The Functional Mock-up Interface 3.0
-
New Features Enabling New Applications

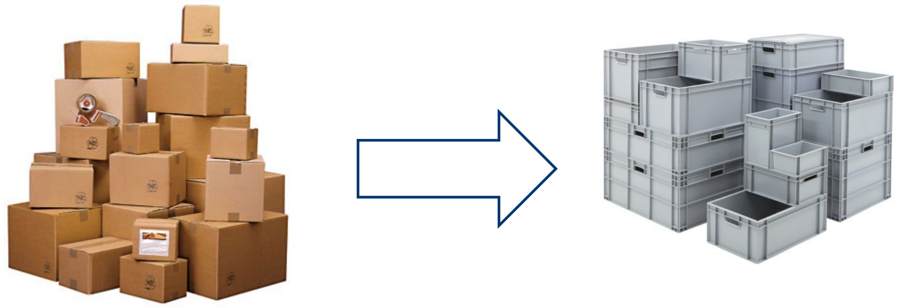
Andreas Junghanns, Torsten Blochwitz, Christian Bertsch, Torsten Sommer, Karl Wernersson, Andreas Pillekeit, Irina Zacharias, Matthias Blesken, Pierre R. Mai, Klaus Schuch, Christian Schulze, Cláudio Gomes, Masoud Najafi

Contents

FMI: Simpler „Plumbing“ for Simulation

- FMI for Model Exchange:
  How to connect systems of equations (ODEs)
- FMI for Co-Simulation:
  How to connect “any” model or tool

- Decouple Know-How between producers and users of FMUs
- Massive Re-use of modelling investment
- Many new use-cases are now viable
- 150+ tools now support FMI:
  See: fmi-standard.org/tools



https://fmi-standard.org/

Motivation for FMI 3.0

150+ tools support FMI now: many users now, many new use-case requests:

- Virtual Electronic Control Units (vECUs):
  - FMI 2.0 works well for physics simulations: better support for vECUs is needed

vECU

- Advanced Co-Simulation
  - Co-Simulation is the more popular interface type: improved co-simulation methods are needed to improve performance and accuracy

Co-Sim

- Multi-FMU simulations are getting more common
  - Events must be synchronized across FMUs

Events

- New ML and AI applications
  - More derivatives computations is required

AI

FMI 3.0: Main Improvements

- Event Mode for Co-Simulation
- Intermediate Variable Update
- Clocks
- New Types
- Array Variables
- Terminals and Icons
- FMI for Scheduled Execution
- Preparation for Layered Standards

Performance

Accuracy

New Application

FMI 3.0: New Interface Type – Scheduled Execution

vECU

Events

FMI 2.0	Model Exchange	Co-Simulation	
FMI 3.0	Model Exchange	Co-Simulation	Scheduled Execution

Scheduled Execution allows coupling several FMUs with one, external scheduler (OS)

FMI 3.0: New Data Types

vECU

FMI 1.0, FMI 2.0	FMI 3.0	Remarks
`fmiReal`	`fmi3Float32`	Discrete and continuous variables
	`fmi3Float64`	States, derivatives, event-indicators
`fmiInteger`	`fmi3Int8, fmi3UInt8`	Discrete variables
	`fmi3Int16, fmi3UInt16`	
	`fmi3Int32, fmi3UInt32`	
	`fmi3Int64, fmi3UInt64`	
`fmiBoolean`	`fmi3Boolean`	`char`
`fmiString`	`fmi3String`	`const char*` ('\0' terminated, UTF-8 encoded)
	`fmi3Binary`	`const char*` (e.g. sensor outputs, bitmaps, …)
	`fmi3Clock`	Transport information about events

Array Variables (Vectors, Matrices, …)

