

---

# Functional Mockup Interface (FMI)



The leading standard to exchange dynamic simulation models

The Functional Mock-up Interface is a free standard that defines a container and an interface to exchange dynamic simulation models using a combination of XML files, binaries and C code, distributed as a ZIP file. It is supported by [170+ tools](#) and maintained as a Modelica Association Project.

[▶ Why FMI](#)

[📄 Complete Package 3.0](#) ▼

[📄 Specification 3.0](#) ▼

[📄 Implementers' Guide](#)

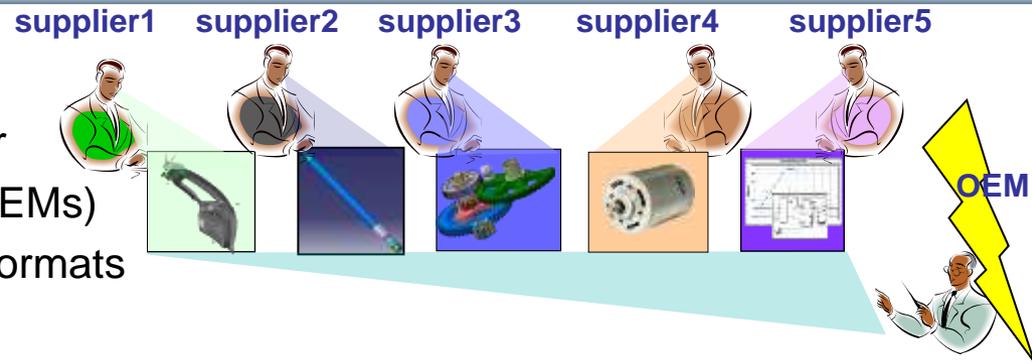
# FMI – Motivation 1

- Need to SOLVE **large integrated** modeling and simulation engineering **problems**
- Hundreds of simulation tools, **different model formats**
- **Exchange** dynamic **models** between different tools and define tool coupling for dynamic system simulation environments.
- *Two main approaches:*
  - 1. **Export** models from some tools, **import** into other tools for **simulation**
  - 2. **Co-simulation** of models in different tools
- Implementation Package Format: **Functional Mockup Unit (FMU)**
- Solution: Functional Mockup Interface (FMI) standard  
[www.fmi-standard.org](http://www.fmi-standard.org)

# FMI – Motivation 2

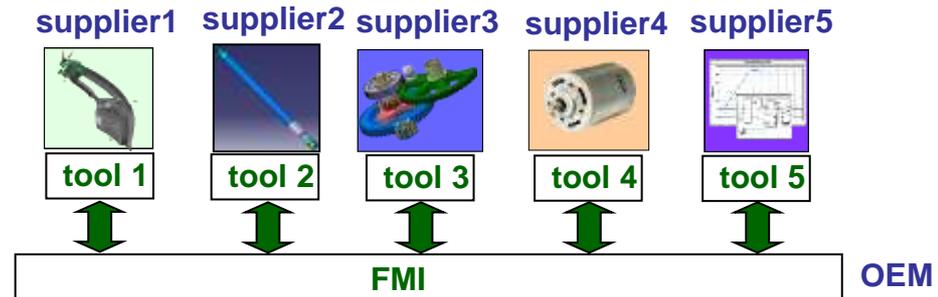
## Problems / Needs

- Component development by supplier
- Integration by product integrators (OEMs)
- Many different simulation tools and formats



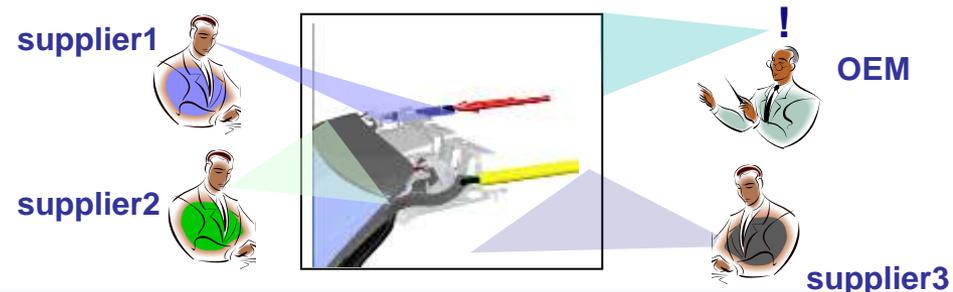
## Solution

- Reuse of supplier models by Product integrator companies (OEMs):
  - Binary (DLL) (model import) and/or
  - Tool coupling (co-simulation)
- Protection of model IP (Intellectual Property) of supplier since binary models



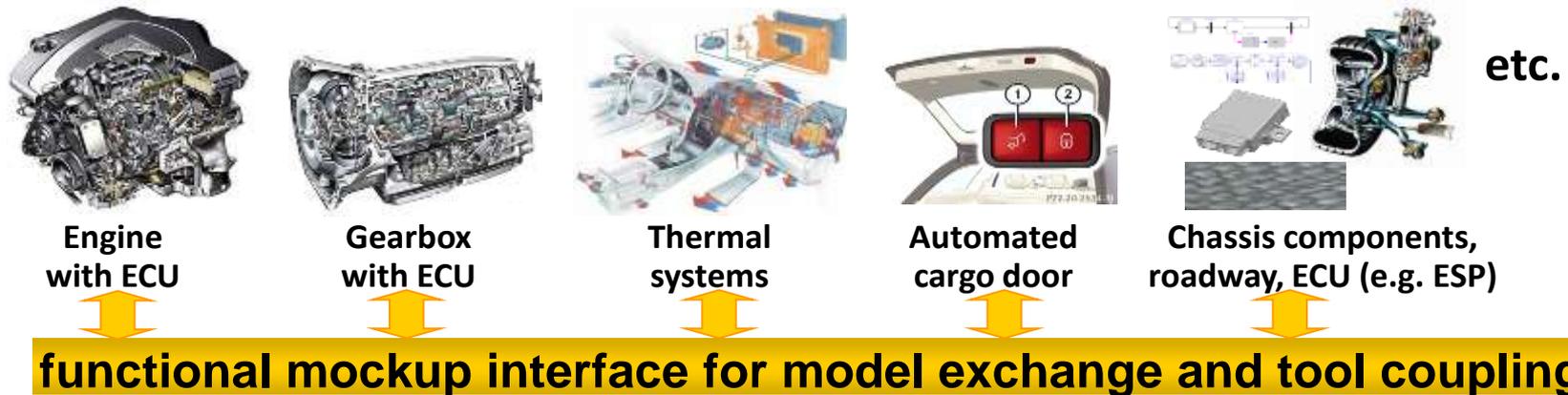
## Added Value

- Early validation of design
- Increased process efficiency and quality



# General Tool Interoperability & Model Exchange

## Functional Mock-up Interface (FMI)



courtesy Daimler

- FMI development was started by ITEA2 MODELISAR project. FMI is now a Modelica Association Project
- **Version 1.0**
- FMI for Model Exchange (released Jan 26,2010)
- FMI for Co-Simulation (released Oct 12,2010)
- **Version 2.0** (released July 25 2014) **2.0.4** (released Dec 1, 2022)
- **Version 3.0** (released May 10 2022)
- FMI for Model Exchange and Co-Simulation
- ~ **180 tools** supporting it (<https://www.fmi-standard.org/tools>)

# FMI 3.0 Enhancements

See <https://fmi-standard.org/docs/3.0/> for more info

## Ports and Icons

Help the user to build consistent systems from FMUs and render the systems more intuitively with better representation of structured ports (for instance busses and physical connectors) in the modelDescription.xml.

## Array variables

Allow FMUs to communicate multi-dimensional variables and change their sizes using structural parameters.

## Clocks and Hybrid Co-Simulation

Introduces clocks for synchronization of variables changes across FMUs. Allows co-simulation with events.

## Binary Data Type

Adds an opaque binary data type to FMU variables to allow, for instance, efficiently exchanging of complex sensor data.

## Intermediate Variable Access

Allow access to intermediate input and output values between communication time points from the FMU to disclose relevant subsystem behavior for analysis or advanced co-simulation master algorithms for enhanced numerical stability.

## Source code FMUs

Adding more information to the modelDescription.xml file to improve automatic import of source code FMUs.

## Numeric Variable Types

Adds 8, 16, 32 and 64-bit signed and unsigned integer and single precision floating point variable types to improve efficiency and type safety when importing / exporting models from the embedded, control and automotive domains.

