Functional Mockup Interface (FMI)

fmi-standard.org

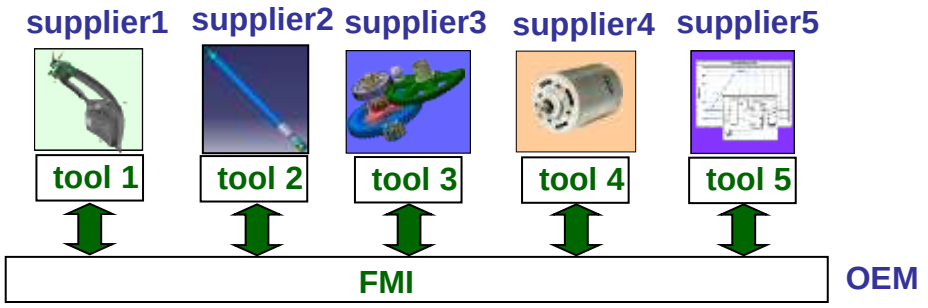
FMI – Motivation 1

- 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**

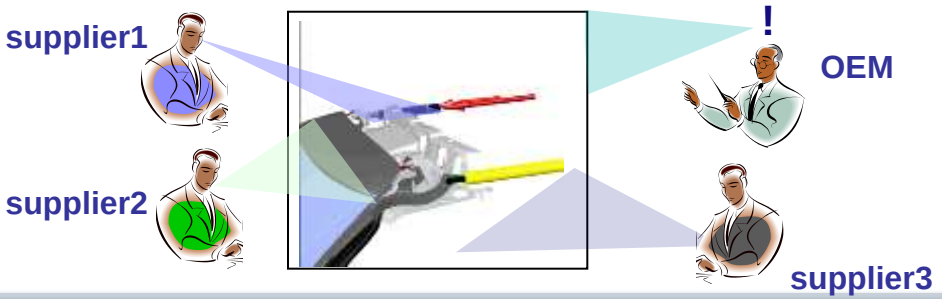
FMI – Motivation 2

- 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

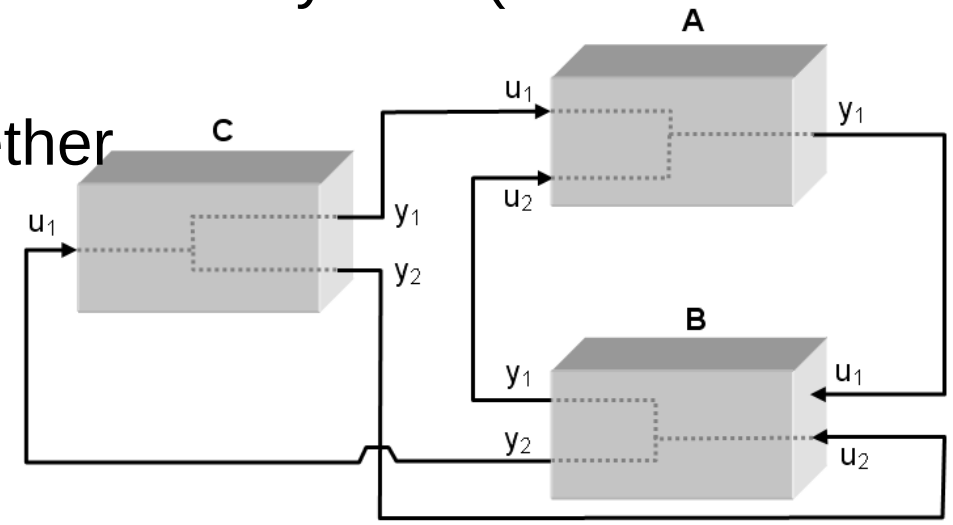
Exported Model in (Functional Mockup Unit) FMU Form

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 – 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

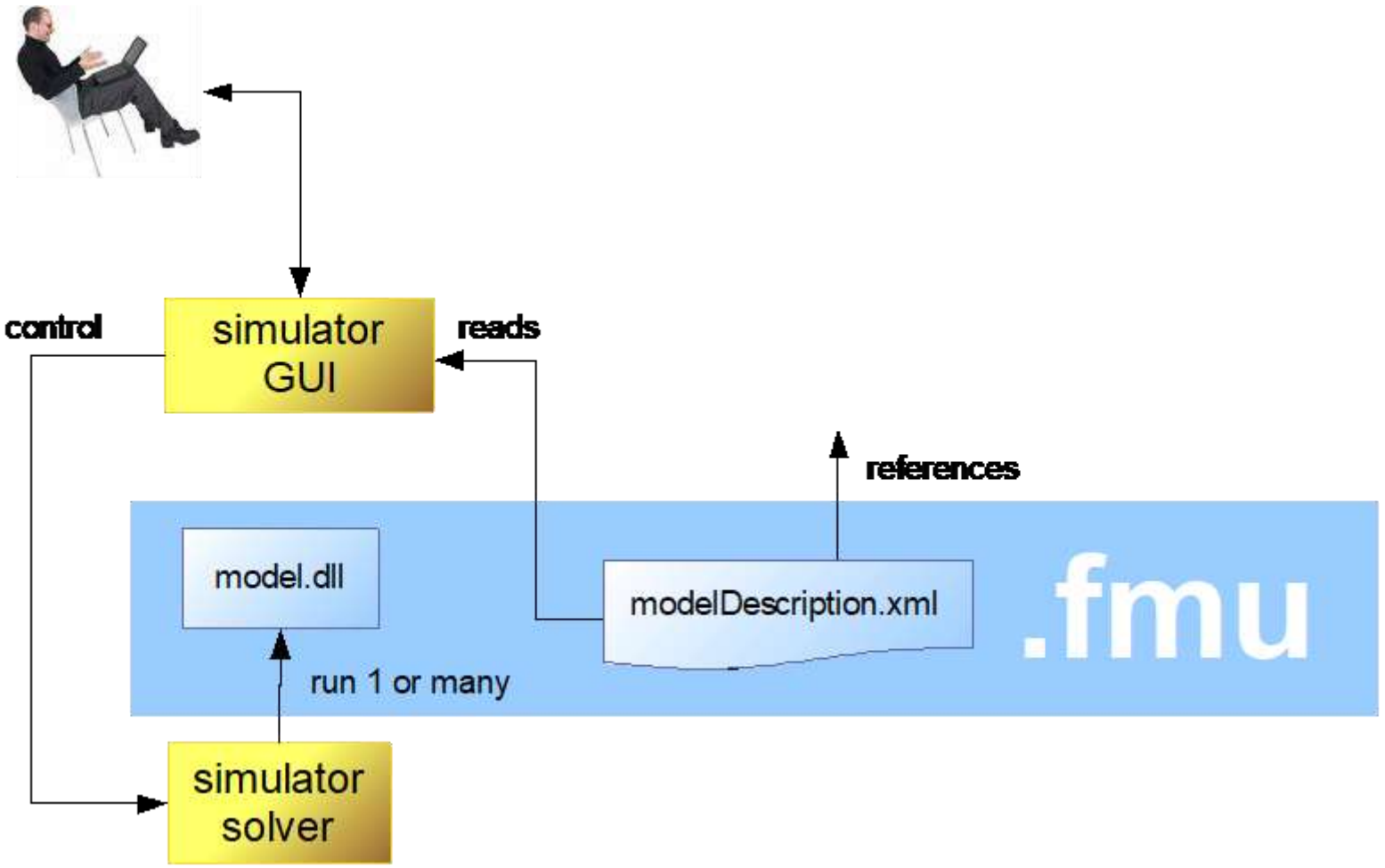


Model Distribution as a Zip-file (.fmu file)

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)

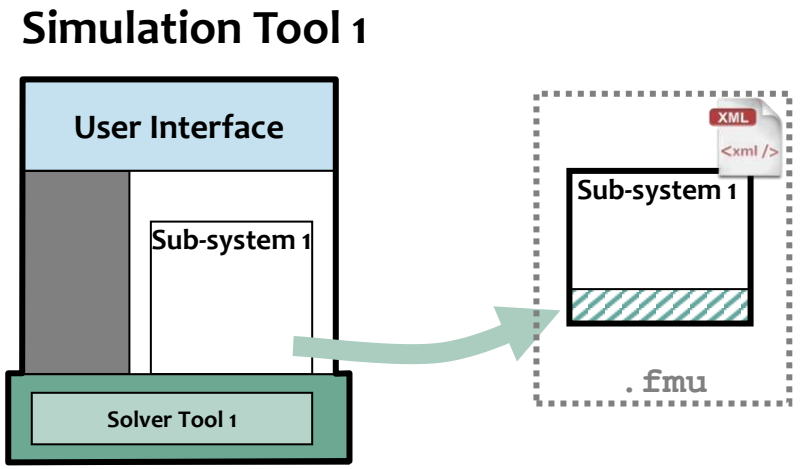
Simulator with GUI and Solver
Executing Imported Model = FMU (Functional Mockup Unit)



