Principles of Equation-Based Object-Oriented Modeling and Languages

Module D: Co-simulation and the Functional Mock-up Interface

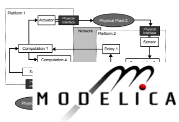
Mini-course, Scuola Superiore Sant'Anna, Pisa, Italy.
December 9-10, 2014

David Broman

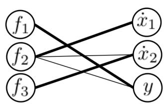
Associate Professor, KTH Royal Institute of Technology

Assistant Research Engineer, University of California, Berkeley

Course Structure



Module A
EOO Languages and Modelica Fundamentals



Module B
DAEs and Algorithms in EOO Languages

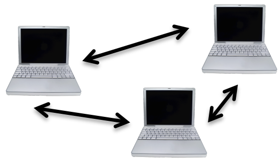
Module C
Modelyze – Defining Equation-Based DSLs

Module D
Co-simulation and the Functional Mock-up Interface

Agenda

Part I

Introduction to FMI

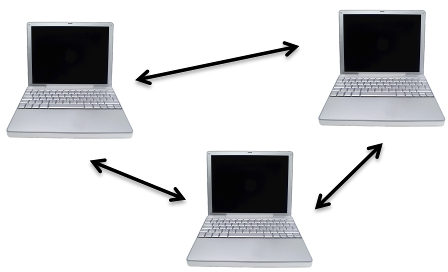


Part II

FMI Formalization and Master Algorithms

$$\begin{aligned}\texttt{init}_c &: \mathbb{R}_{\geq 0} \to S_c \\ \texttt{set}_c &: S_c \times U_c \times \mathbb{V} \to S_c \\ \texttt{get}_c &: S_c \times Y_c \to \mathbb{V} \\ \texttt{doStep}_c &: S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0}\end{aligned}$$

Part I
Introduction to FMI



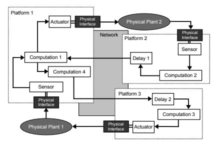
Cyber-Physical Systems Design

Model

$$\begin{aligned} J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

Equation-based model

FMI Co-simuation

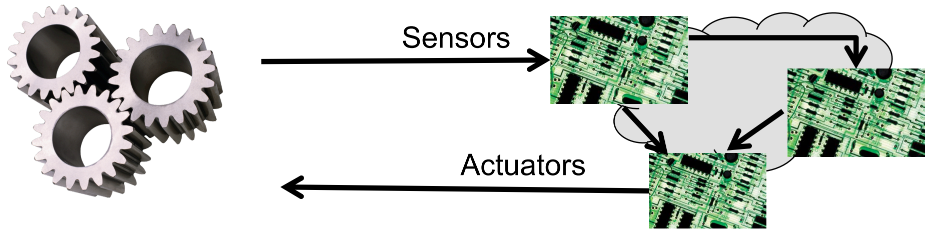


Various models of computation (MoC)

Modeling

Modeling

System



Physical system (the plant) Cyber system: Computation (embedded) + Networking

What is FMI?

FMI

Functional Mock-Up Interface (FMI) is a standard, not a tool.

Initiative from Daimler AG. Developed in a EU project called MODELISAR. Now maintained by Modelica Association.

Current version is 2.0.

Currently supported by more than 50 tools (open source and commercial).

Functional Mock-Up Unit (FMU) is a model instance that can me used in a simulation.

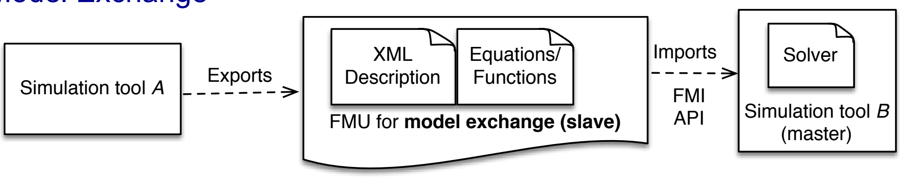
An FMU is a zip file containing:

- An XML file describing static info (e.g., port names, variables, supported properties)
- C-files and dynamically loadable libraries implementing the behavior.