## Extra directory

Adding a new folder in the ZIP Archive representing an FMU, providing additional data to travel with the FMU which can be modified by different tools, allowing for layered standards

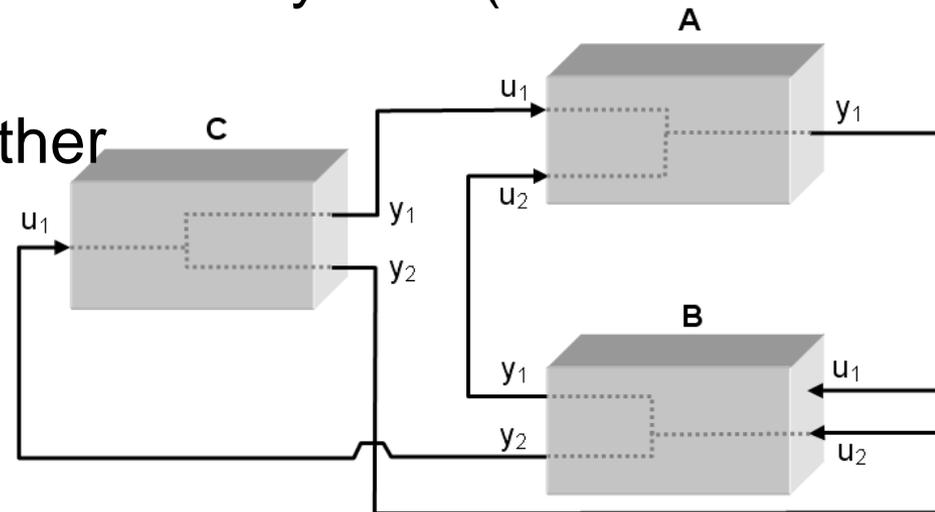
# Exported Model in (Functional Mockup Unit) FMU Form

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

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

# Functional Mockup Units

- Import and export of input/output blocks – **Functional Mock-Up Units – FMUs**
- 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



# 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 preferred language (C++, C#, Java, ...)

- **Model equations** defined by a small set of **C-functions**. In zip-file:

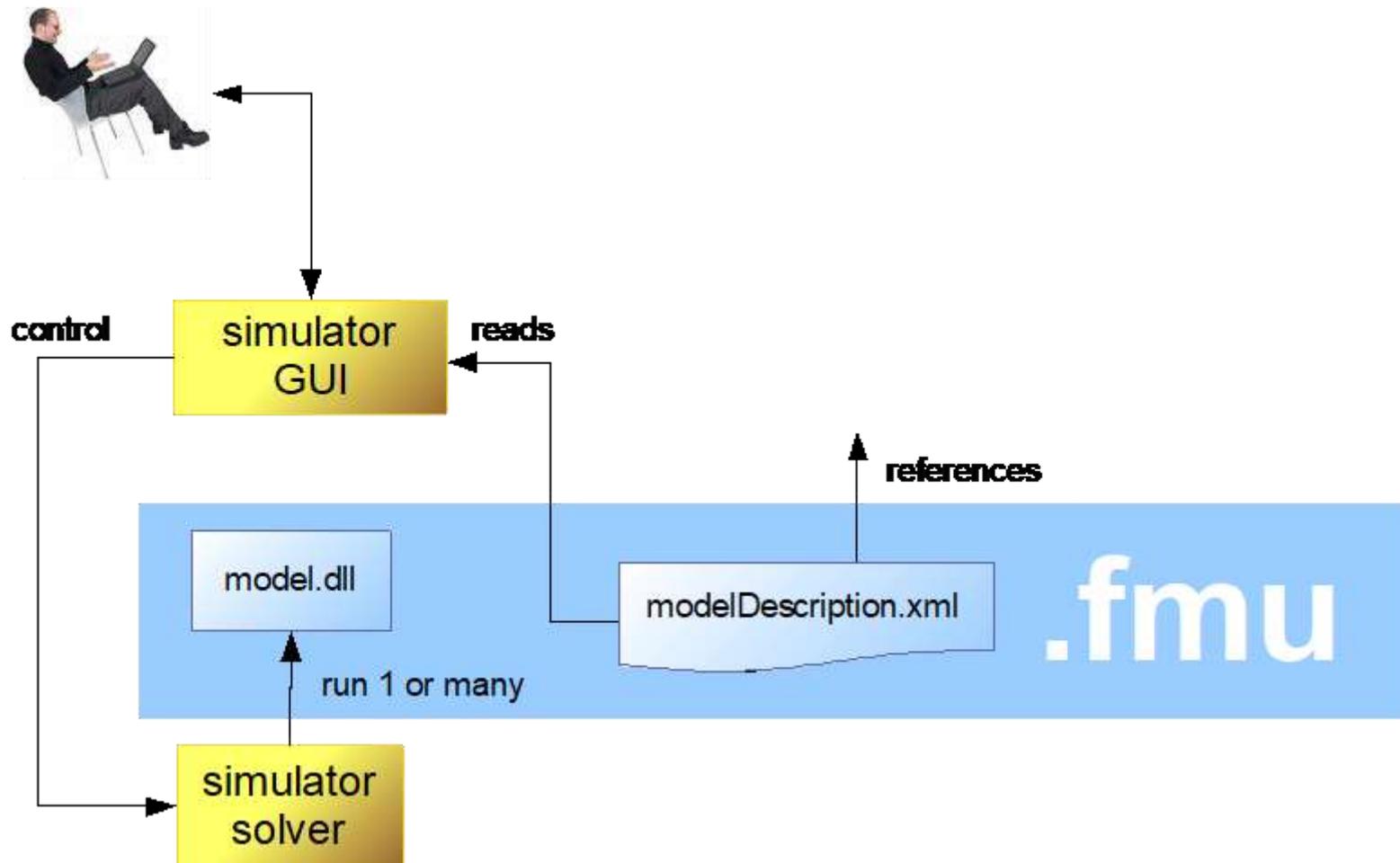
- **C source code** and/or
- **Binary code** (DLL) for one or more platforms (Windows, Linux, ...)

- **Resources**

- Documentation (html files)
- Model icon (bitmap file)
- Maps and tables (read by model during initialization)