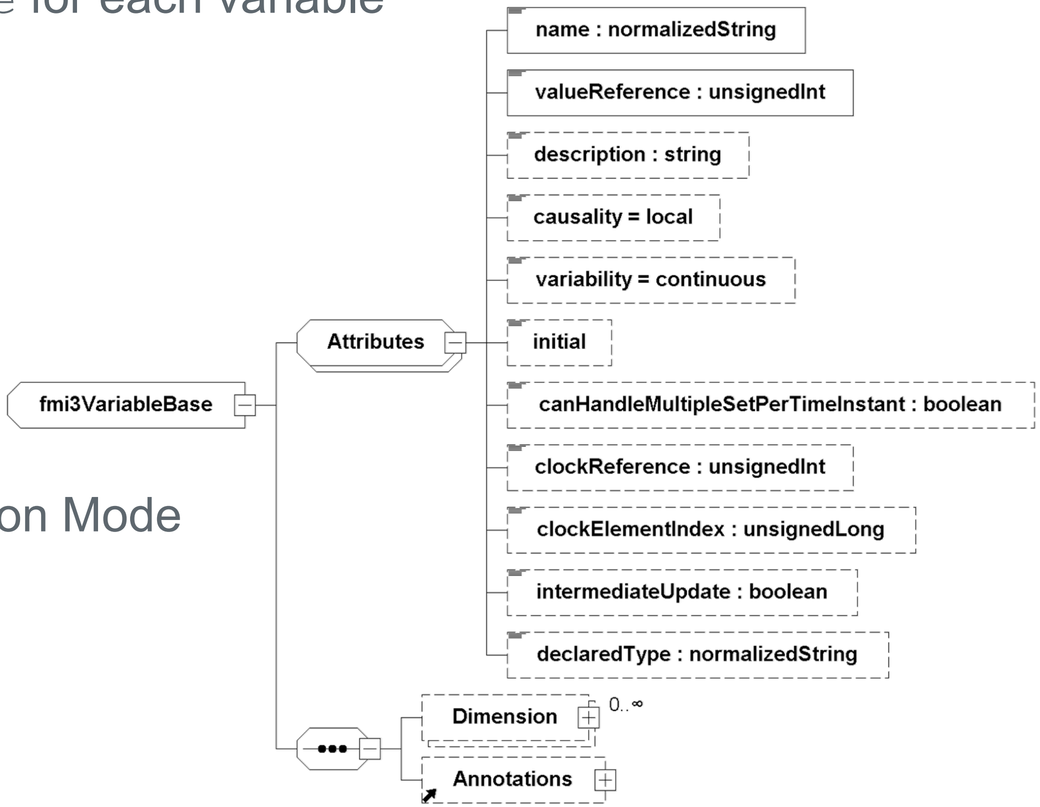
vECU

FMI 1.0 and FMI 2.0:

- Naming convention for array elements (`building.room.temp[0], building.room.temp[1], …`)
- Array elements are treated as scalars, one `valueReference` for each variable