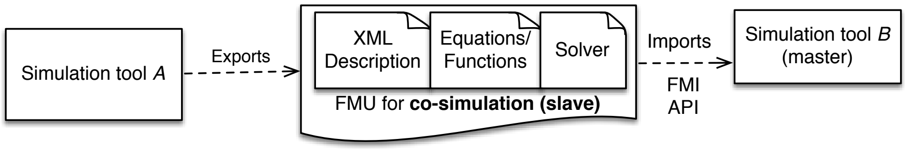
FMU

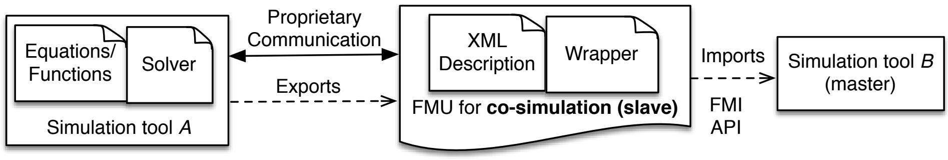
Model Exchange or Co-Simulation?

FMI for Model Exchange



FMI for Co-Simulation

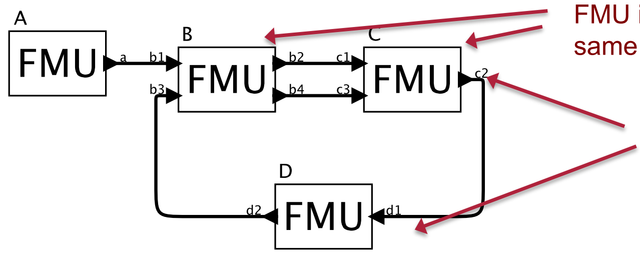




Part II
FMI Formalization and Master Algorithms

$$\begin{aligned}
\texttt{init}_c &: \mathbb{R}_{\geq 0} \to S_c \\
\texttt{set}_c &: S_c \times U_c \times \mathbb{V} \to S_c \\
\texttt{get}_c &: S_c \times Y_c \to \mathbb{V} \\
\texttt{doStep}_c &: S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0}
\end{aligned}$$

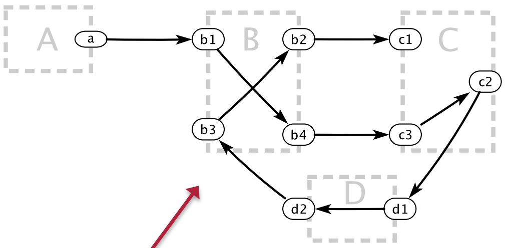
FMU Connections and I/O Dependencies



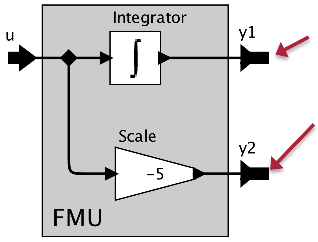
FMU instances. Can come from the same FMU

Input and output port variables

Can output be directly dependent on the input?

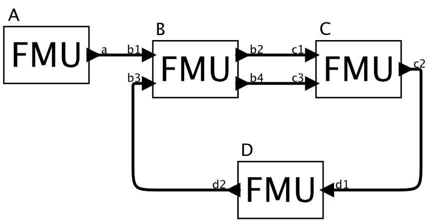


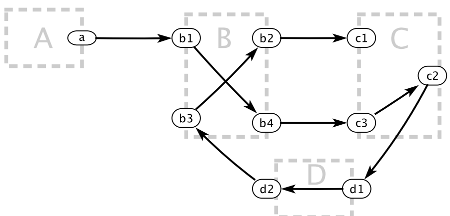
I/O dependencies together with port connections form a graph.



y1 does not have a direct dependency on u, but y2 does.

A FMI formalization (a subset of the standard)





Set of FMU instances in a model	C
FMU instance identifier	$c \in C$
Set of state valuations for instance c	S_c
Set of input port variables for instance c	U_c
Set of output port variables for instance c	Y_c
Set of values that a variable may take on	$\mathbb{V}$
I/O dependency for instance c	$D_c \subseteq U_c \times Y_c$
Set of all input variables in a model	$U = \bigcup_{c \in C} U_c$
Set of all output variables in a model	$Y = \bigcup_{c \in C} Y_c$
Set of all I/O dependencies	$D = \bigcup_{c \in C} D_c$
Port mapping	$P : U \to Y$

A FMI formalization (subset of standard)

$$\texttt{init}_c : \mathbb{R}_{\geq 0} \to S_c$$
$$\texttt{set}_c : S_c \times U_c \times \mathbb{V} \to S_c$$
$$\texttt{get}_c : S_c \times Y_c \to \mathbb{V}$$
$$\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0}$$

Set of FMU instances in a model	C
FMU instance identifier	$c \in C$
Set of state valuations for instance c	S_c
Set of input port variables for instance c	U_c
Set of output port variables for instance c	Y_c
Set of values that a variable may take on	$\mathbb{V}$
I/O dependency for instance c	$D_c \subseteq U_c \times Y_c$
Set of all input variables in a model	$U = \bigcup_{c \in C} U_c$
Set of all output variables in a model	$Y = \bigcup_{c \in C} Y_c$
Set of all I/O dependencies	$D = \bigcup_{c \in C} D_c$
Port mapping	$P : U \to Y$

(A0) If $\texttt{doStep}_c(s, h) = (s', h')$ then $0 \leq h' \leq h$.