FMI for Model-Exchange

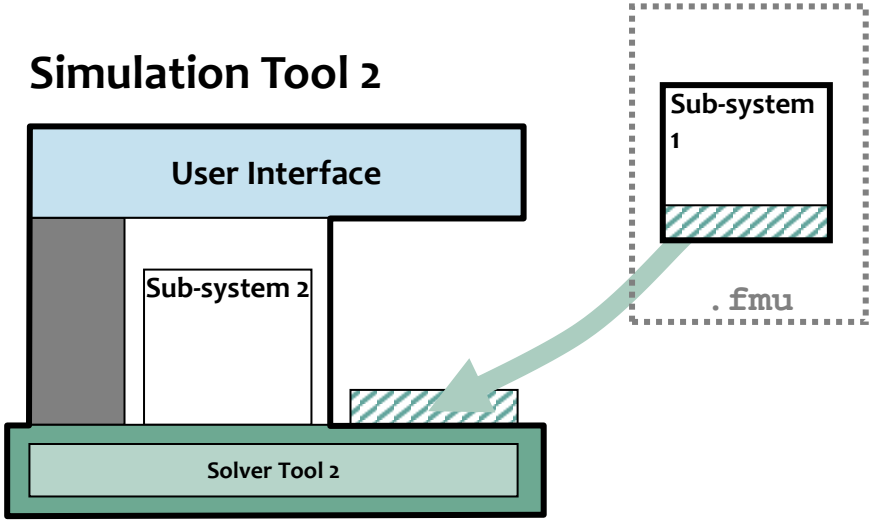
FMI for Model Exchange Export

- **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

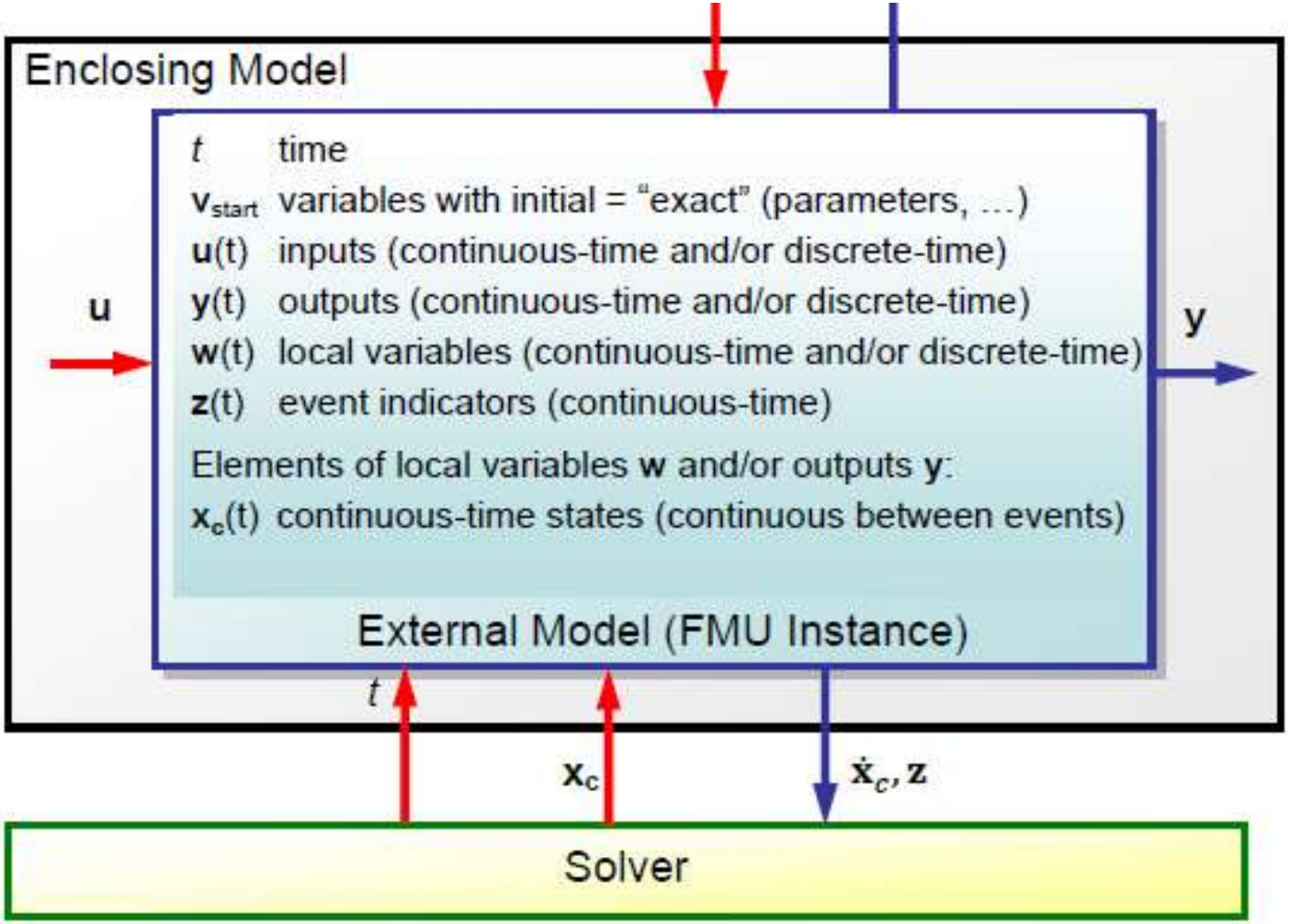


FMI for Model Exchange Import

- **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



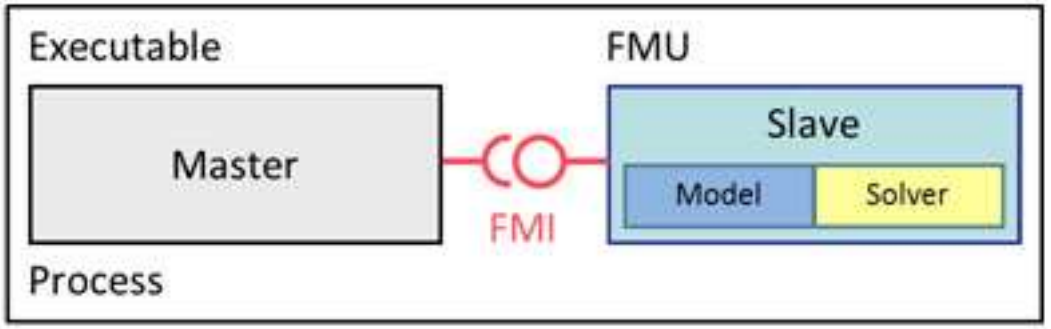
FMI for Model Exchange (contd.)