# Simulator with GUI and Solver

## Executing Imported Model = FMU (Functional Mockup Unit)

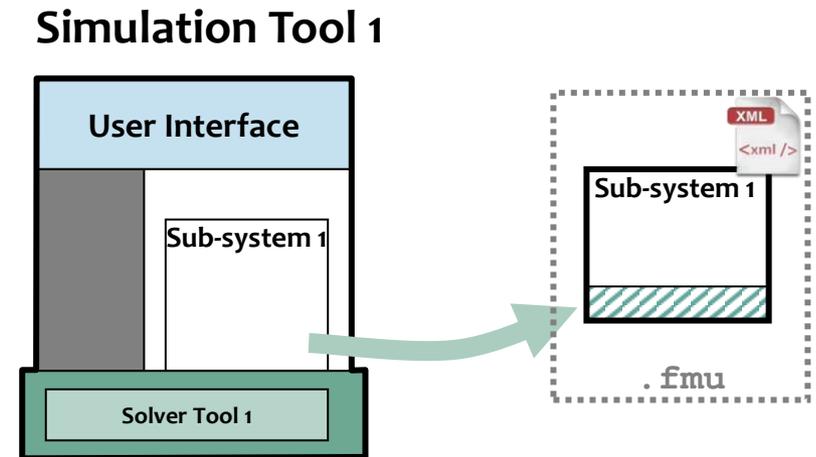


---

# FMI for Model-Exchange

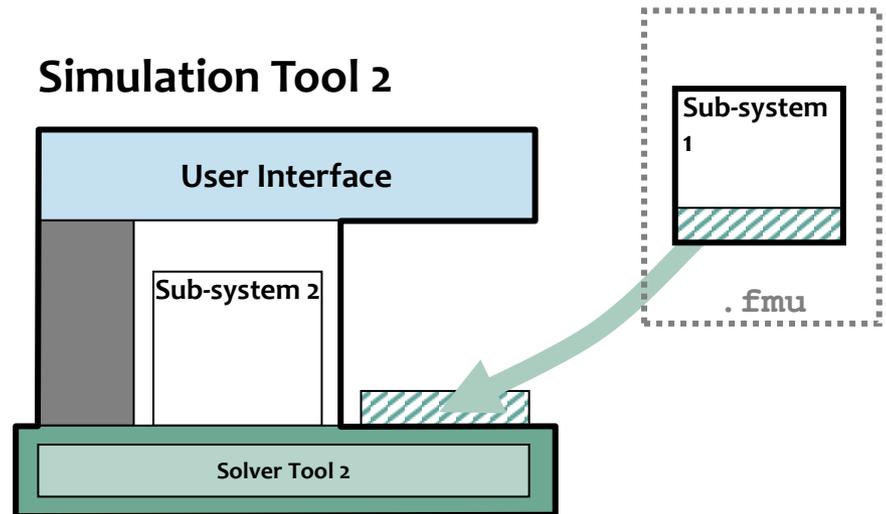
# FMI for Model Exchange Export

- **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

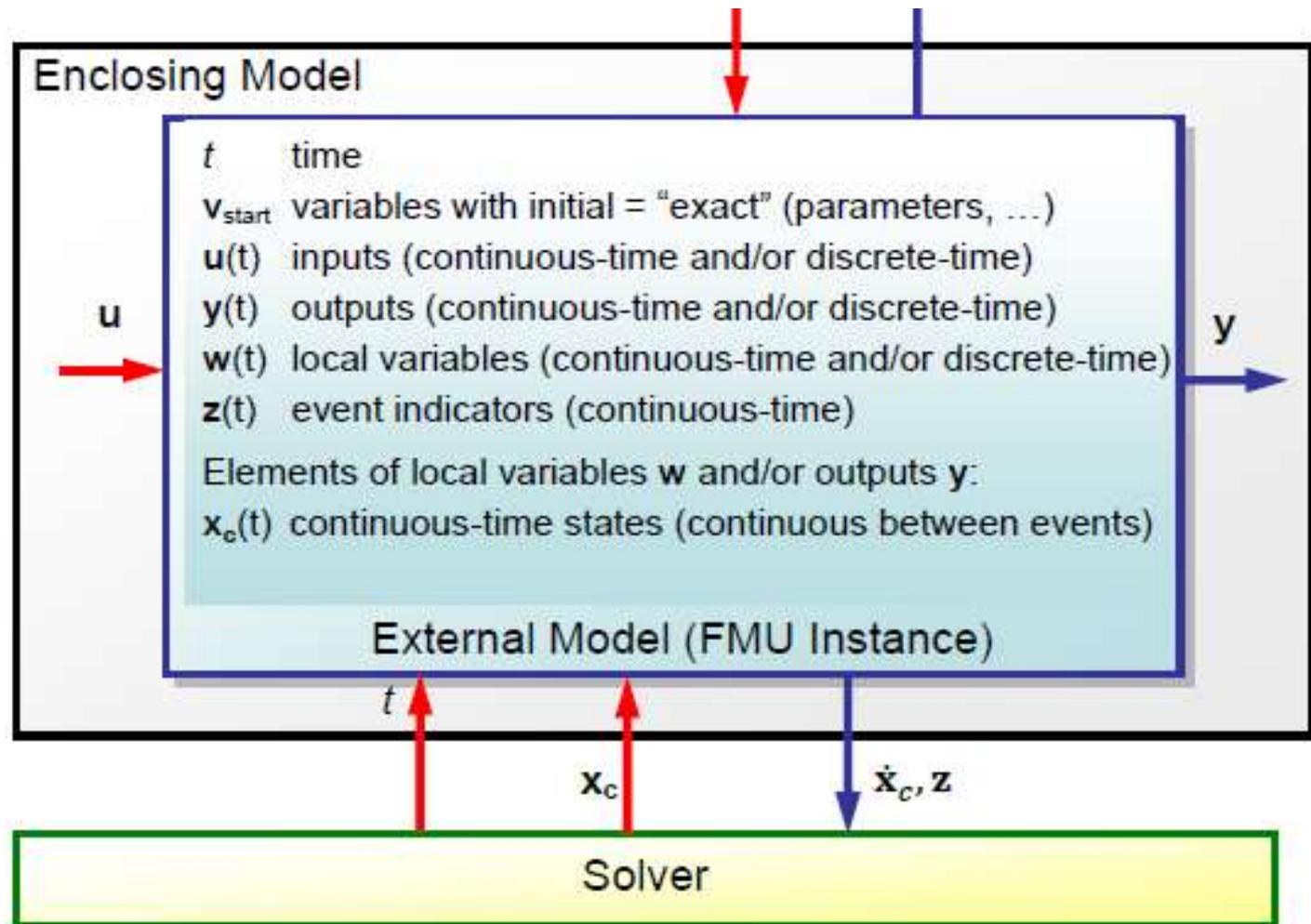


# FMI for Model Exchange Import

- **Import: Subsystem model is imported** into simulation system for **system simulation**
  - Reading FMU-archive
    - model information from xml-file
    - connecting subsystem variables
    - executable model equations (dll)
    - running system simulation



# FMI for Model Exchange (contd.)

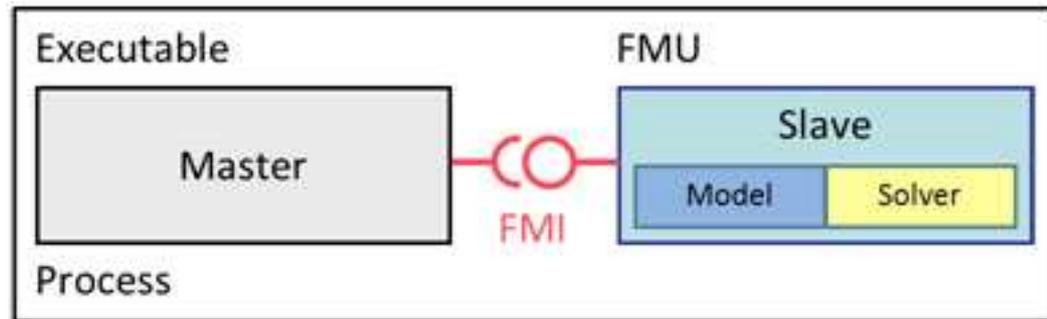


---

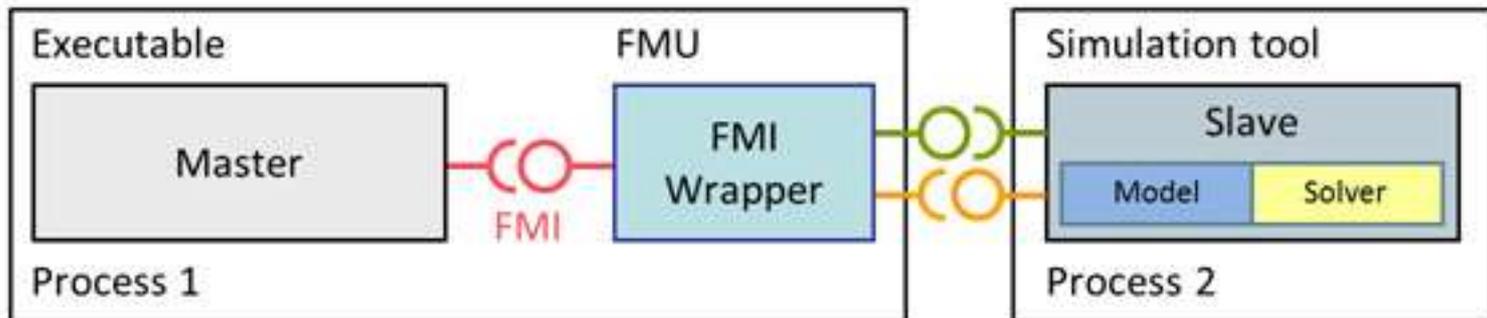
# FMI for Co-Simulation

# FMI for Co-Simulation

- Its been designed both for coupling with subsystem models, which have been exported by their simulators together with its solvers as runnable code,



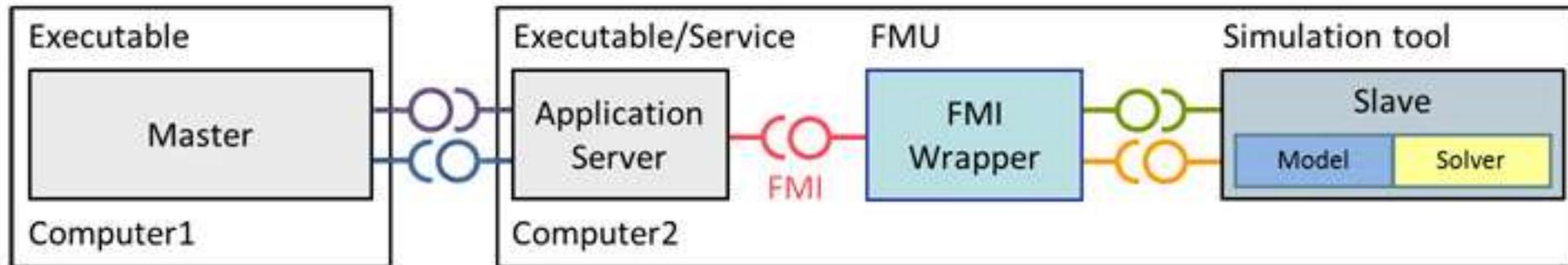
- And for coupling of simulation tools,



# FMI for Co-Simulation

- Distributed Co-Simulation Scenario

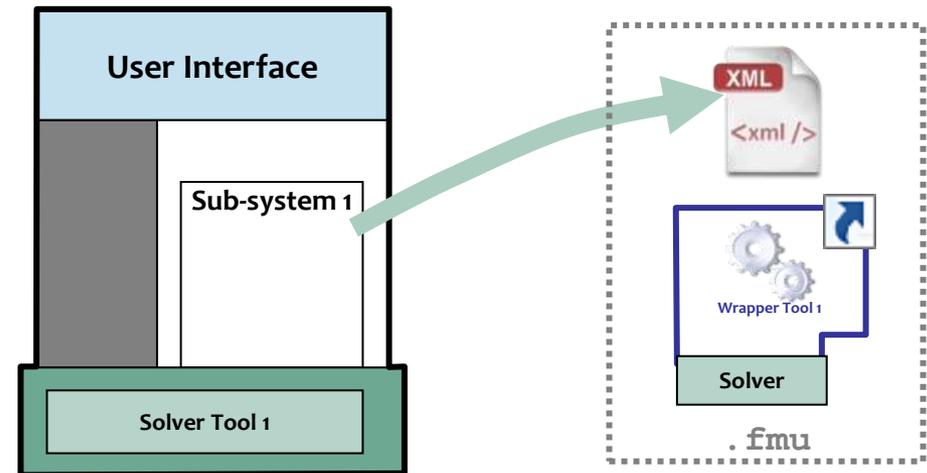
- Data exchange is handled by some network communication technology.
- Communication layer not part of the FMI standard.
- Master is responsible for the communication layer implementation.