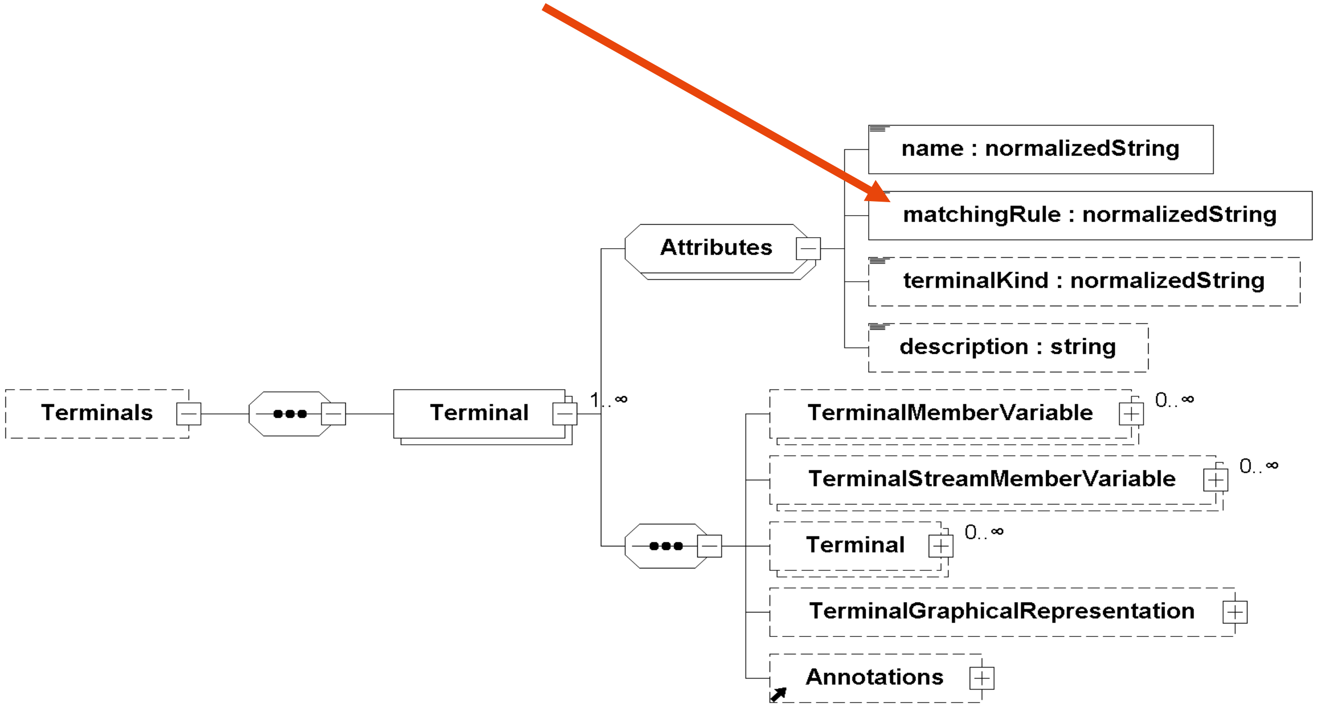
FMI 3.0:

- Variables can have multiple `<Dimension>` elements
- Each dimension can be:
  - Constant: specified by `<start>` attribute
  - Changeable: depends on a `structural` parameter which can change its value only in Configuration or Reconfiguration Mode
- `fmi3SetXXX`, `fmi3GetXXX` work on whole arrays



Terminals

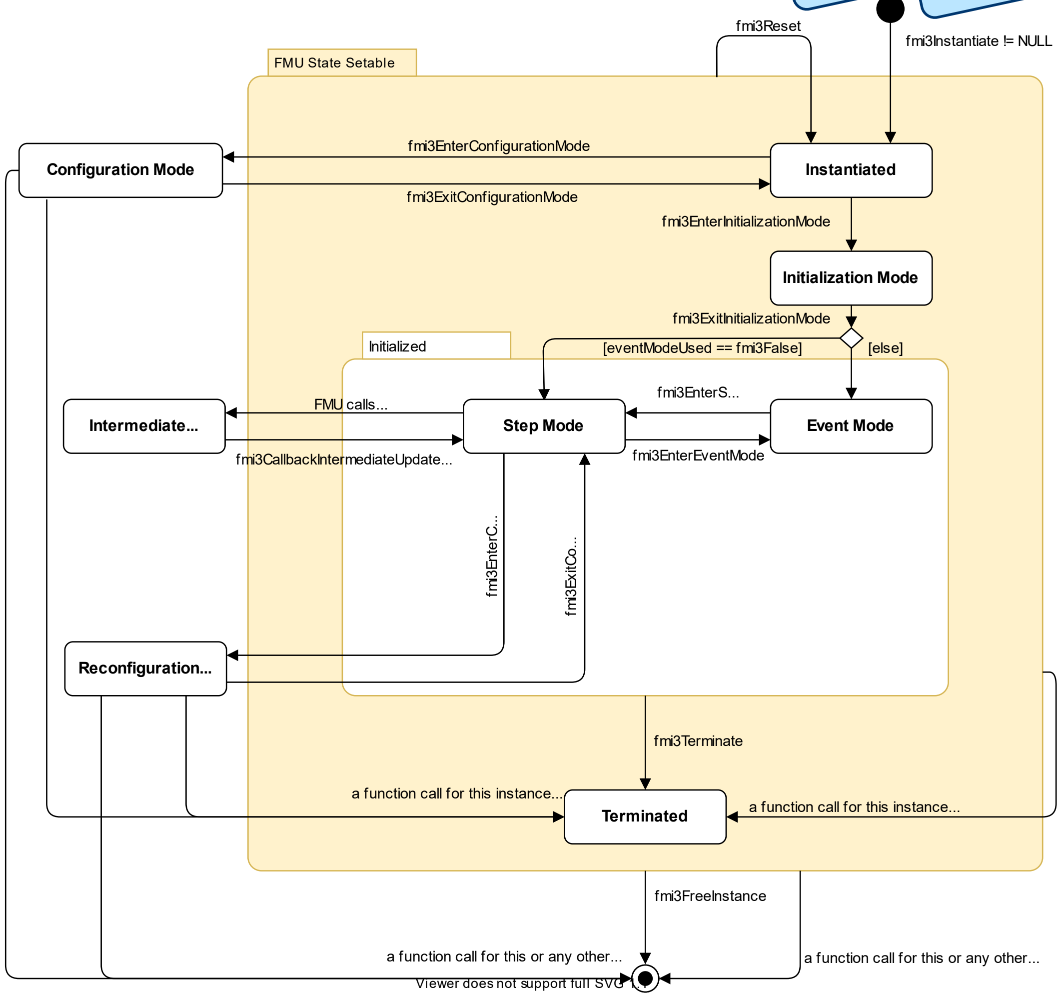
- Terminals group input and output variables to Terminals, which represent buses or physical connectors
- Predefined matching rules (`plug, bus, sequence`) for the whole Terminal, other rules are possible