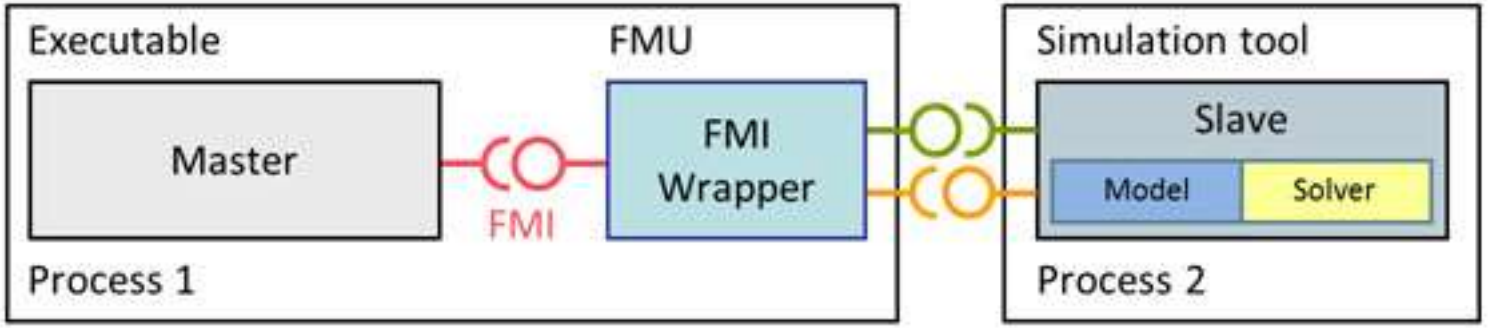
FMI for Co-Simulation

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,

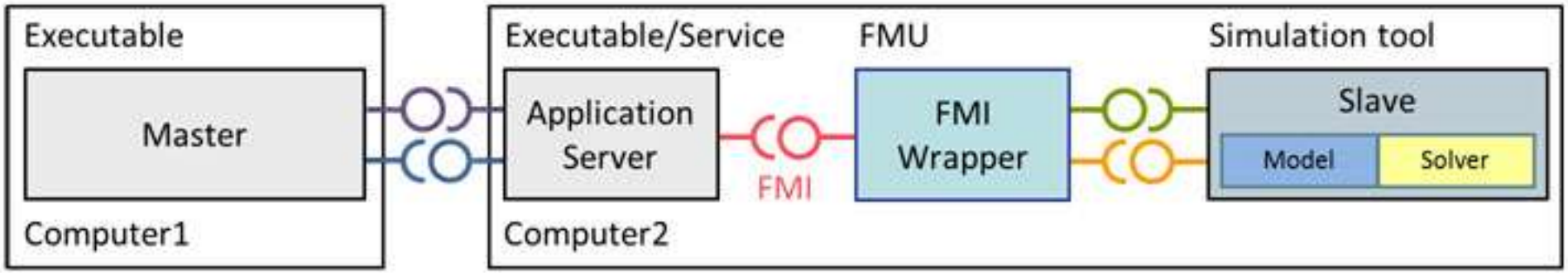


- And for coupling of simulation tools,



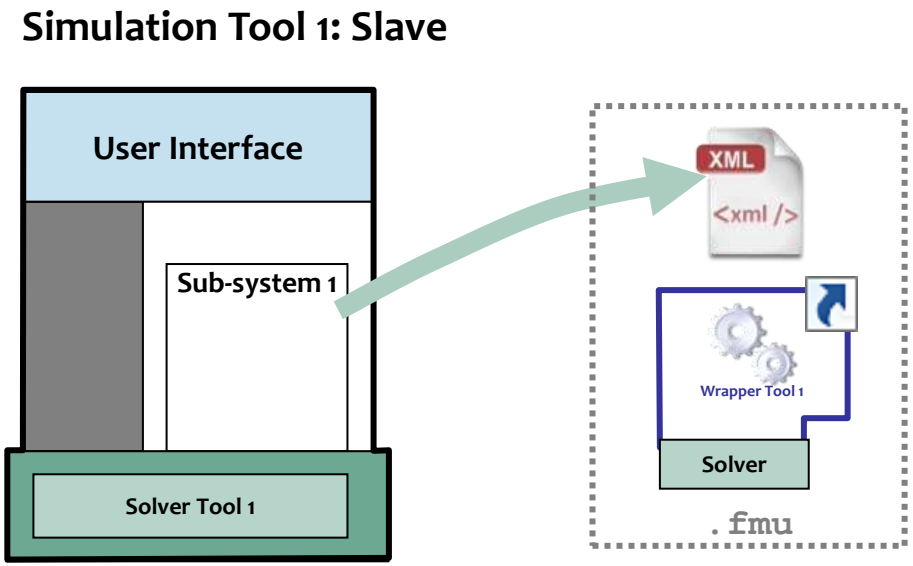
FMI for Co-Simulation

- 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.



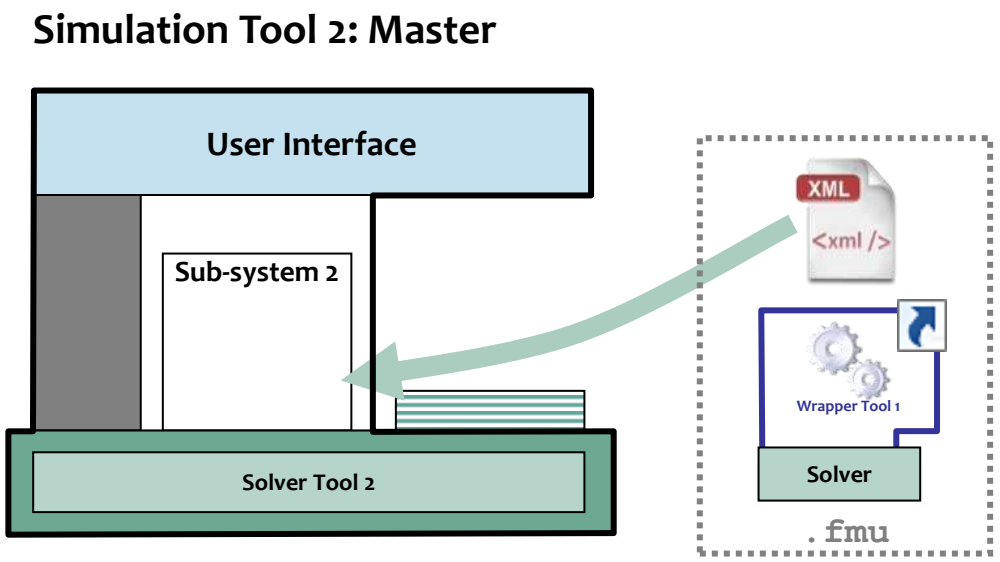
FMI for Co-Simulation Export FMU with Solver

- **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



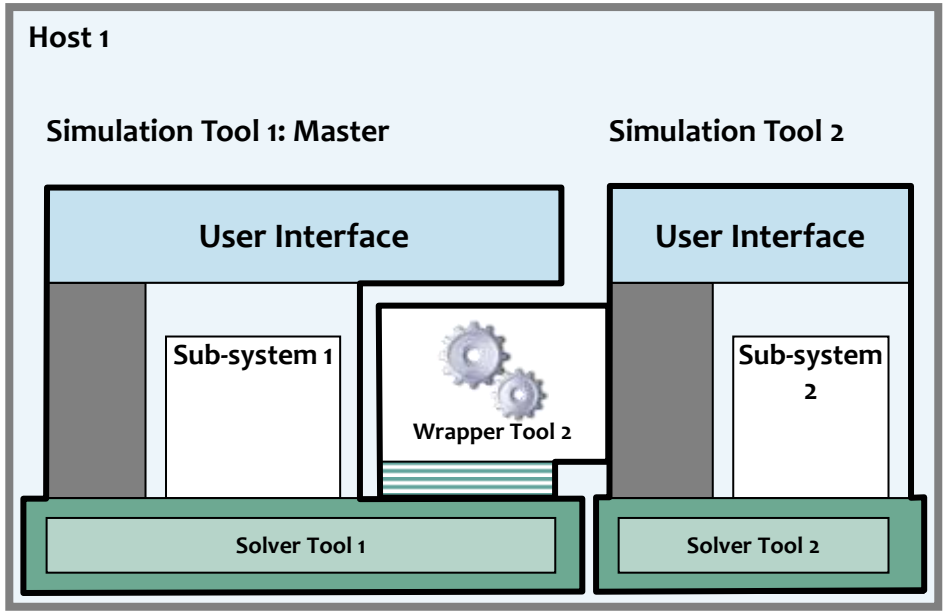
FMI for Co-Simulation Import Stand-alone