# FMI for Co-Simulation Export FMU with Solver

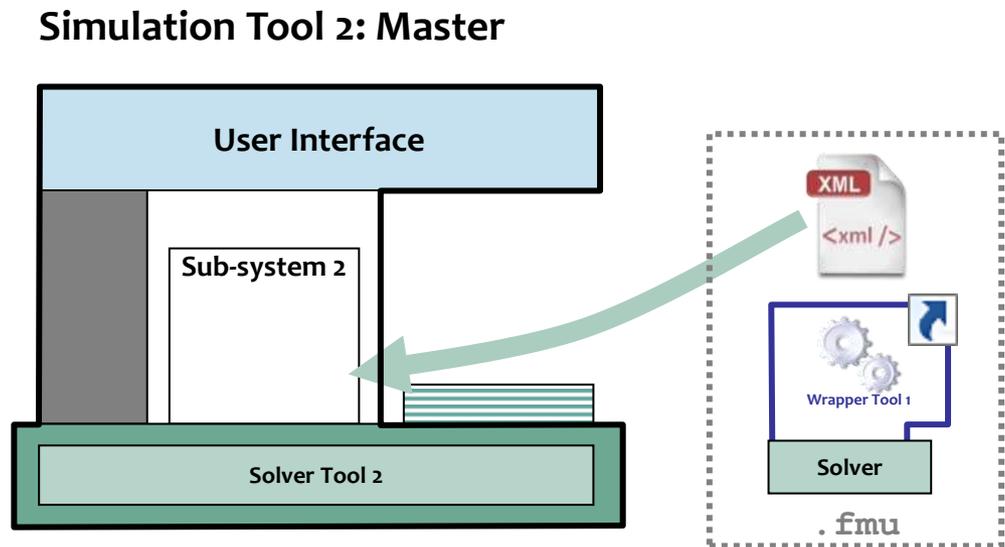
- **Export: Subsystem description** is exported from its simulation tool
  - Preparation as FMU-archive containing
    - model description (xml-file), describes also solver/tool capabilities
    - reference to executable dll-file as, wrapper which provides a tool specific implementation of the co-simulation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

Simulation Tool 1: Slave



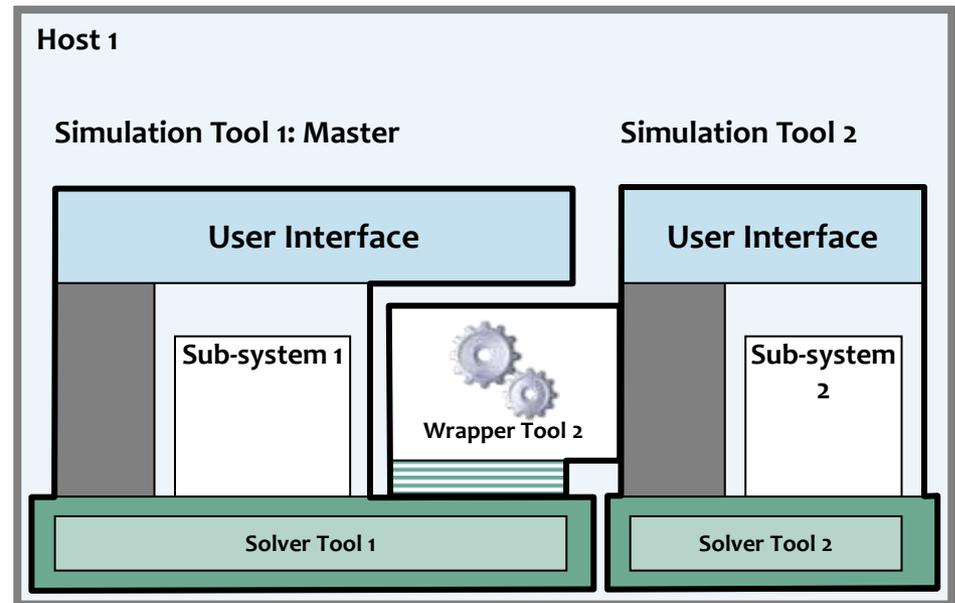
# FMI for Co-Simulation Import Stand-alone

