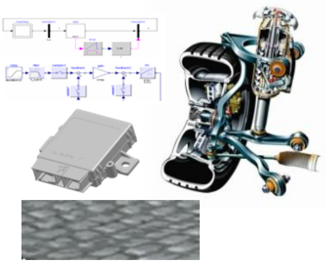


Gearbox
with ECU



Thermal
systems

Automated
cargo door



Chassis components,
roadway, ECU (e.g. ESP)

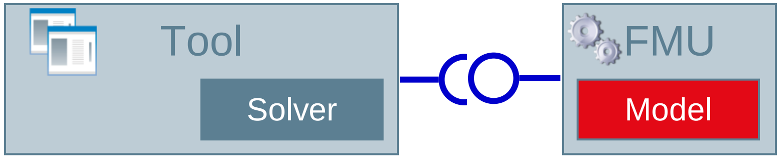
etc.

functional mockup interface for model exchange and tool coupling

courtesy Daimler

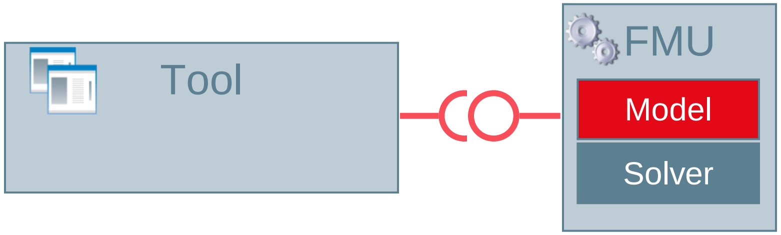
FMI - Main Design Idea (1)

- FMI for Model Exchange:



  - Version 1.0 released in January 2010

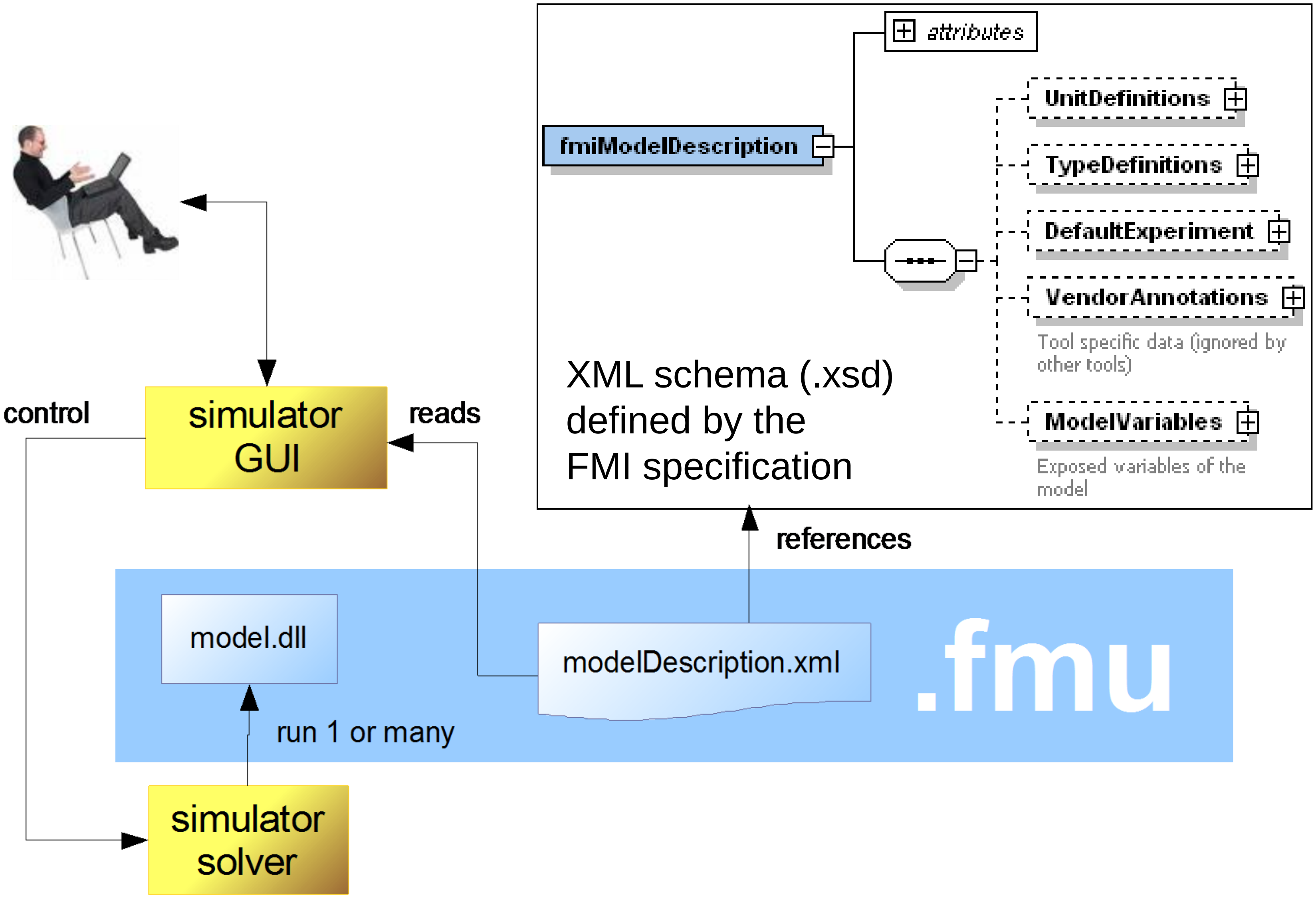
- FMI for Co-Simulation:
  - Reuses as much as possible from FMI for Model Exchange standard