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



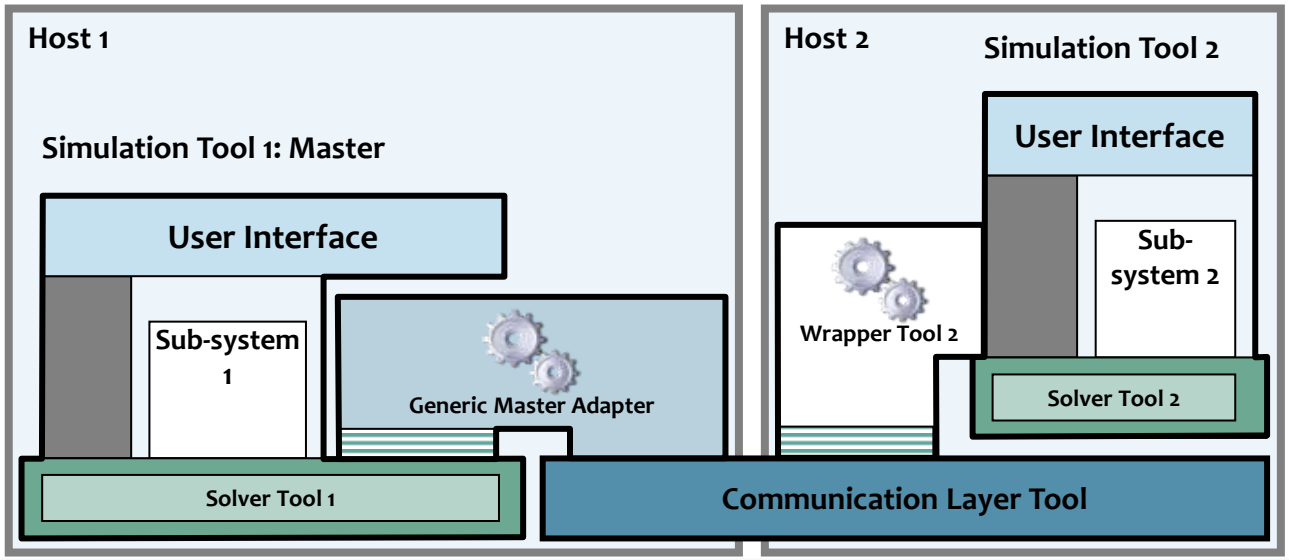
FMI for Co-Simulation Tool coupling

- **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

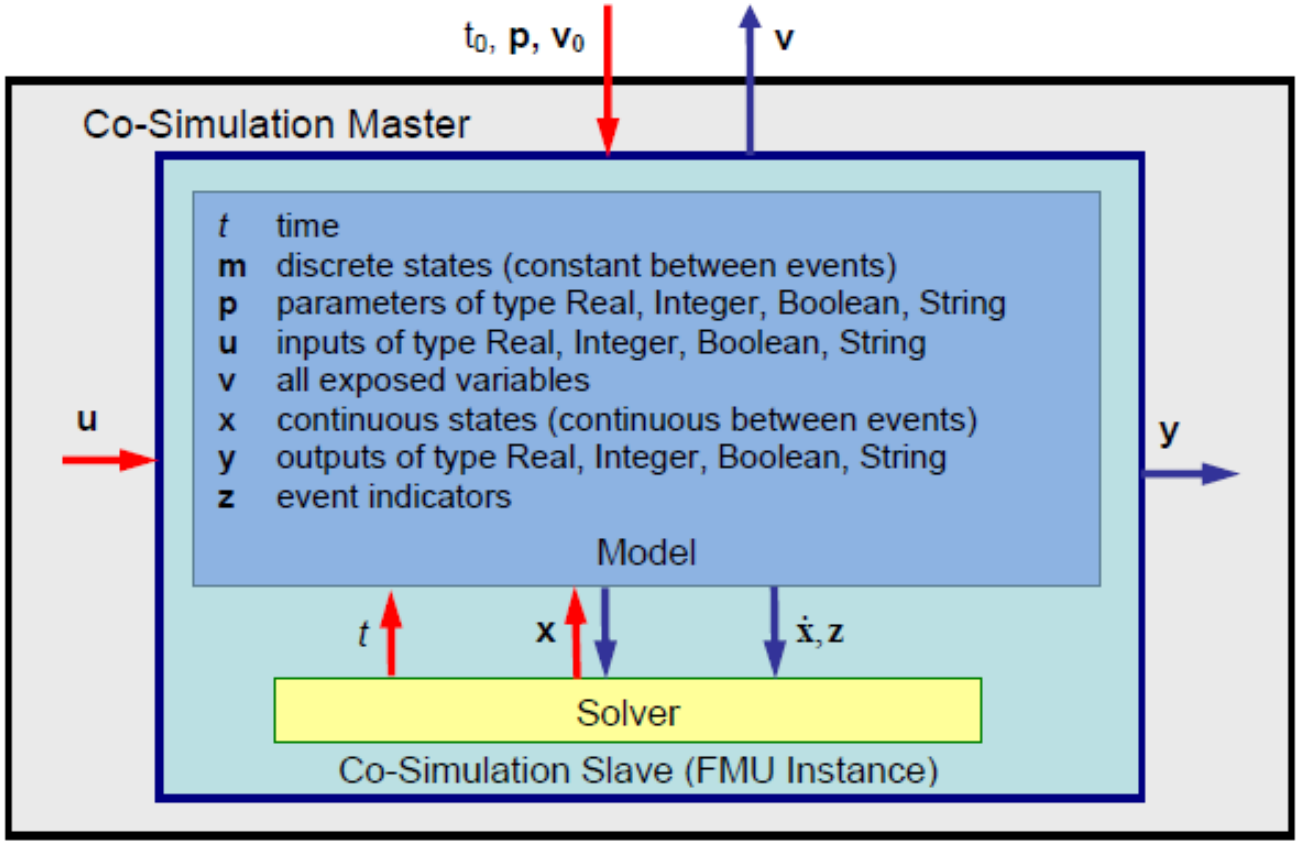


FMI for Co-Simulation distributed tool coupling

- **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

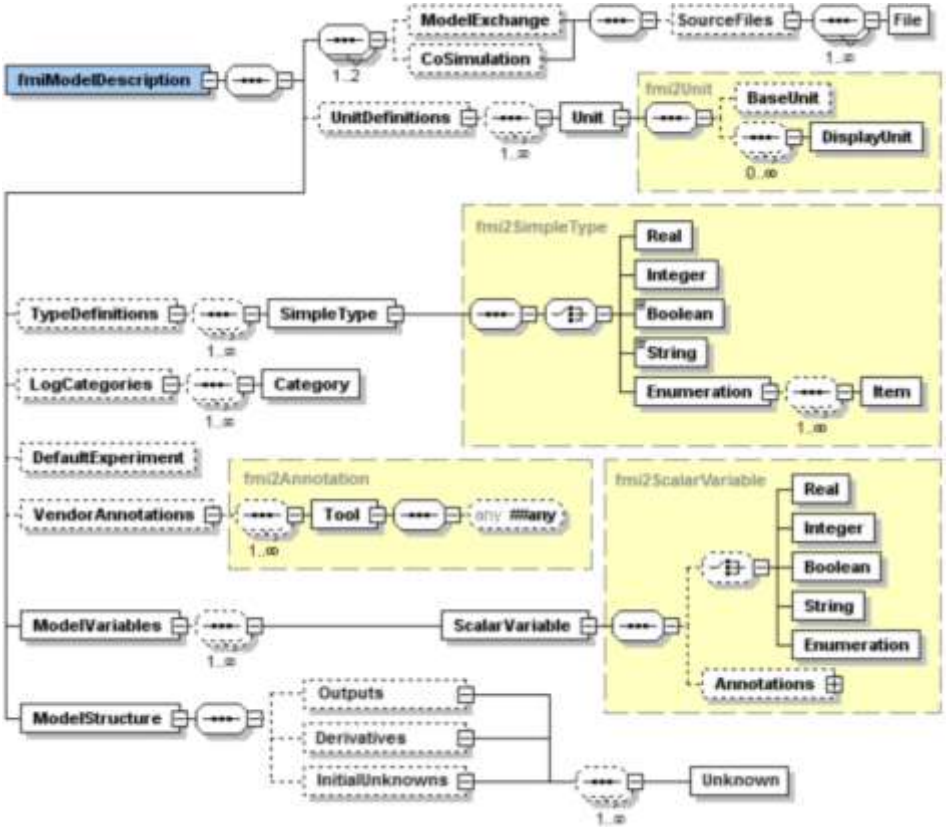
Structure of an FMU zip-file

```
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="inertia1.J"
      valueReference="1073741824"
      description="Moment of load inertia"
      causality="parameter"
      variability="fixed">
    <Real declaredType="Modelica.SIunits.Inertia" start="1"/>
    </ScalarVariable> <!—index="1" -->
    ...
  </ModelVariables>
</fmiModelDescription>
```