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



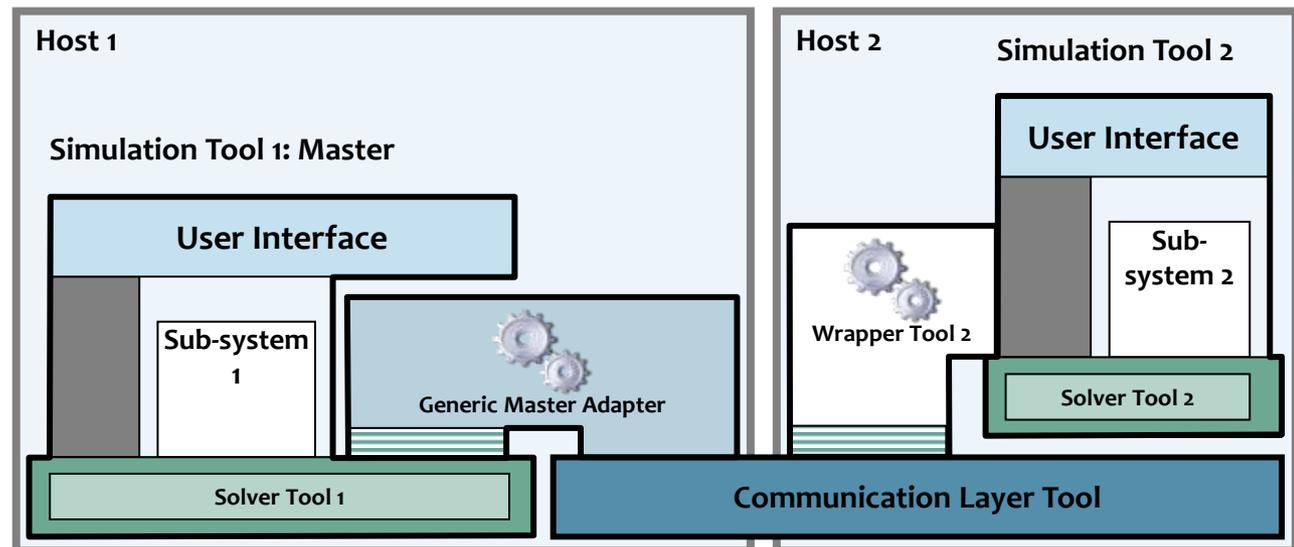
# FMI for Co-Simulation Tool coupling

- **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

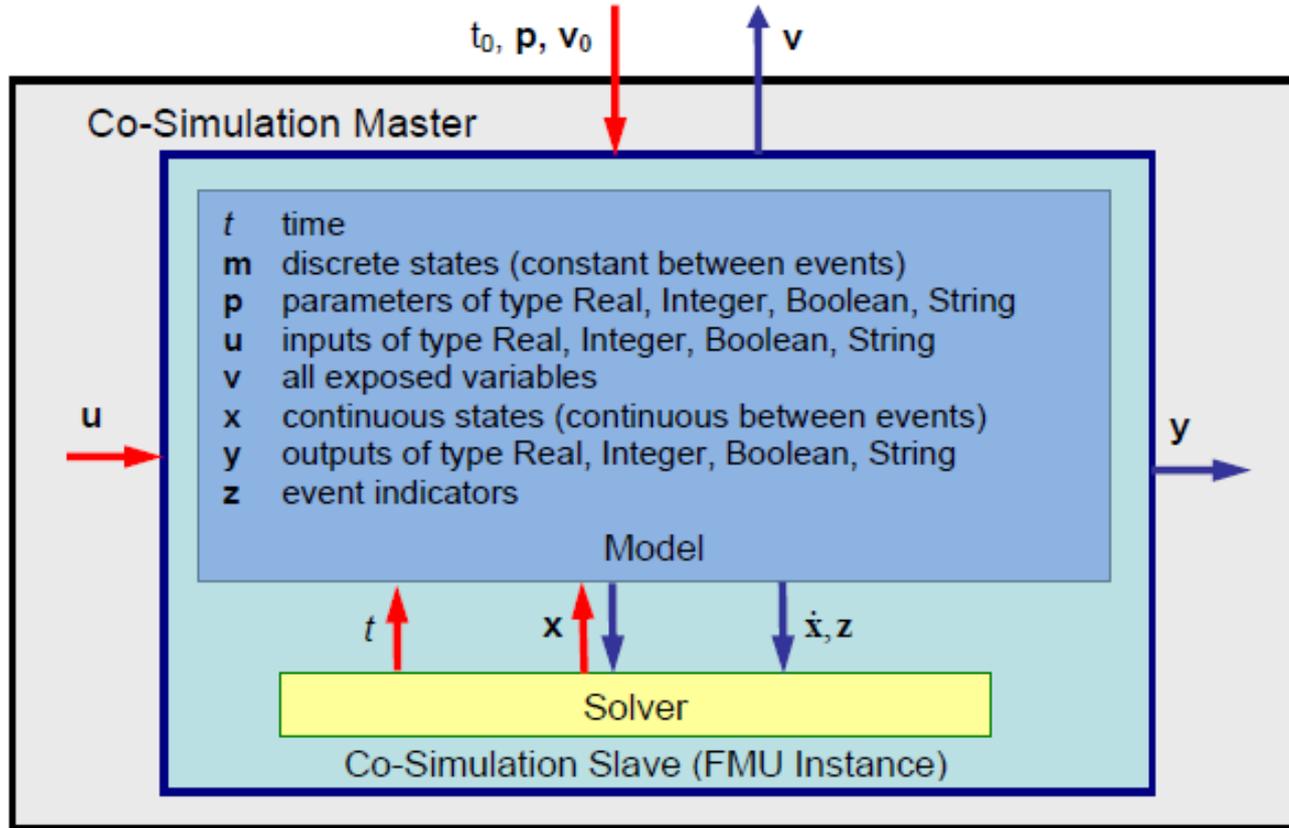


# FMI for Co-Simulation distributed tool coupling

- **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



# FMI for Co-Simulation



---

# FMU

## Functional Mockup Unit in more detail

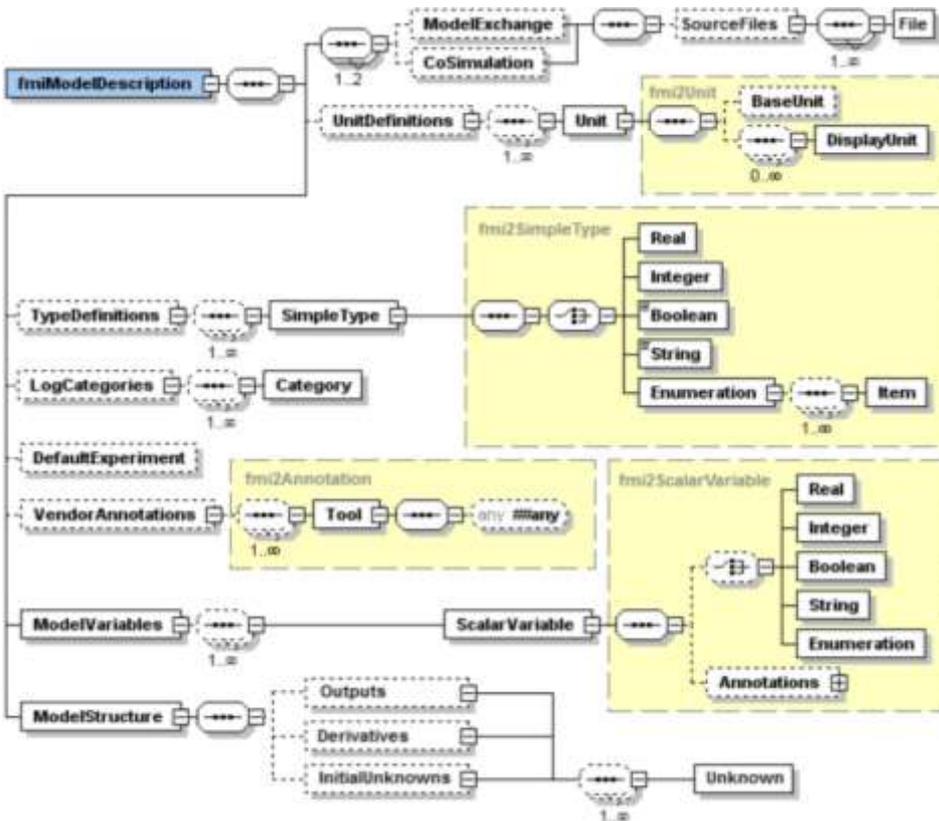
# Structure of an FMU zip-file

```
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 >
```

# Example Model Description XML File

```
<?xml version="1.0" encoding="UTF8"?>
<fmiModelDescription
  fmiVersion="2.0"
  modelName="Modelica.Mechanics.Rotational.Examples.Friction"
  modelIdentifier="Modelica_Mechanics_Rotational_Examples_Friction"
  guid="{8c4e810f-3df3-4a00-8276-176fa3c9f9e0}"
  ...
  numberOfEventIndicators="34"/>
<UnitDefinitions>
  <Unit name="rad">
    <BaseUnit rad="1"/>
    <DisplayUnit name="deg" factor="57.2957795130823"/>
  </Unit>
</UnitDefinitions>
<TypeDefinitions>
  <SimpleType name="Modelica.SIunits.Inertia">
    <Real quantity="MomentOfInertia" unit="kg.m2" min="0.0"/>
  </SimpleType>
</TypeDefinitions>
<ModelVariables>
  <ScalarVariable
    name="inertial.J"
    valueReference="1073741824"
    description="Moment of load inertia"
    causality="parameter"
    variability="fixed">
    <Real declaredType="Modelica.SIunits.Inertia" start="1"/>
  </ScalarVariable> <!--index="1" -->
  ...
</ModelVariables>
</fmiModelDescription>
```

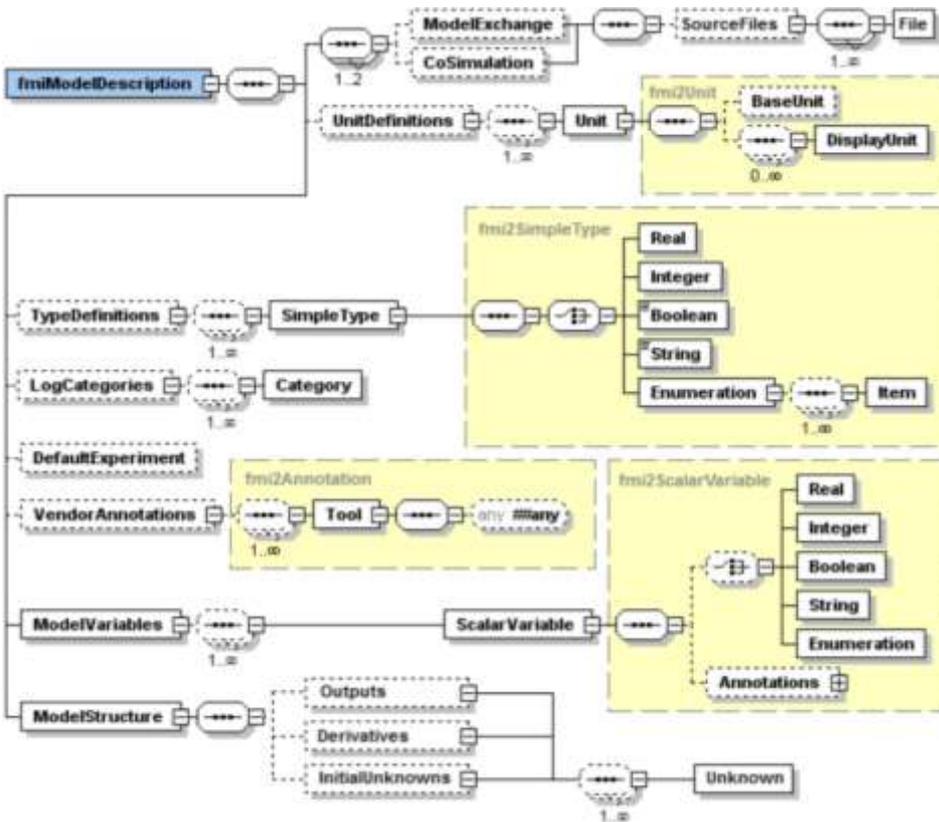
# Inspecting an FMU



```

Ubuntu 18.04 LTS
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  fmiVersion="2.0"
  modelName="DualMassOscillator_ReferenceSystem"
  guid="{046262a0-9e88-485d-b9bf-5aba0fdb6222}"
  description=""
  generationTool="OpenModelica Compiler OMCompiler v1.16.0-dev.193+g693c8881d"
  generationDateAndTime="2020-02-25T16:43:18Z"
  variableNamingConvention="structured"
  numberOfEventIndicators="0">
  <ModelExchange
    modelIdentifier="DualMassOscillator_ReferenceSystem">
    <SourceFiles>
      <File name="DualMassOscillator_ReferenceSystem.c" />
      <File name="DualMassOscillator_ReferenceSystem_functions.c" />
      <File name="DualMassOscillator_ReferenceSystem_records.c" />
      <File name="DualMassOscillator_ReferenceSystem_01exo.c" />
      <File name="DualMassOscillator_ReferenceSystem_02nls.c" />
      <File name="DualMassOscillator_ReferenceSystem_03lsy.c" />
      <File name="DualMassOscillator_ReferenceSystem_04set.c" />
      <File name="DualMassOscillator_ReferenceSystem_05evt.c" />
      <File name="DualMassOscillator_ReferenceSystem_06ins.c" />
      <File name="DualMassOscillator_ReferenceSystem_07diy.c" />
      <File name="DualMassOscillator_ReferenceSystem_08bnd.c" />
      <File name="DualMassOscillator_ReferenceSystem_09alg.c" />
      <File name="DualMassOscillator_ReferenceSystem_10asr.c" />
      <File name="DualMassOscillator_ReferenceSystem_11mix.c" />
      <File name="DualMassOscillator_ReferenceSystem_12jac.c" />
    </SourceFiles>
  </ModelExchange>
</fmiModelDescription>
  