  - Version 1.0 released in October 2010

FMI - Main Design Idea (2)

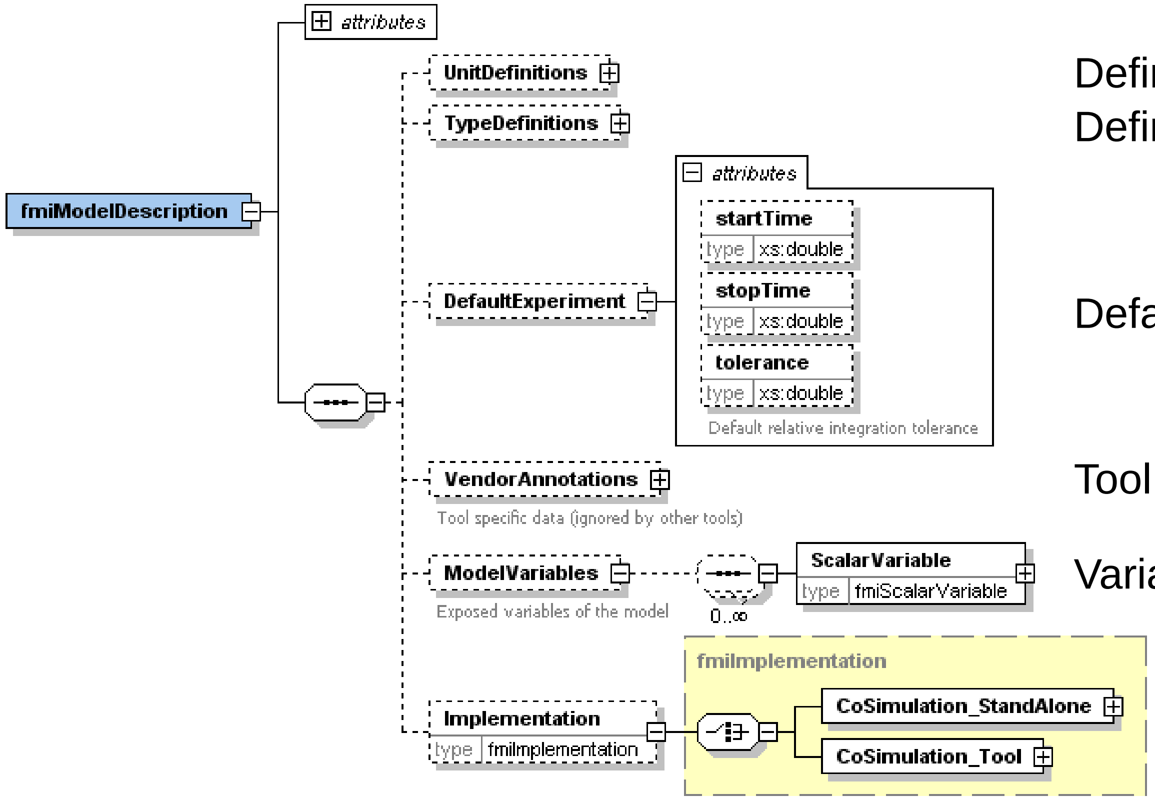
- A component which implements the interface is called *Functional Mockup Unit (FMU)*
- Separation of
  - Description of interface data (XML file)
  - Functionality (C code or binary)
- A FMU is a zipped file (*.fmu) containing the XML description file and the implementation in source or binary form
- Additional data and functionality can be included
- Interface specification: www.functional-mockup-interface.org