- FMI Terminals **are not acausal**! The **causality (`input, output`) is defined** by the referenced variable!

FMI for Co-Simulation: Event Handling

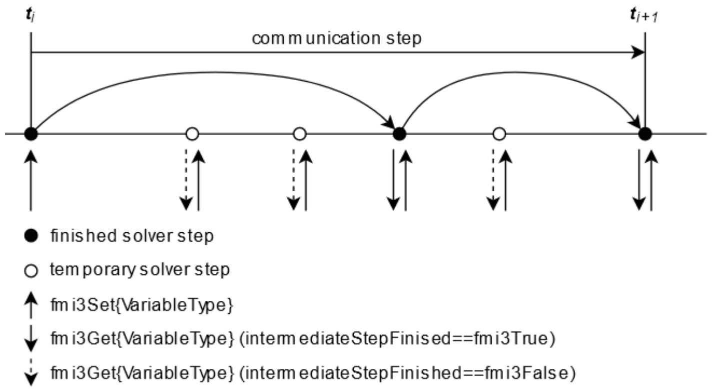
- Introduction of **Event Mode** like in Model Exchange
- **Event Mode** can be entered by the importer:
  - to handle input events triggered outside the FMU
  - on request of the FMU
- `fmi3DoStep(…)` can return before `communicationStepSize` is reached if an FMU internal event needs to be treated outside
- Capability flag `hasEventMode` signals if event handling is supported by the FMU
- Argument `eventModeUsed` of `fmi3InstantiateCoSimulation` signals if event handling is supported by the importer



FMI for Co-Simulation: Intermediate Variable Access

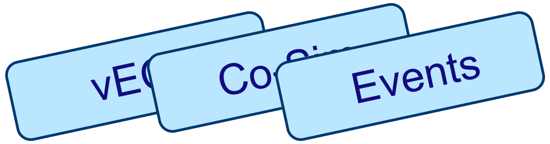
Co-Sim

- FMI1/2 CS: FMUs exchange values only at communication points.
- FMI3.0: allows information exchange between the FMU and the master also at intermediate time points
- Realized with an extension to the state machine: Intemeidate update mode which can be entered by the FMU via the callback function `fmi3CallbackIntermediateUpdate(…)` at arbitrary times out of `fmi3DoStep(…)`