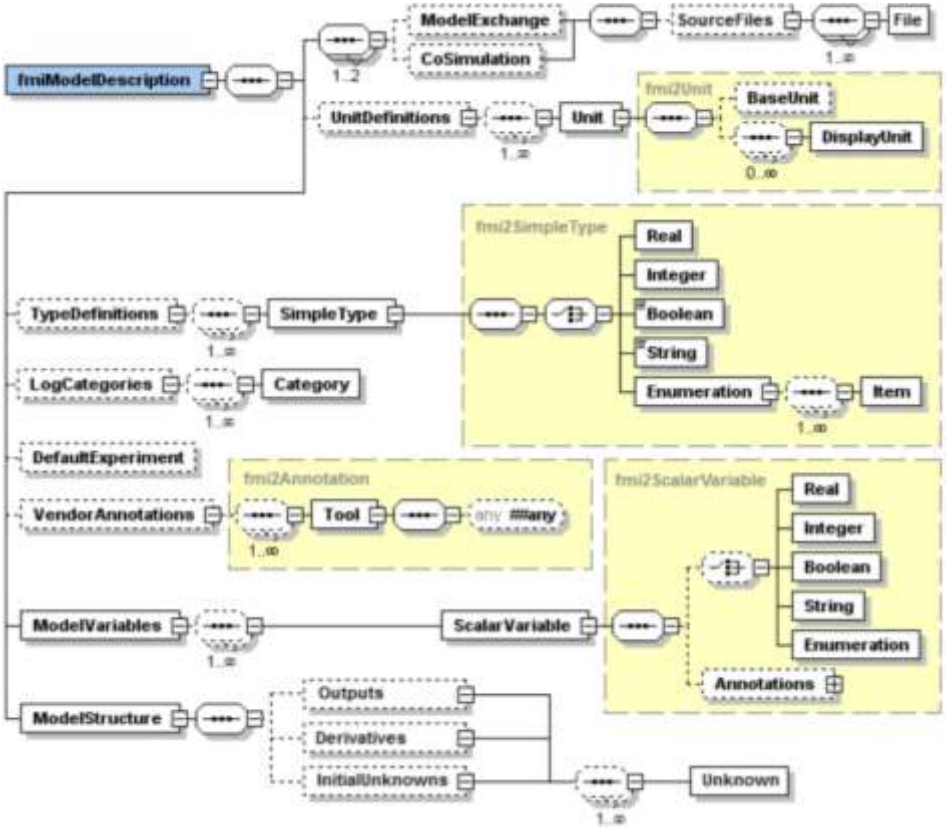
Inspecting an FMU



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+g693c0881d"
  generationDateAndTime="2020-02-25T16:43:10Z"
  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_06inz.c" />
      <File name="DualMassOscillator_ReferenceSystem_07dly.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" />
```

Inspecting an FMU

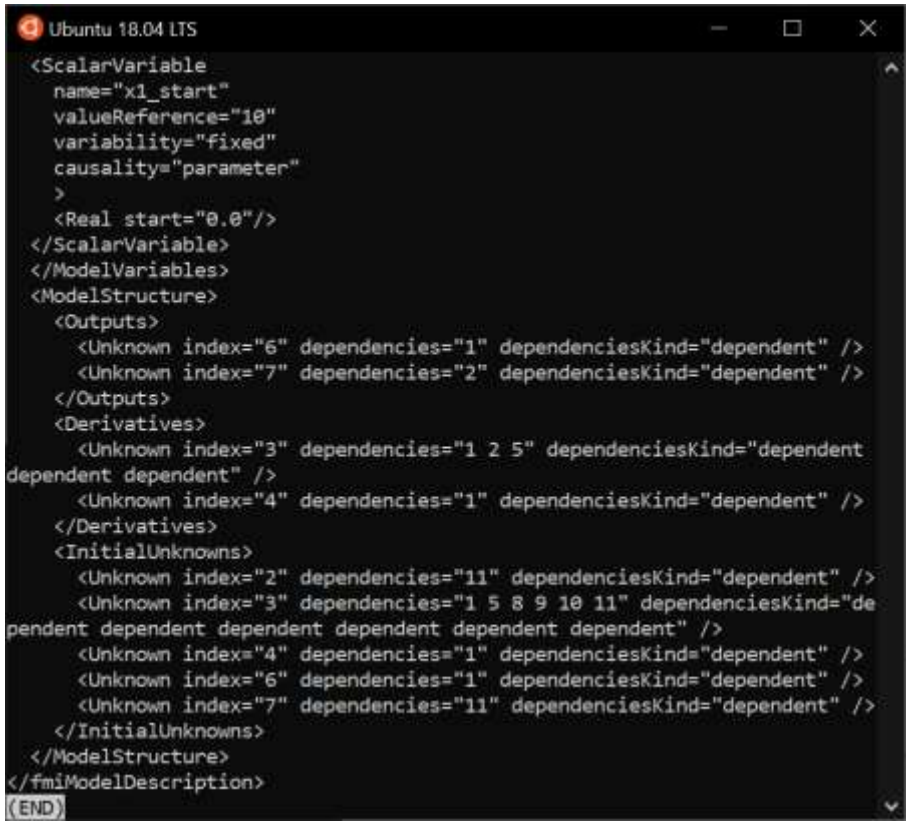


- ModelExchange
- CoSimulation
- UnitDefinitions
- TypeDefinition
- LogCategories
- DefaultExperiment
- VendorAnnotations
- **ModelVariables**
- **ModelStructure**

Inspecting an FMU – Model Variables