(A1) If $\texttt{doStep}_c(s, h) = (s', h')$, then for any h'' where $0 \leq h'' \leq h'$, $\texttt{doStep}_c(s, h'') = (s'', h'')$ for some s''.

If $h' < h$, doStep rejected h. In such a case “roll-back” is needed.

FMI 2.0 restrictions not seen in previous versions

$$\texttt{init}_c : \mathbb{R}_{\geq 0} \to S_c$$
$$\texttt{set}_c : S_c \times U_c \times \mathbb{V} \to S_c$$
$$\texttt{get}_c : S_c \times Y_c \to \mathbb{V}$$
$$\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0}$$

Version 2.0 makes it impossible to implement a component with zero latency (this was not a restriction in previous versions!)

“There is the additional restriction in “slaveInitialized” state that it is not allowed to call fmi2GetXXX functions after fmi2SetXXX functions without an fmi2DoStep call in between.”
(FMI standard 2.0, July 25, 2014, page 104)

"... communication step size (hc). The latter must be > 0.0”
(FMI standard 2.0, July 25, 2014, page 100)

For FMI to support **hybrid co-simulation**, we believe that this restriction was a major mistake. We hope that future versions will improve this situation...

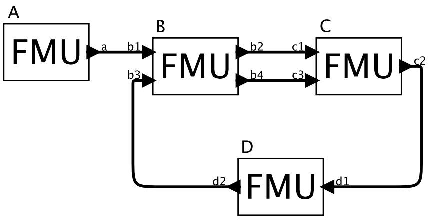
Master Algorithm

Master Algorithm (MA)

The MA orchestrates the execution of the FMUs

Master Algorithms are <u>not</u> part of the standard. It is "up to the tool" to implement it.

EMSOFT 2013 work. Define a MA that can be proven to terminate and is determinate.



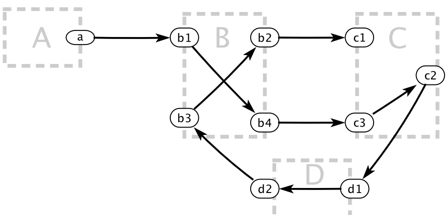
Algorithm 1: Order-Variables

Algorithm 1: *Order-Variables.*

Input: Port mapping P, global dependency relation D, and global set of variables $\mathbb{X}$. $\mathbb{X} = U \cup Y$

Output: An ordered list $\bar{x}$ of variables, or error.

1. Let G be a directed graph, where the vertices are represented by port variables $\mathbb{X}$ and an edge $e \in \mathbb{X} \times \mathbb{X}$ is a variable dependency. The set of all edges E is then constructed by $E = D \cup \{(y, u) \mid u \in U \wedge P(u) = y\}$.
2. Perform a topological sort on G. If a cycle in G is found, terminate and return error. If no cycles are found, the resulting list of variables is $\bar{x}$.



Algorithm 2: Master-Step

Algorithm 2: *Master-Step.*

Input: Set of instances C, ordered variable list $\bar{x}$, port mapping P, the maximal step size h_{max}, and a mutable state mapping m of size $|C|$.
Output: Updated state mapping m and the performed step size h.

1. Set values for all input variables:
   For each $u \in \bar{x}$ (in order) where $u \in U$ do
   (a) $y := P(u)$
   (b) $v := \texttt{get}_{c_y}(m[c_y], y)$
   (c) $m[c_u] := \texttt{set}_{c_u}(m[c_u], u, v)$
2. Save the states of all FMUs to enable rollback:
   $r := m$
3. Set communication step size to an initial default value:
   $h := h_{max}$
4. Find h acceptable by all FMUs:
   For each $c \in C$ do
   (a) $(s', h') := \texttt{doStep}_c(m[c], h_{max})$
   (b) $h := min(h, h')$
   (c) $m[c] := s'$
5. Assert $0 \leq h \leq h_{max}$ // follows from Assumption (A0)
6. If $h < h_{max}$ then // roll back and perform step h
   For each $c \in C$ do
   (a) $(s', h') := \texttt{doStep}(r[c], h)$
   (b) Assert $h' = h$ // follows from Assumption (A1)
   (c) $m[c] := s'$
7. Return m and h.