- This enables various use cases:
  - Advanced co-simulation algorithms using interpolation / extrapolation techniques
  - Transmission Line Modeling (TLM) co-simulation
  - Input approximation similar to what is possible in FMI 2.0
  - More detailed plotting of simulation results (using additional time point)

Clocks



- Clocks synchronize FMUs with the importer and with other FMUs:
  - Clocks carry the information that a specific event happens
  - Clocked variables belong to one clock (a so-called clocked model partition). They change only if this clock is active.
- Clocks allow precise handling of time events (independent from continuous time: `fmi3SetTime()`, or arguments of `fmi3DoStep()`)

- In **Scheduled Execution** Communication Clocks are used:
  - by the importer to identify the specific partition which is to be executed
  - by the FMU to announce, which model partition wants to be scheduled

Clock Types

Events

Clock Type		causality	interval	Example
Time-based	**periodic clock**	input	constant	Clocked PI-controller with a defined constant interval
			fixed	Clocked PI-controller, interval is defined by periodic fmi3SetClock calls
			calculated	Clocked PI-controller, interval is defined by fixed parameter(s) of the FMU
			tunable	Clocked PI-controller, interval is defined by tuneable parameter(s) of the FMU
	aperiodic clock	input	changing	Simulation of the behaviour of a control algorithm with non constant execution time, Generation of pulse sequences
			countdown	Time delayed action after an event, for example ignition signal some time after crankshaft angle event
Triggered		input	triggered	Control algorithm, triggered by a crankshaft angle sensor
		output	triggered	Crankshaft angle sensor which ticks several times per revolution

Adjoint Derivatives

AI

- In several applications, including backpropagation for gradient-based training AI models, adjoint derivatives ("*vector gradient products" (VJPs))* are needed
- They can be implemented efficiently using revers mode automatic differentiation (AD)
- FMI 3.0 provides now two access functions for partial derivatives:
  - `fmi3GetDirectionalDerivative` to compute the directional derivatives $\mathbf{v}_{sensitivity} = \mathbf{J} \cdot \mathbf{v}_{seed}$, and
  - `fmi3GetAdjointDerivative` to calculate the adjoint derivatives $\mathbf{v}^T_{sensitivity} = \mathbf{v}^T_{seed} \cdot \mathbf{J}$

Benefit:

- This will allow to more efficiently encapsulate and train AI models with FMI
- Connection of the Python/Julia tool world of AI to the system simulation world
- Enabling the combination of physics-based and AI-based models (e.g. neural ODEs) and training in a unified framework

Functional Mock-Up Interface

Concept of Layered Standard

vECU

- The layered standard concept allows the specification of standards on top of FMI
- XML element annotations and strings allow additional semantic for variables and terminals
- Extra folder in FMU zip-file allows shipping of additional files at a well-defined place without disturbing compatibility

- Examples:
  - XCP: When packaging virtual electronic control units (vECUs), XCP allows standardized access (see ASAM) to ECU internal variables (in preparation on FMI GitHub)
  - Network2Signals: Allows grouping and description of FMU inputs and outputs as network signals (in preparation on FMI GitHub)
  - Including of 3D-Visualization to FMUs which represent multi body simulation models (prototype from ESI ITI and TU Dresden)

Miscellaneous

- Graphical representations for the whole FMU and for Terminals can be defined
- Alias variable names are now specified by a list of alias names for each variable and no longer by a separate variable with the same `valueReference`.
- Dependencies might change at runtime due to variable structure of the model or due to changes of array sizes. Dependencies for (array) variables can now be retrieved at runtime.
- Asynchronous execution of `fmi2DoStep` was removed for simplification. This feature was never used and can be implemented by the importer.
- Improvement and clarification of source code FMUs for better platform independency.

Roadmap

- FMI 3.0 Beta 2 is available now
- 2 PlugFests held, 2 more planned in 2021
- We are planning to release FMI 3.0 early 2022
- Resources:
  - Development process can be tracked on GitHub: https://github.com/modelica/fmi-standard
  - FMPy is permanently updated to support FMI 3.0: https://github.com/CATIA-Systems/FMPy
  - Reference FMUs: https://github.com/modelica/Reference-FMUs