attributes
UnitDefinitions
fmiModelDescription
TypeDefinitions
DefaultExperiment
VendorAnnotations
Tool specific data (ignored by other tools)
XML schema (.xsd) defined by the FMI specification
ModelVariables
Exposed variables of the model
control
simulator GUI
reads
references
model.dll
modelDescription.xml
.fmu
run 1 or many
simulator solver

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.



Definition of display units

Definition of type defaults

Default stop time, tol. etc.

Tool specific data

Variable names and attributes

Example

modelDescription.xml


```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>
```

C-Interface

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

C-Interface

- Instantiation:

```
fmiComponent fmiInstantiateXXX(fmiString instanceName, ...)
```

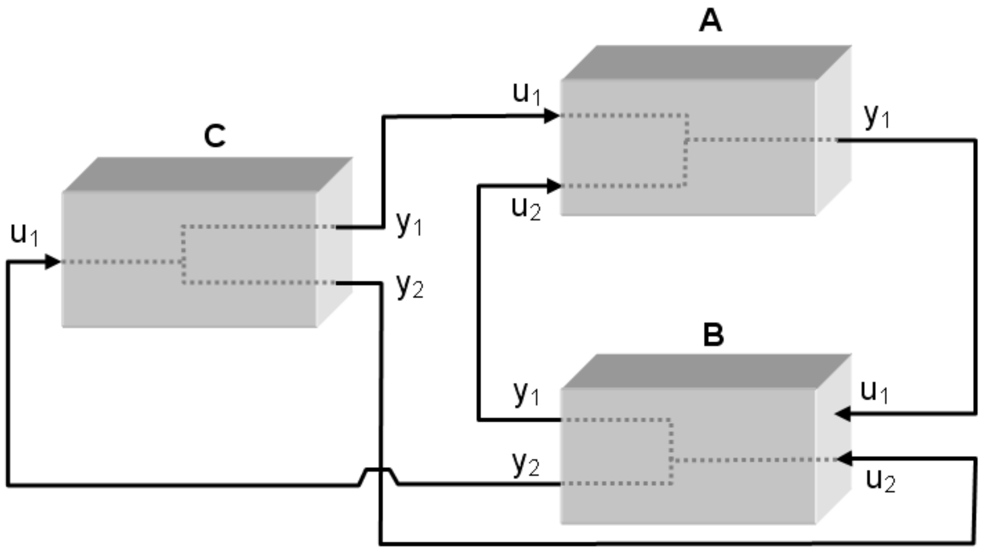
- Returns an instance of the FMU. Returned `fmiComponent` is a parameter of the other interface functions. It is of type `void*` for the master. The FMU uses it to hold all necessary information.
- Functions for initialization, termination, destruction
- Support of real, integer, boolean, and string inputs, outputs, parameters
- `Set` and `Get` functions for each type:

```
fmiStatus fmiSetReal    (fmiComponent c,
                         const fmiValueReference vr[], size_t nvr,
                         const fmiReal value[])
fmiStatus fmiSetInteger(fmiComponent c,
                         const fmiValueReference vr[], size_t nvr,
                         const fmiInteger value[])
```

- Identification by `valueReference`, defined in the XML description file for each variable