- 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



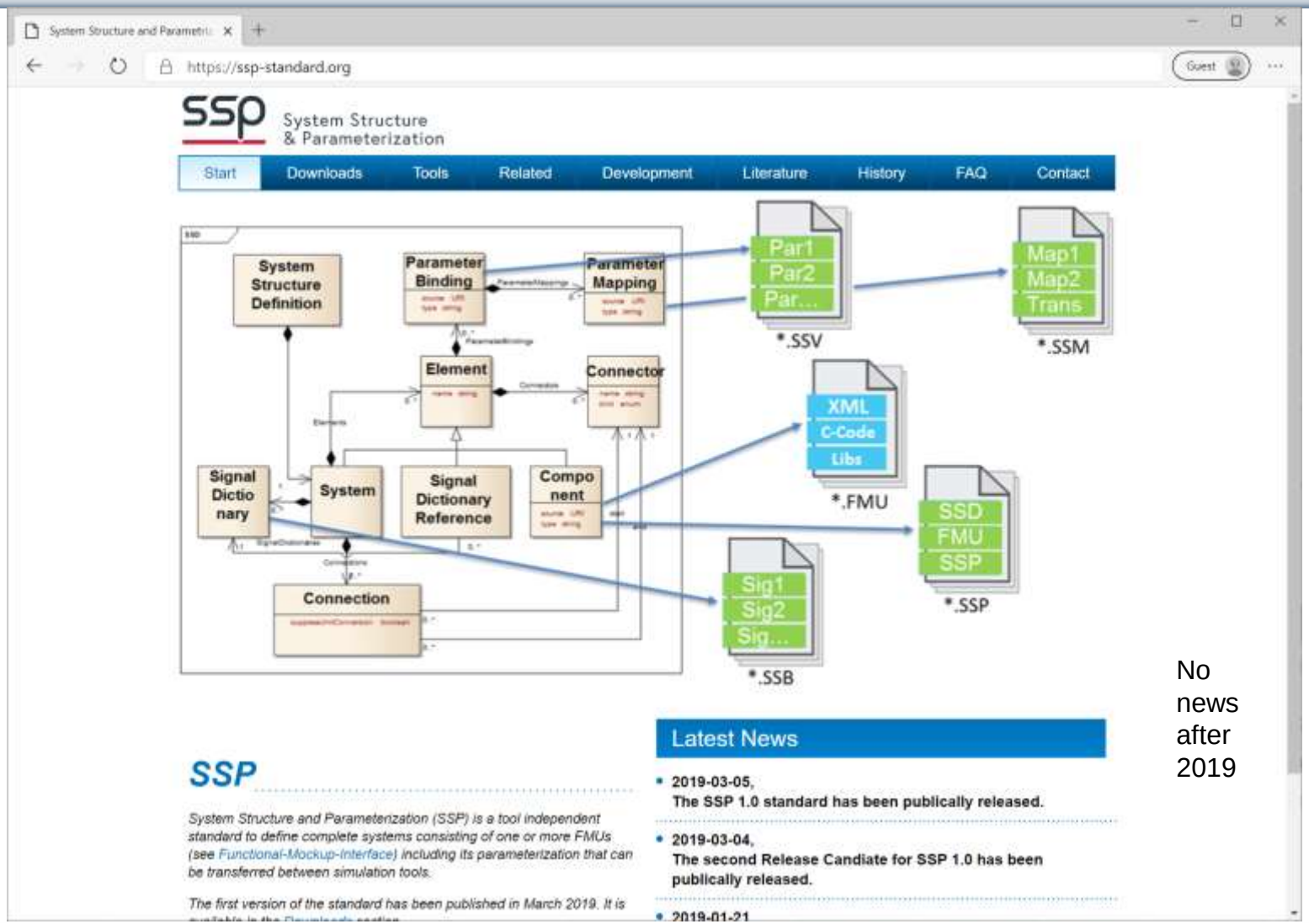
Defines the dependencies of context dependent unknowns

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.

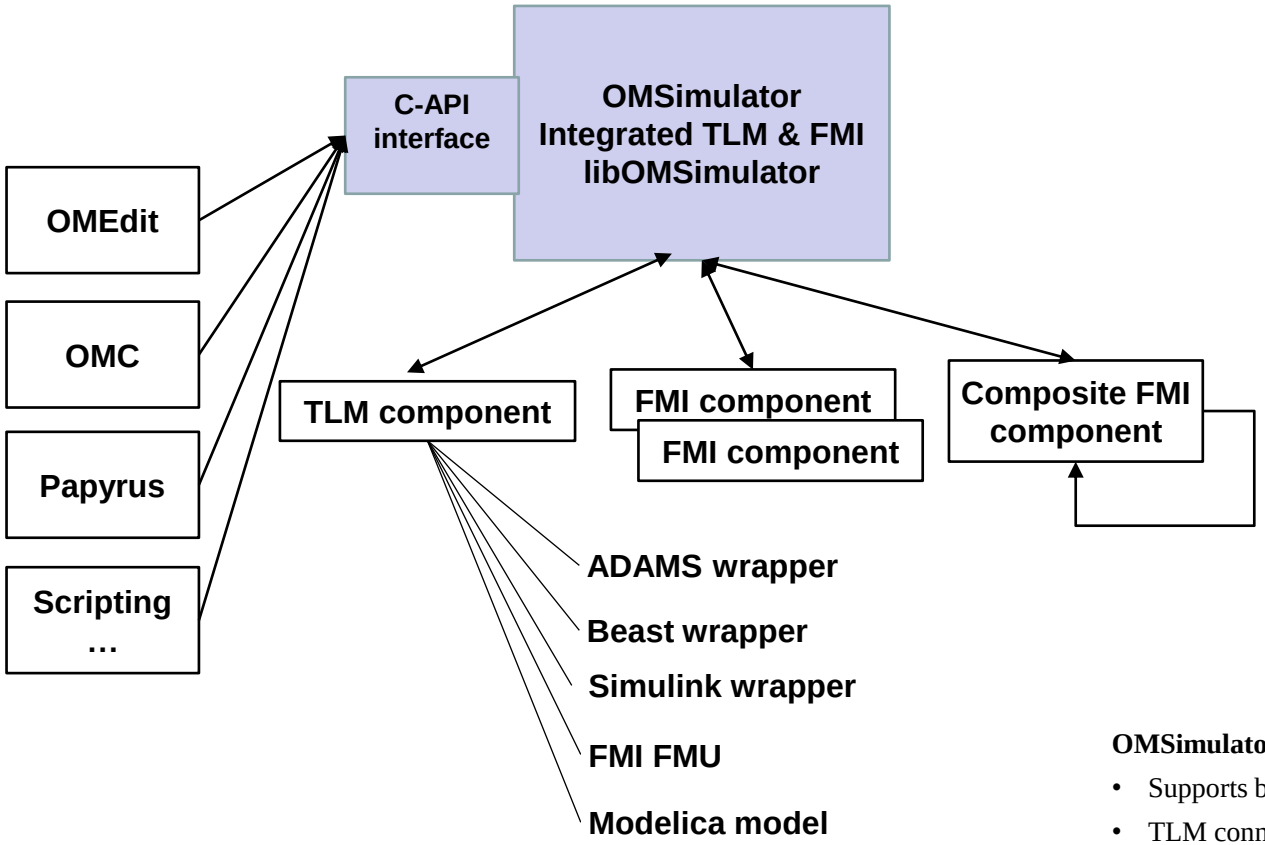
SSP - System Structure and Parameterization



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

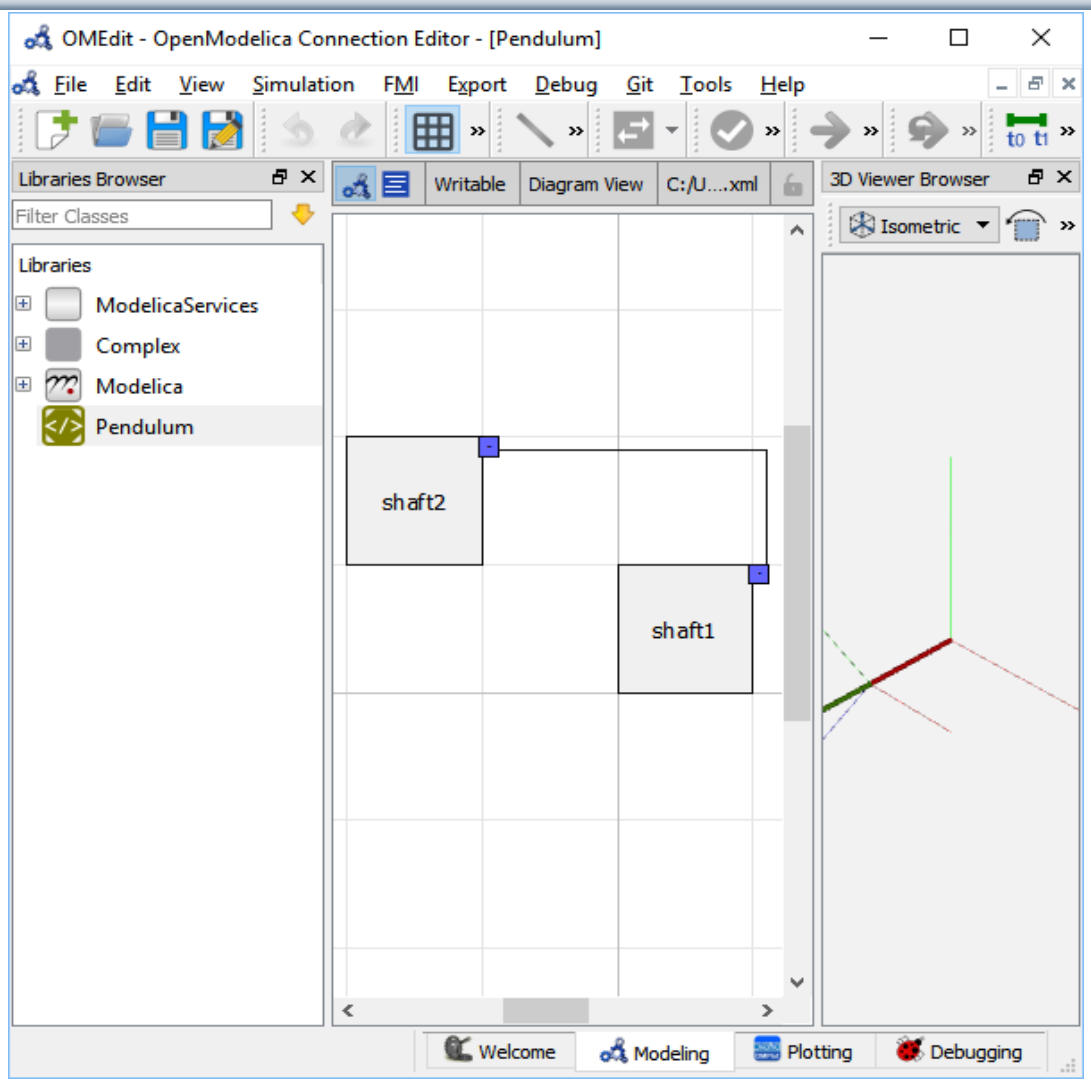
- Supports both FMI and TLM
- TLM connections are optional
- Co-simulation to multiple tools
- Composite model editor operational
- External API interface and scripting not yet finalized

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