```

# Inspecting an FMU



- ModelExchange
- CoSimulation
- UnitDefinitions
- TypeDefinition
- LogCategories
- DefaultExperiment
- VendorAnnotations
- **ModelVariables**
- **ModelStructure**

# Inspecting an FMU – Model Variables

```
Ubuntu 18.04 LTS
<ModelVariables>
<!-- Index of variable = "1" -->
<ScalarVariable
  name="v"
  valueReference="0"
  initial="exact"
  <Real start="0.0"/>
</ScalarVariable>
<!-- Index of variable = "2" -->
<ScalarVariable
  name="x"
  valueReference="1"
  >
  <Real/>
</ScalarVariable>
<!-- Index of variable = "3" -->
<ScalarVariable
  name="der(v)"
  valueReference="2"
  >
  <Real derivative="1"/>
</ScalarVariable>
<!-- Index of variable = "4" -->
<ScalarVariable
  name="der(x)"
  valueReference="3"
  >
  <Real derivative="2"/>
</ScalarVariable>
```

- index: implicitly defined by order of variables in the xml file
- name
- valueReference: used as internal identifier (i.e. usually hidden from the user)
- initial, e.g. exact or calculated
- causality, e.g. parameter
- derivative
- start, used for initialization

# Inspecting an FMU – Model Structure

```
Ubuntu 18.04 LTS
<ScalarVariable
  name="x1_start"
  valueReference="10"
  variability="fixed"
  causality="parameter"
  >
  <Real start="0.0"/>
</ScalarVariable>
</ModelVariables>
<ModelStructure>
  <Outputs>
    <Unknown index="6" dependencies="1" dependenciesKind="dependent" />
    <Unknown index="7" dependencies="2" dependenciesKind="dependent" />
  </Outputs>
  <Derivatives>
    <Unknown index="3" dependencies="1 2 5" dependenciesKind="dependent
dependent dependent" />
    <Unknown index="4" dependencies="1" dependenciesKind="dependent" />
  </Derivatives>
  <InitialUnknowns>
    <Unknown index="2" dependencies="11" dependenciesKind="dependent" />
    <Unknown index="3" dependencies="1 5 8 9 10 11" dependenciesKind="de
pendent dependent dependent dependent dependent" />
    <Unknown index="4" dependencies="1" dependenciesKind="dependent" />
    <Unknown index="6" dependencies="1" dependenciesKind="dependent" />
    <Unknown index="7" dependencies="11" dependenciesKind="dependent" />
  </InitialUnknowns>
</ModelStructure>
</fmiModelDescription>
(END)
```

Defines the dependencies of context dependent unknowns

All outputs (continuous and discrete) of the FMU.

All continuous state derivatives. Each derivative corresponds to exactly one state variable.

All continuous state derivatives. Each derivative corresponds to exactly one state variable.

# SSP - System Structure and Parameterization

The screenshot displays the SSP website interface. At the top, there is a navigation menu with links for Start, Downloads, Tools, Related, Development, Literature, History, FAQ, and Contact. The main content area features a class diagram on the left and a collection of file icons on the right. The class diagram includes boxes for System Structure Definition, Parameter Binding, Parameter Mapping, Element, Connector, Signal Dictionary, System, Signal Dictionary Reference, Component, and Connection, with arrows indicating relationships. The file icons represent various formats: \*.SSV (Parameter Mapping), \*.SSM (Parameter Mapping), \*.FMU (XML, C-Code, Libs), \*.SSP (SSD, FMU, SSP), and \*.SSB (Sig1, Sig2, Sig...). Below the diagram, there is a 'Latest News' section with a blue header and a list of updates from 2019.

**SSP**

System Structure and Parameterization (SSP) is a tool independent standard to define complete systems consisting of one or more FMUs (see Functional-Mockup-Interface) including its parameterization that can be transferred between simulation tools.

The first version of the standard has been published in March 2019. It is available in the Downloads section.

**Latest News**

- 2019-03-05, The SSP 1.0 standard has been publically released.
- 2019-03-04, The second Release Candidate for SSP 1.0 has been publically released.
- 2019-01-21

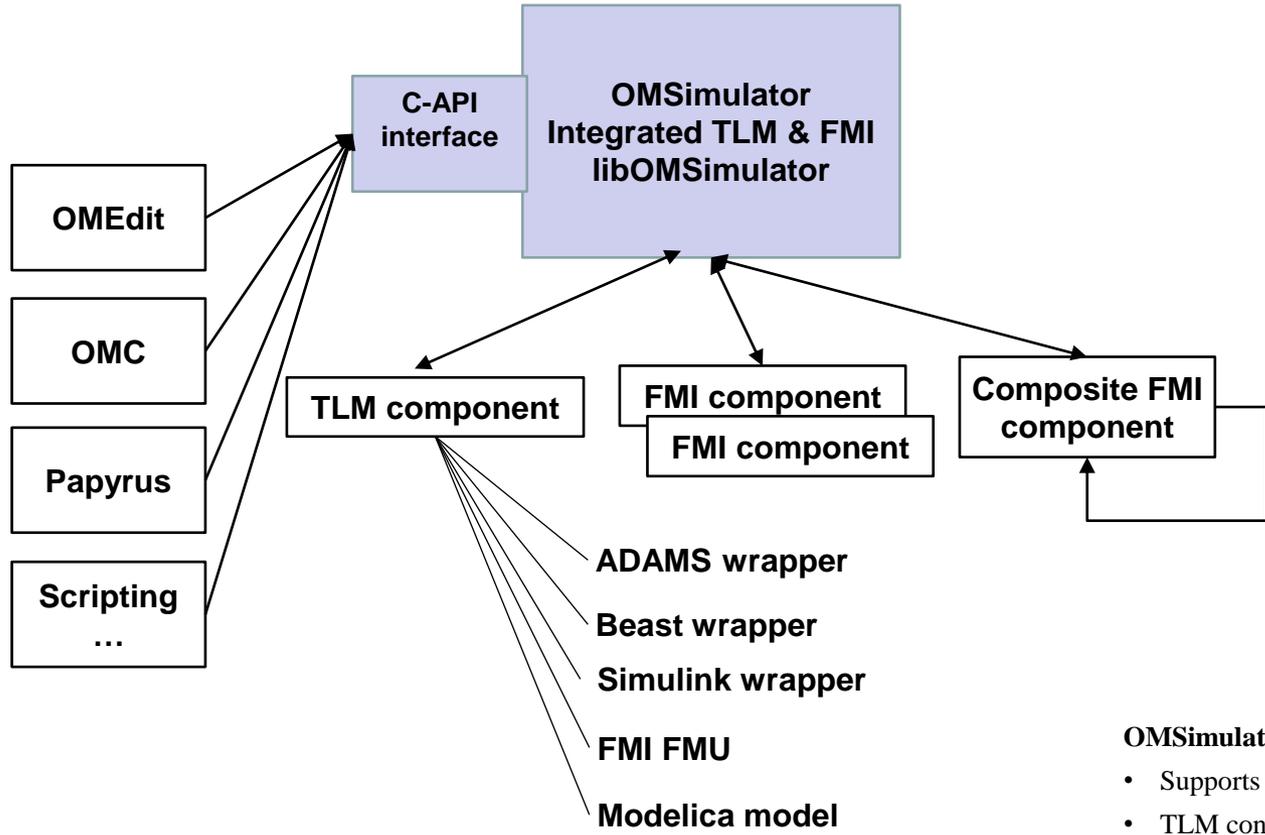
No news after 2019

---

# **FMI**

# **OpenModelica Implementation and Applications**

# OMSimulator – Integrated FMI and TLM-based Cosimulator/Simulator



### Main Framework Aspects

#### Unified co-simulation/simulation tool

- FMI 2.0 (model exchange and co-simulation)
- TLM (transition line modelling)
- Real-time and offline simulation

#### Standalone open source simulation tool with rich interfaces

- C/Java
- Scripting languages

#### Co-simulation framework as a solid base for engineering tools

- Integration into OpenModelica/Papyrus
- Open for integration into third-party tools and specialized applications (e.g. flight simulators, optimization)