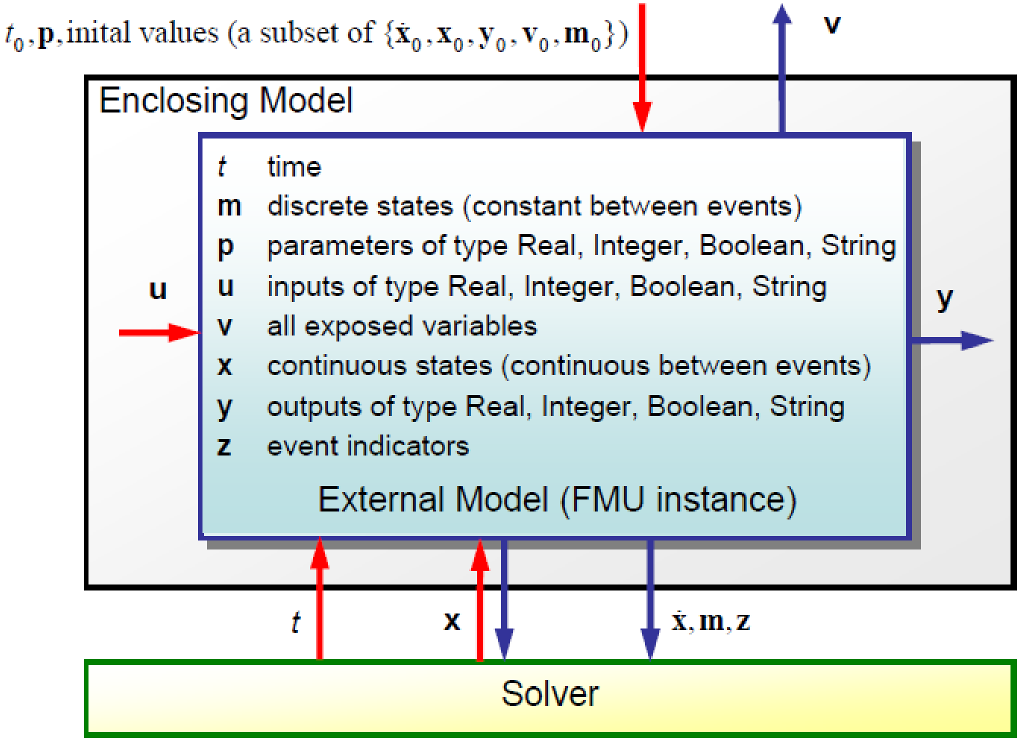
FMI for Model Exchange (1)