Predictable Step Size

Problems with Algorithm 2

- Requires that all FMUs support rollback
- Inefficient if the FMU can predict the next event.

Propose to extend FMI with one new function:

$$\texttt{getMaxStepSize}_c : S_c \to \mathbb{R}_{\geq 0} \cup \{\infty\}$$

Returns the upper bound of the communication step-size that an FMU can accept

C_P: Set of Predictable FMU Instances
C_R: Set of FMUs with rollback capabilities
C_L: Set of Legacy FMUs that neither implement rollback, nor are predictable.

(A4) If $c \in C_P$ and $s \in S_c$ and $\texttt{getMaxStepSize}_c(s) = h$ then for all h' where $0 \leq h' \leq h$, $\texttt{doStep}_c(s, h') = (s', h')$ for some s'.

(a) $|C_L| \leq 1$.
(b) $C_L \cup C_R \cup C_P = C$.
(c) $C_L \cap C_R = \emptyset$ and $C_R \cap C_P = \emptyset$ and $C_P \cap C_L = \emptyset$.

Algorithm 3: Master-Step with Predictable Step Size

Algorithm 3: *Master-Step With Predictable Step Sizes.*

Input: Set of instances C, ordered variable list $\bar{x}$, port mapping P, the maximal step size h_{max}, and a mutable state mapping m of size $|C|$.

Output: Updated state mapping m and the performed step size h.

1. Set values for all input variables:
   For each $u \in \bar{x}$ (in order) where $u \in U$ do
   (a) $y := P(u)$
   (b) $v := \texttt{get}_{c_y}(m[c_y], y)$
   (c) $m[c_u] := \texttt{set}_{c_u}(m[c_u], u, v)$
2. Find the minimal predictable communication size:
   $h := min(\{\texttt{getMaxStepSize}_c(m[c]) \mid c \in C_P\} \cup \{h_{max}\})$
3. Save the states for all instances that can perform rollback:
   (a) For each $c \in C_R$ do $r[c] := m[c]$
   (b) $doStepOnLegacy := \texttt{true}$
   (c) Goto step 5.
4. Restore states for rollback instances.
   For each $c \in C_R$ do $m[c] := r[c]$
5. Perform `doStep` on all instances with rollback:
   $h_{min} := h$
   For each $c \in C_R$ do
   (a) $(s', h') := \texttt{doStep}_c(m[c], h)$
   (b) $h_{min} := min(h', h_{min})$
   (c) $m[c] := s'$
6. If $h_{min} < h$ then $h := h_{min}$ and goto step 4.
7. Perform `doStep` on the legacy FMU (if it exists)
   If $c \in C_L$ and $doStepOnLegacy$ then
   (a) $(s', h') := \texttt{doStep}_c(m[c], h)$
   (b) $m[c] := s'$
   (c) $doStepOnLegacy := \texttt{false}$
   (d) If $h' < h$ then $h := h'$ and goto step 4.
8. Perform `doStep` on all FMUs with predictable step size:
   For each $c \in C_P$ do
   (a) $(s', h') := \texttt{doStep}_c(m[c], h)$
   (b) Assert $h' = h$ // follows from Assumption (A4)
   (c) $m[c] := s'$
9. Return m and h.

Reference Papers

- Blockwitz et al. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, In Proceedings of the 9th International Modelica Conference, 2012
  (General overview of an early version of 2.0, not the final release)
- David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate Composition of FMUs for Co-Simulation. In *Proceedings of the International Conference on Embedded Software (EMSOFT 2013)*, Montreal, Canada, 2013.
  (Defines the master algorithm and formalization that we presented here)
- David Broman, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. *Requirements for Hybrid Cosimultion*, EECS Technical report No. UCB/EECS-2014-157, UC Berkeley, August 14, 2014.
  (A document describing requirements and test cases for hybrid co-simulation)

Summary and Conclusions

Summary and Conclusions

Some key take away points:

- The **FMI standard** is used in both industry and academia.
- There are two main modes: FMI for **model exchange** and FMI for **co-simulation.**
- We have presented a **formalization** of a core of FMI and proposed master algorithms that are determinate.
- Note, however, that the latest FMI standard for co-simulation, have made **it impossible to encode such hybrid co-simulation correctly**. We hope that we can change this in the next version of the FMI standard.

Thanks for listening!