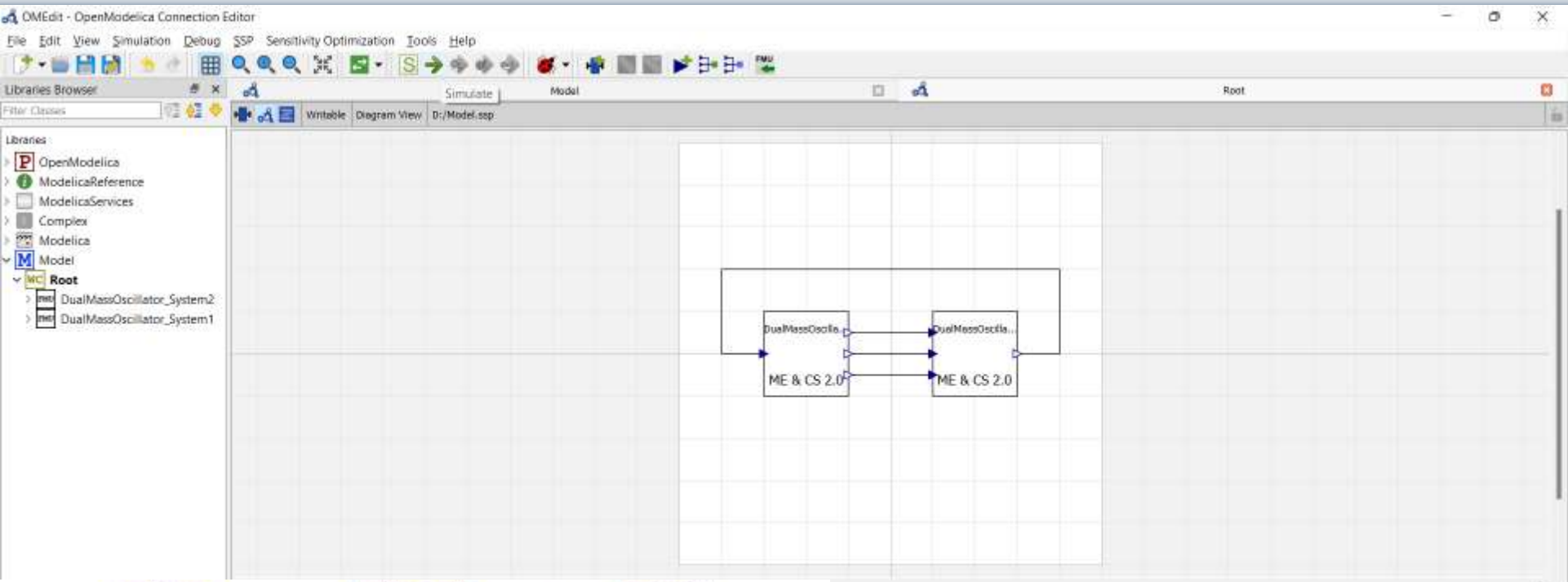
- ITEA Vice-chairman, Philippe Letellier referred to the major results delivered by the project, calling it "**a milestone on the path of open and standardised co-design and simulation of complex systems, that delivers major results**".

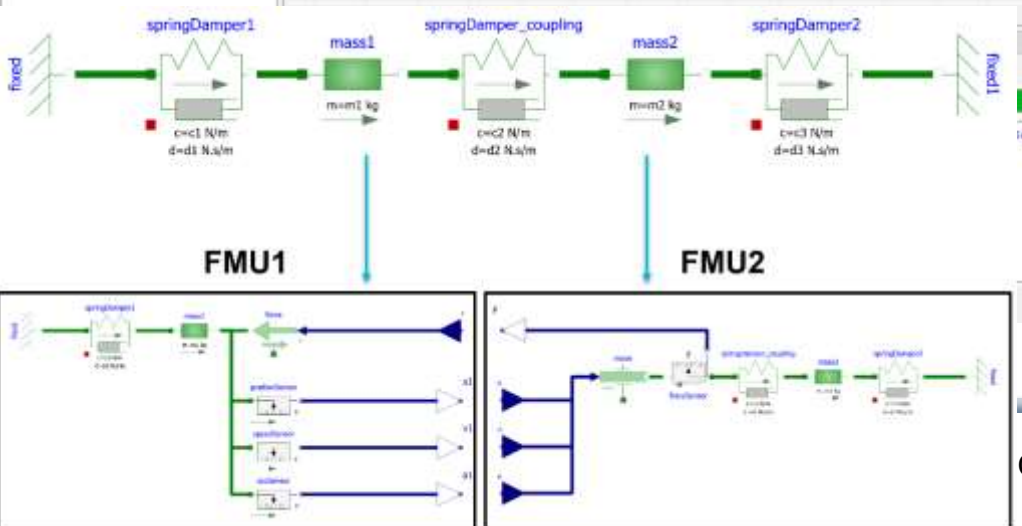
OMSimulator Composite Model Editor with 3D Viewer



- **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

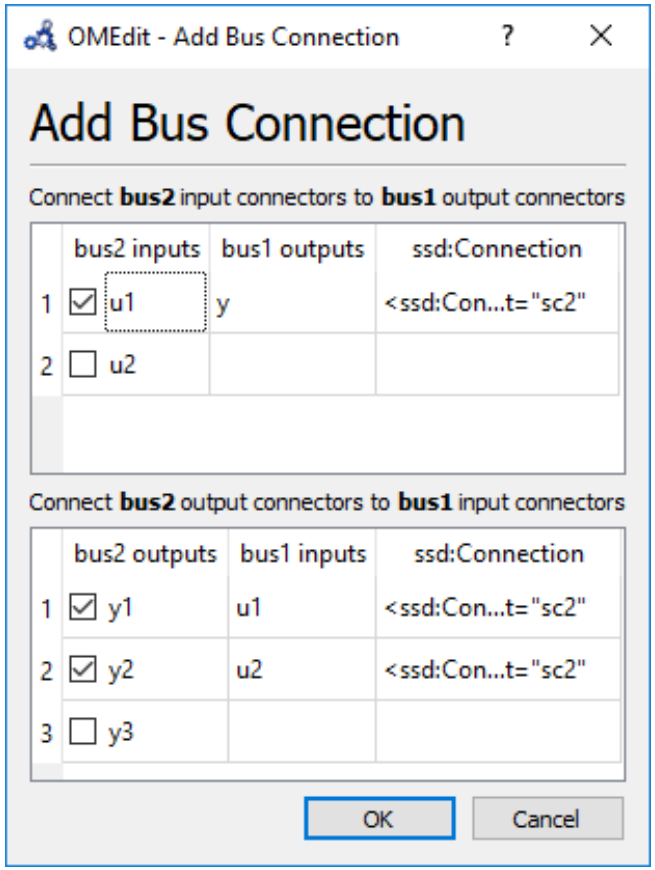




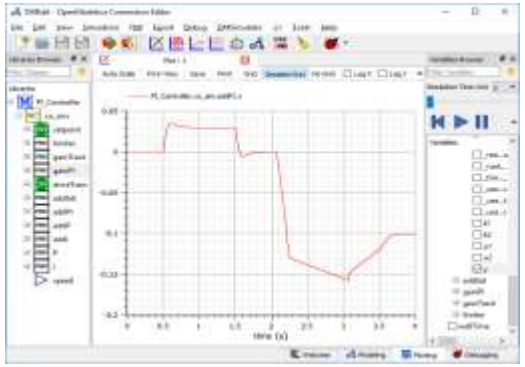
MODELICA

OMSimulator Simulation, SSP, and Tool Comparison

Adding SSP bus connections