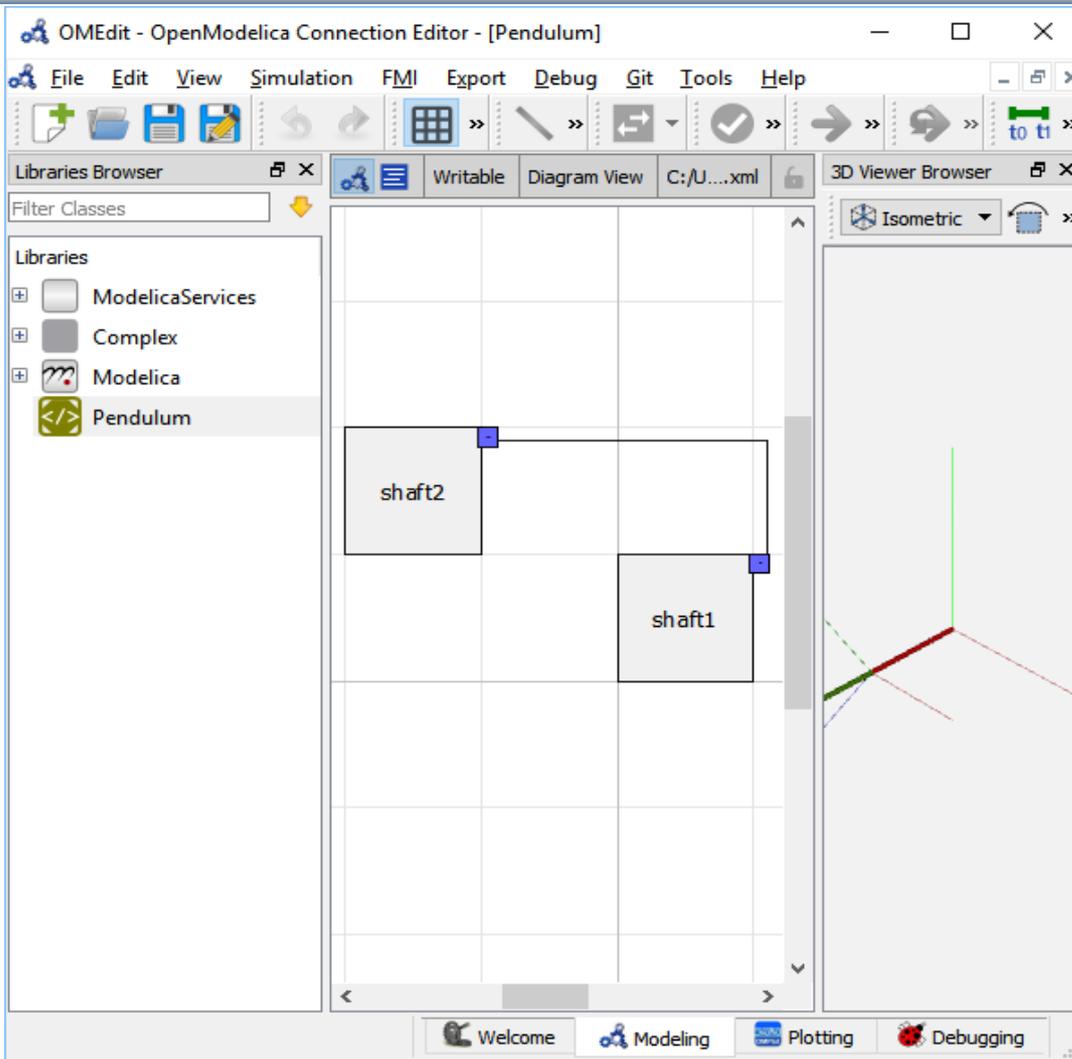
#### OMSimulator 2.0 in OpenModelica 1.13.0

- Supports both FMI and TLM
- TLM connections are optional
- Co-simulation to multiple tools
- Composite model editor operational
- External API interface and scripting not yet finalized

## Received ITEA Award of Excellence, Sept, 2019

- The **OPENCPS** project for which **OMSimulator** was the major result, received the ITEA award of excellence, September 2019
- ITEA Vice-chairman, Philippe Letellier referred to the major results delivered by the project, calling it "**a milestone on the path of open and standardised co-design and simulation of complex systems, that delivers major results**".

# OMSimulator Composite Model Editor with 3D Viewer



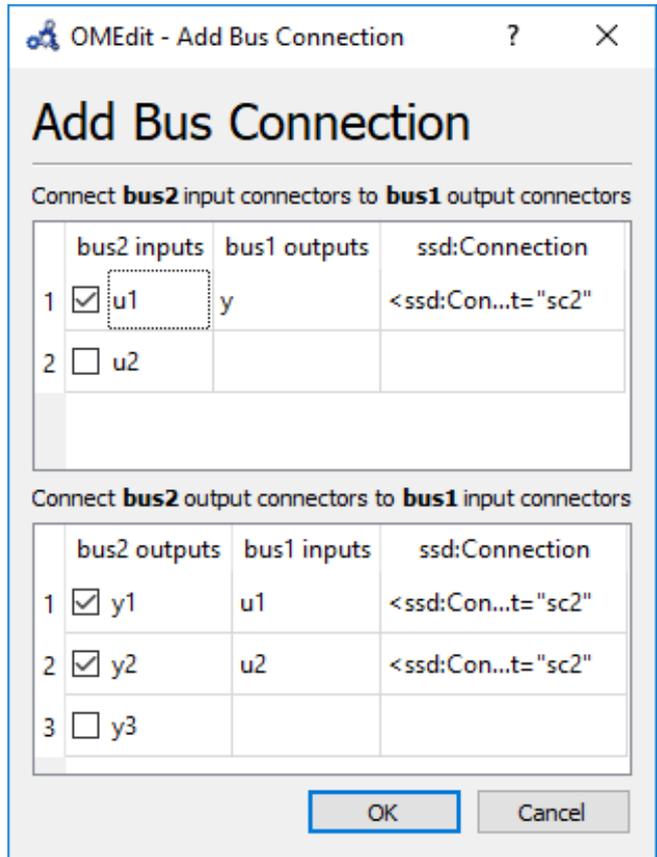
- **Composite model editor** with 3D visualization of connected mechanical model components which can be FMUs, Modelica models, etc., or co-simulated components
- **3D animation** possible
- Composite model saved as SSP XML-file
- **Support for SSP** – System Structure and Parameterization standard
- **Numerically stable** co-simulation with **TLM**

# OMSimulator – GUI and SSP support

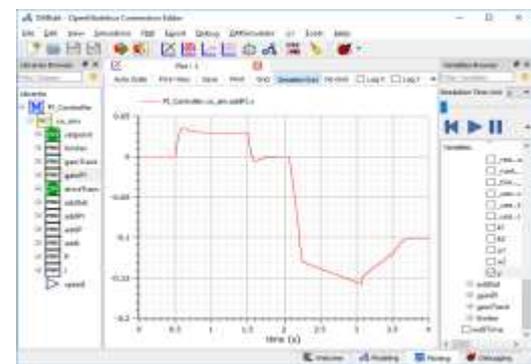
The screenshot displays the OMEdit - OpenModelica Connection Editor interface. The main workspace shows a diagram of two 'DualMassOscilla...' blocks, each labeled 'ME & CS 2.0', connected in a loop. The left sidebar shows the 'Libraries Browser' with the 'Model' library expanded to 'Root', containing 'DualMassOscillator\_System2' and 'DualMassOscillator\_System1'. Below the main workspace, a detailed diagram of a dual mass oscillator is shown, consisting of a fixed support, a spring damper (springDamper1), mass1 (m=m1 kg), a coupling spring damper (springDamper\_coupling), mass2 (m=m2 kg), and another spring damper (springDamper2) connected to a second fixed support. Two blue arrows labeled 'FMU1' and 'FMU2' point to the mass1 and mass2 blocks respectively. At the bottom left, two smaller diagrams show the internal structure of the FMU1 and FMU2 blocks, including components like 'springDamper1', 'mass', 'springDamper\_coupling', and 'springDamper2'. The bottom right of the screenshot shows a terminal window with the command: `server.py --model=C:/Users/Andre/AppData/Local/Temp/OpenModelica/OMEdit//Model.ssp --endpoint-pub=ocp://`. The status bar at the bottom indicates 'XI -211, Y: 103' and 'Welcome Modeling Plotting Debugging'.

# OMSimulator Simulation, SSP, and Tool Comparison

## Adding SSP bus connections



## FMI Simulation results in OMEdit



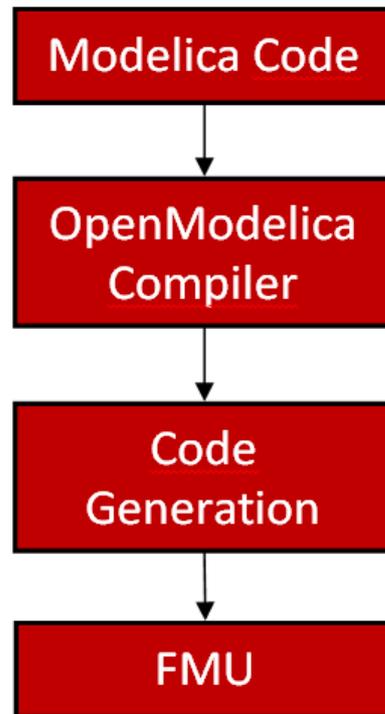
## FMI Simulation Tool Comparison

|                     | OMSimulator     | DACCOSIM    | Simulink        | PyFMI           |
|---------------------|-----------------|-------------|-----------------|-----------------|
| <b>Commercial</b>   | No              | No          | Yes             | No              |
| <b>Open-source</b>  | OSMC-PL, GPL    | AGPL2       | No              | LGPL            |
| <b>Lookup Table</b> | Yes             | Yes         | Yes             | No              |
| <b>Alg. Loops</b>   | Yes             | Yes         | No              | Yes             |
| <b>Scripting</b>    | Python, Lua     | proprietary | proprietary     | Python          |
| <b>GUI</b>          | Yes             | Yes         | Yes             | No              |
| <b>SSP</b>          | Yes             | No          | No              | No              |
| <b>platform</b>     | Linux/Win/macOS | Linux/Win   | Linux/Win/macOS | Linux/Win/macOS |

|                     | Dymola      | PySimulator | FMI Go!         | FMI Composer    |
|---------------------|-------------|-------------|-----------------|-----------------|
| <b>Commercial</b>   | Yes         | No          | No              | Yes             |
| <b>Open-source</b>  | No          | BSD         | MIT             | No              |
| <b>Lookup Table</b> | Yes         | Yes         | Yes             | Yes             |
| <b>Alg. Loops</b>   | Yes         | Yes         | Yes             | Yes             |
| <b>Scripting</b>    | proprietary | Python      | Go              | No              |
| <b>GUI</b>          | Yes         | Yes         | No              | Yes             |
| <b>SSP</b>          | No          | No          | Yes             | Yes             |
| <b>platform</b>     | Linux/Win   | Linux/Win   | Linux/Win/macOS | Linux/Win/macOS |