- 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

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 $\dot{\mathbf{x}}(t)$	$t_i \leq t < t_{i+1}$	$\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x},\mathbf{m},\mathbf{u},\mathbf{p},t)$	`fmiGetDerivatives`
outputs $\mathbf{y}(t)$	$t_i \leq t < t_{i+1}$	$\mathbf{y} = \mathbf{f}_y(\mathbf{x},\mathbf{m},\mathbf{u},\mathbf{p},t)$	`fmiGetReal/Integer/Boolean/String`
internal variables $\mathbf{v}(t)$	$t_i \leq t < t_{i+1}$	$\mathbf{v} = \mathbf{f}_v(\mathbf{x},\mathbf{m},\mathbf{u},\mathbf{p},t)$	`fmiGetReal/Integer/Boolean/String`
event indicators $\mathbf{z}(t)$	$t_i \leq t < t_{i+1}$	$\mathbf{z} = \mathbf{f}_z(\mathbf{x},\mathbf{m},\mathbf{u},\mathbf{p},t)$	`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:

```
// 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

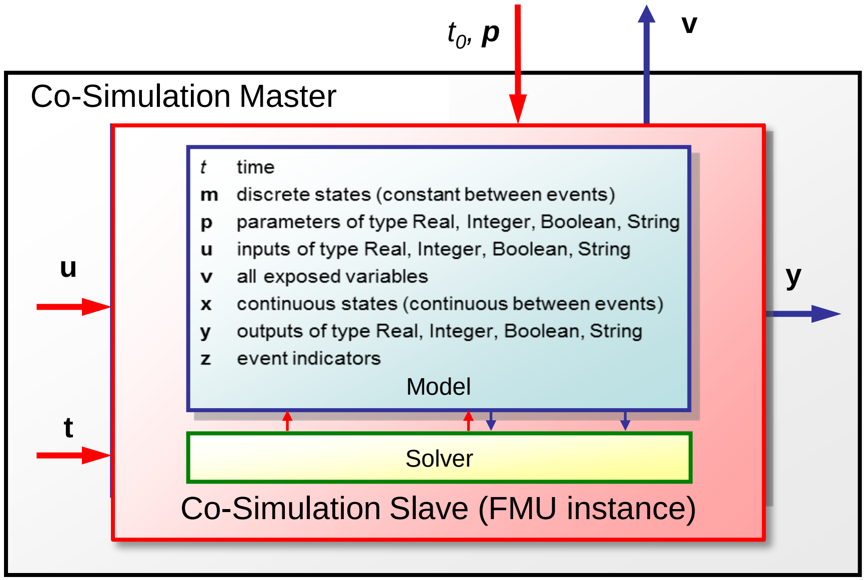
FMI for Co-Simulation

- Master/slave architecture
- Considers different capabilities of simulation tools
- Support of simple and sophisticated coupling algorithms:
  - Iterative and straight forward 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



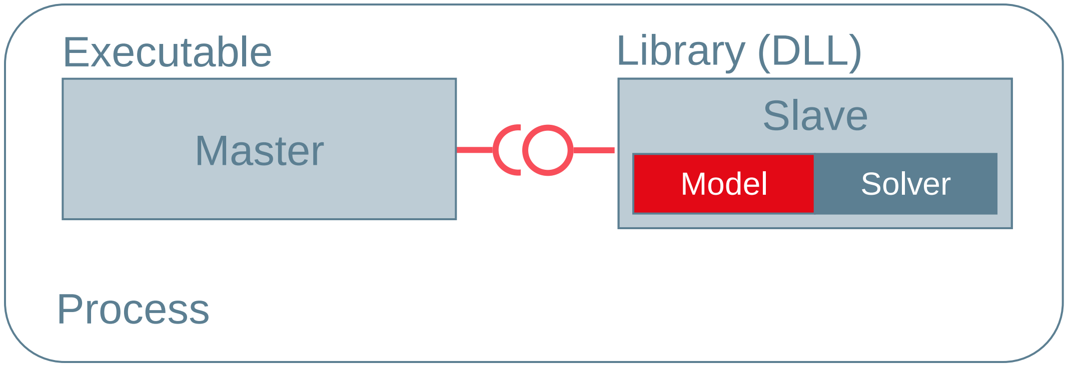
  - Inputs, outputs, and parameters, status information
  - Derivatives of inputs, outputs w.r.t. time can be set/retreived for supporting of higher order approximation

FMI for Co-Simulation
C-Interface

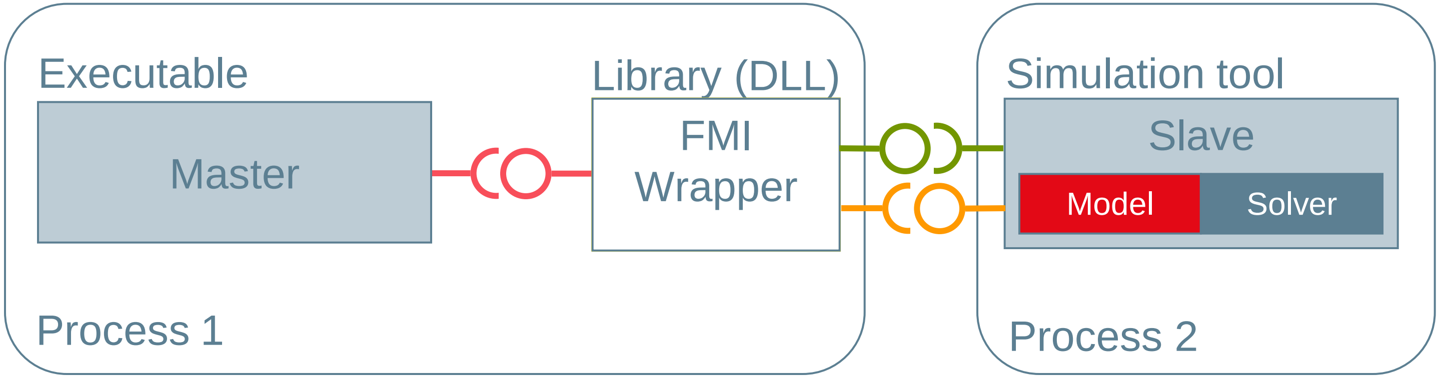
- Execution of a time step:
```