FMI Simulation results in OMEdit

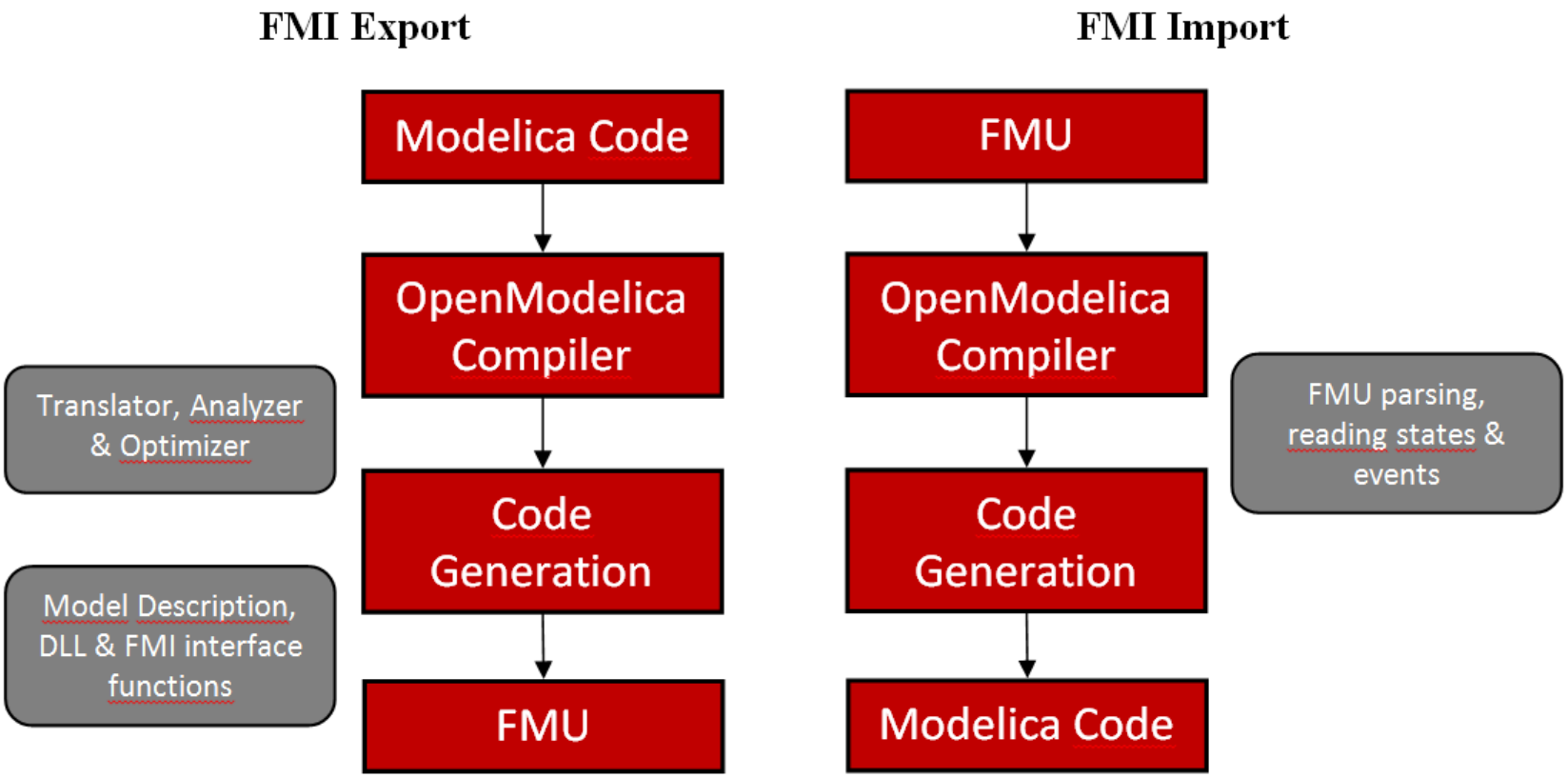


FMI Simulation Tool Comparison

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

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

OpenModelica Functional Mockup Interface (FMI)



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

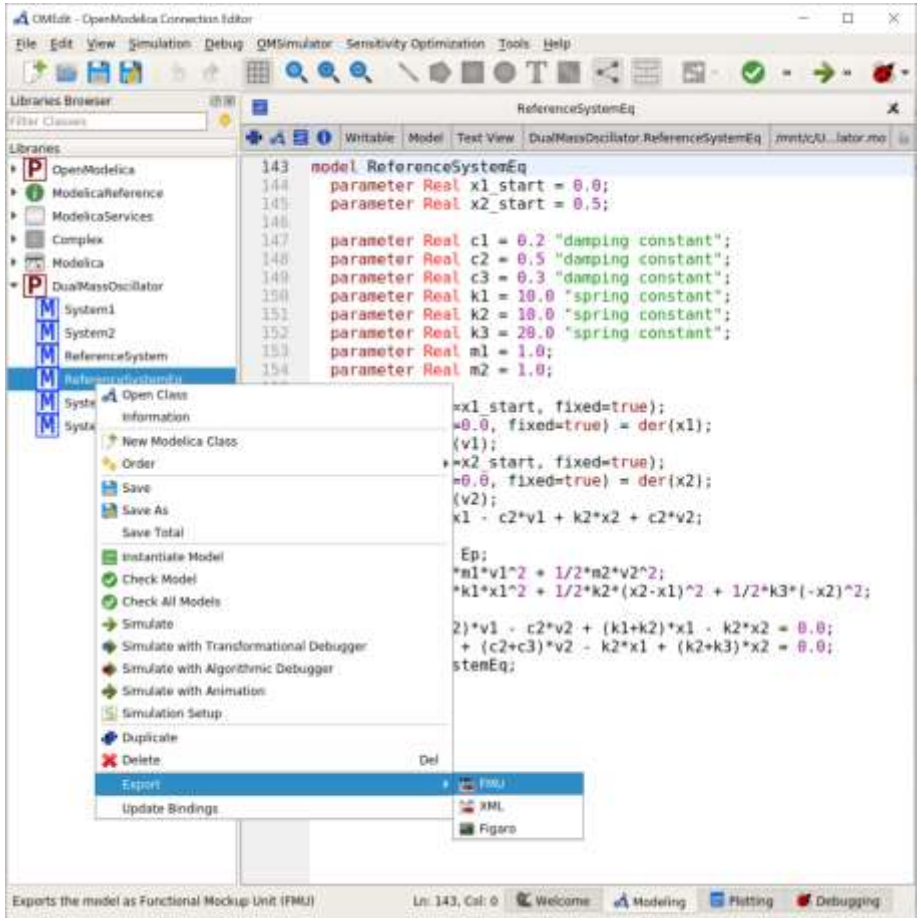


ABB OPTIMAX – OpenModelica Industrial Use Case

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

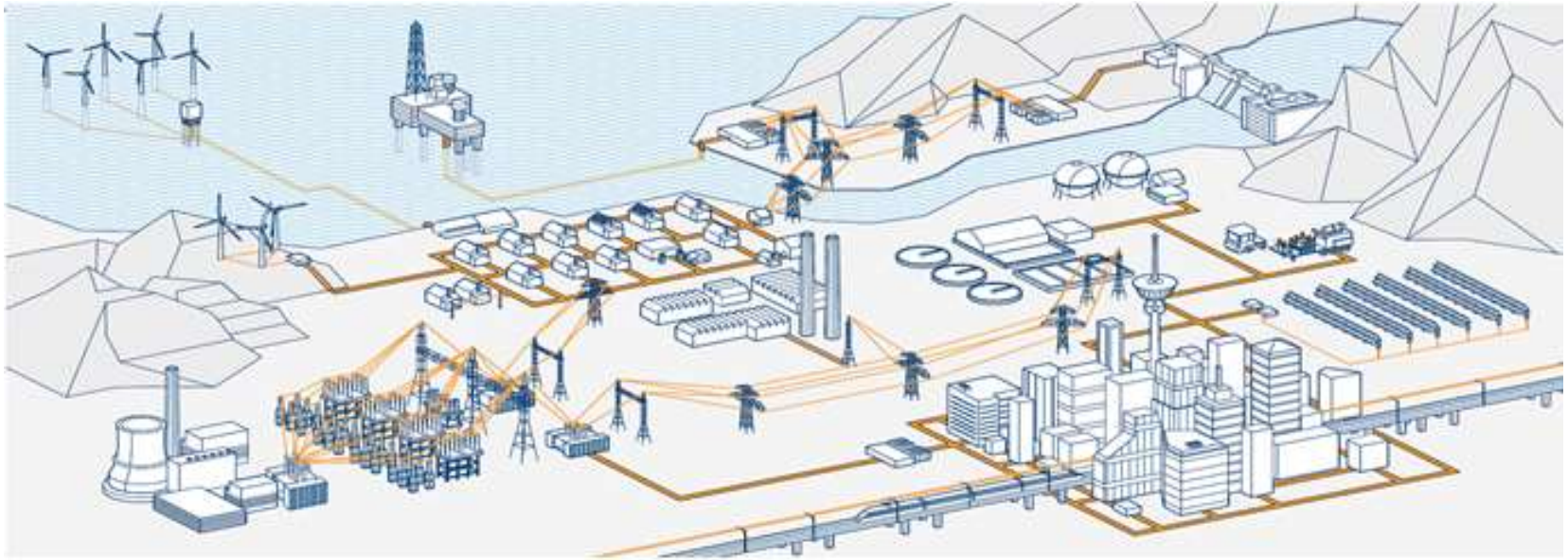


ABB OPTIMAX – OpenModelica Industrial Use Case

- 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