# OpenModelica Functional Mockup Interface (FMI)

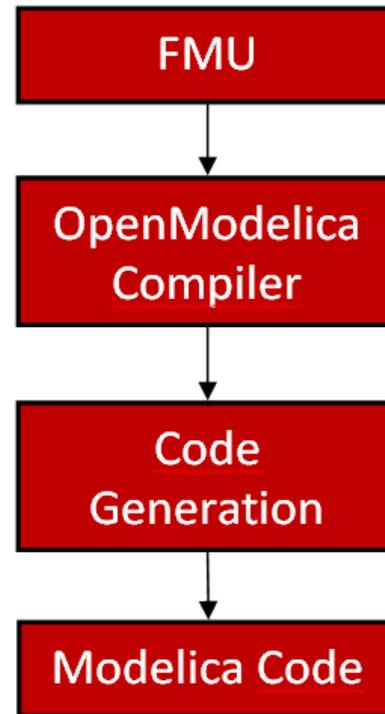
## FMI Export



Translator, Analyzer & Optimizer

Model Description, DLL & FMI interface functions

## FMI Import



FMU parsing, reading states & events

# FMI in OpenModelica Including OMSimulator

---

- Model Exchange implemented (FMI 2.0)
- Co-simulation implemented (FMI 2.0)
- Implementation of FMI 3.0 started (year 2023)

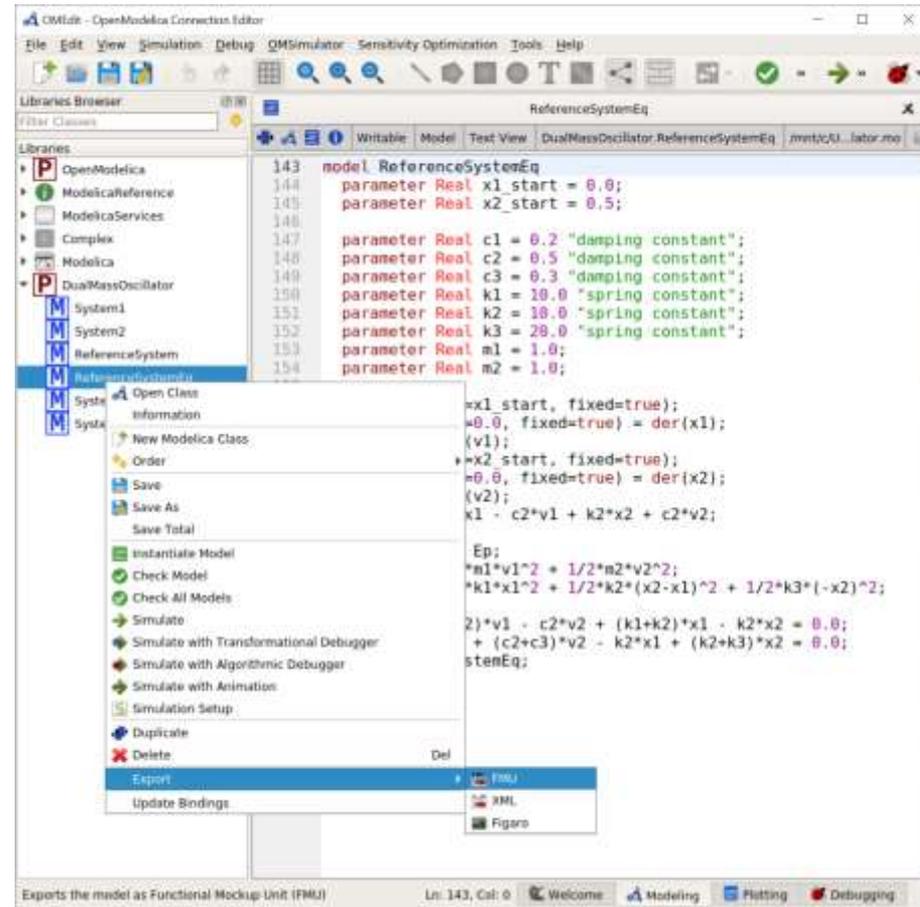
# FMI Export in OpenModelica

- OpenModelica scripting API

```
function translateModelFMU
  input TypeName className
  "the class that should be
  translated to FMU";
  input String version=
  "1.0" "Version of the FMU";
end translateModelFMU;
```

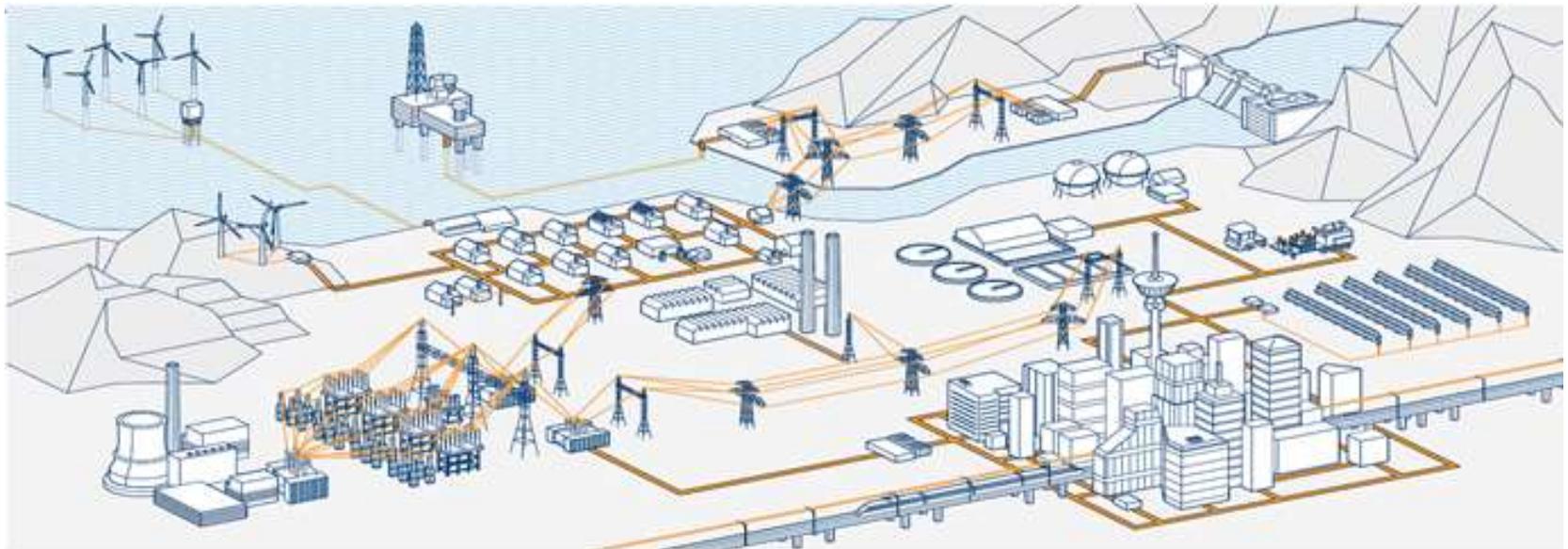
- Creates an FMU of the model
- Version parameter specifies the version of the FMU

## Exporting an FMU from OMEdit



# ABB OPTIMAX – OpenModelica Industrial Use Case

- ABB OPTIMAX® provides advanced model based control products for power generation and water utilities.



# ABB OPTIMAX – OpenModelica Industrial Use Case

- Model-based optimization of power plants using OpenModelica FMI 2.0.
- Plant models are formulated in Modelica and deployed through FMI 2.0
- Link : <http://new.abb.com/power-generation/power-plant-optimization>

---

# FMI Conclusions

# FMI Conclusions

- **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  
Clocks and events for co-simulation will be introduced with FMI 3.0
- **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 the size of result files is typically reduced by factor 4-5.

---

# **Some Industry Perspectives by Christian König, TWT Company INTO-CPS project**

# Industry Perspective – FMI in Practice

- Some automotive customers see FMI as the future
  - Integrate collaboration between departments and suppliers
- First TWT customer projects for demonstration of FMI
  - Coupling of longitudinal and lateral dynamics (CarMaker, Dymola, Matlab/Simulink)
  - CoSimulation of FMUs / Comparison with native Simulation

# Industry Perspective - Goals for the Future

- Exchange of models is a good start, but not enough
  - Goal: Integrate FMI/FMU in development process
    - requirements
    - traceability
    - model management / database,
    - documentation

→ one of the reasons why industry are engaged