fmiStatus fmiDoStep(fmiComponent c,
    fmiReal currentCommunicationPoint,
    fmiReal communicationStepSize, fmiBoolean newStep)
```
- `communicationStepSize` can be zero in case of event iteration
- `newStep = fmiTrue` if last step was accepted by the master
- It depends 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:

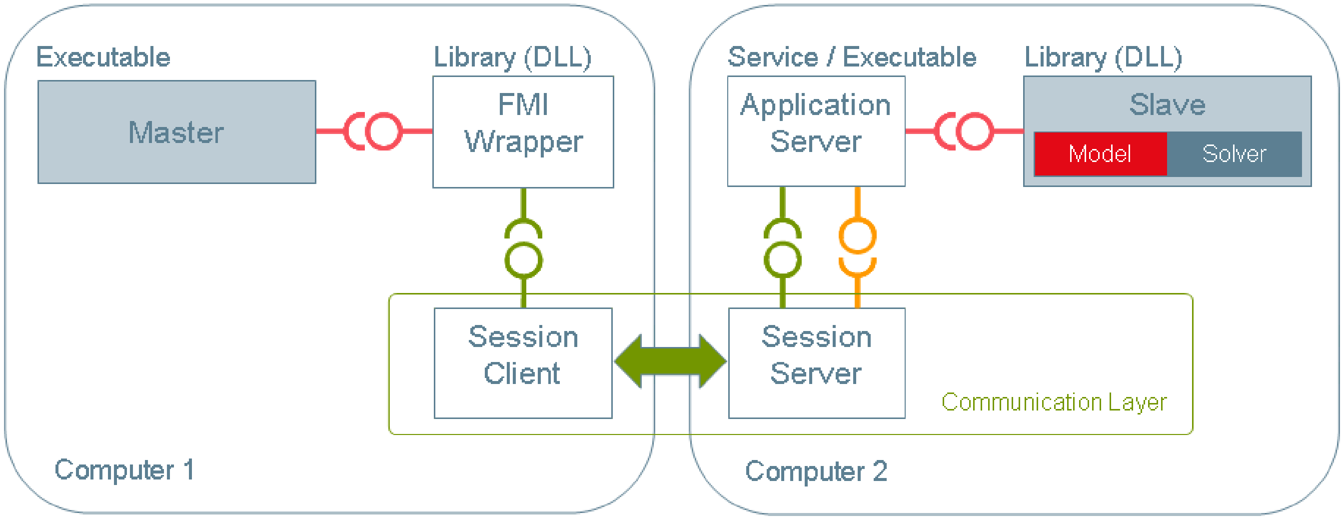


- Co-Simulation tool:



FMI for Co-Simulation
Use Case

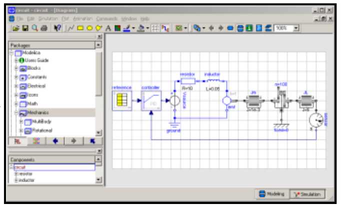
- Distributed co-simulation scenario

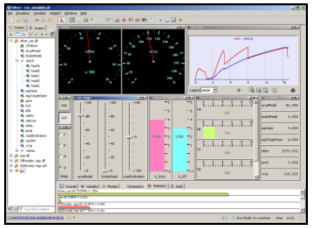


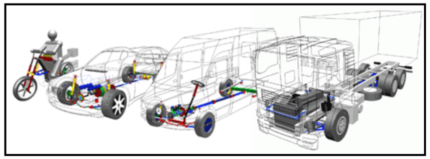
- Data exchange is handled by a communication layer which is implemented by a special FMI wrapper
- Master and slave utilize FMI for Co-Simulation only

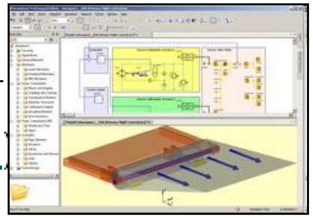
Tools supporting FMI (from FMI web site, March 2011)

AMESim	(FMU export and import)
Dymola 7.4	(FMU export and import; now available)
EXITE ACE	(FMU export and import)
EXITE	(FMU import)
FMU SDK	(FMU export and import; now available)
JModelica.org	(FMU export and import; now available)
NI VeriStand	(FMU Co-Simulation)
NI LabVIEW	(FMU import)
Silver 2.0	(FMU import; now available)
SIMPACK	(FMU import)
SimulationX 3.4	(FMU export and import; FMU Co-Simulation; now available)
Simulink	(FMU export now available by Dymola 7.4 via Real-Time Workshop)
Simulink	(FMU export now available via @Source)
TISC	(FMU import)









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