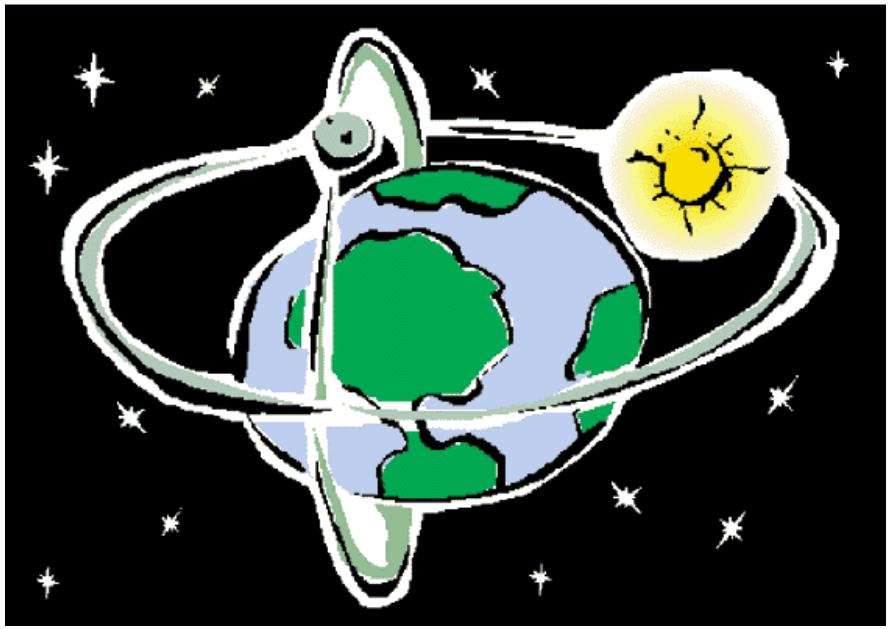
Automatic generation of master algorithms for FMI 2.0 Co-Simulation

Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks and Edward A. Lee

UC Berkeley



11th Biennial Ptolemy Miniconference

Berkeley, CA
October 16, 2015

Outline

- Simulation and design of a Cyber-Physical System: Co-Simulation.
- FMI, a standard for the exchange of models and Co-simulation of CPSs.
  - Lacks, formalization and extensions.
- A Master Algorithm for Discrete Event and Continuous time dynamics.
- FIDE – An FMI Integrated Design Environment.
  - Designed to test our proposed extension to the standard.

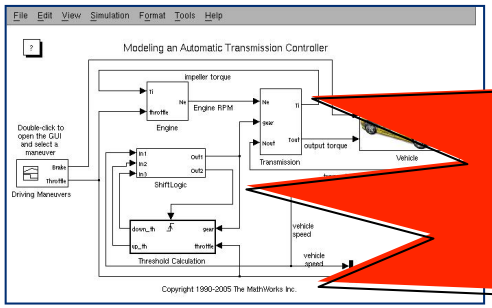
Simulation of Cyber-Physical Systems

- Cyber-physical systems (CPS)
  - **Cyber**: computer-based systems, Discrete Events (DE) Model of Computation (MoC)
  - **Physical**: the model of the environment, Continuous Time system (CT) MoC
- Heterogeneous modeling
  - The design involves a wide breath of components and different area of expertise
  - Automata, state machines, transition systems, dataflow, discrete event systems, timed automata, ODEs, DAEs, PDEs, hybrid automata, ...
  - Different modeling paradigms, languages and tools for different components

Simulation of Cyber-Physical Systems

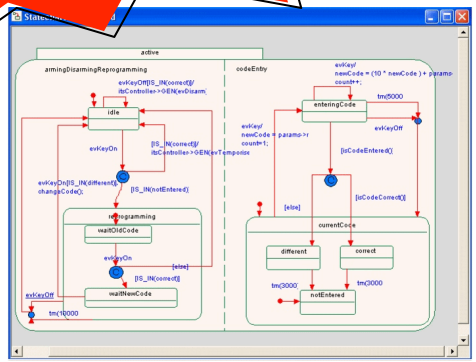
Cyber-Physical Systems: Cyber (digital, computer-based) + Physical

Typically heterogeneous modeling: state machines, discrete-event systems, differential equations, …

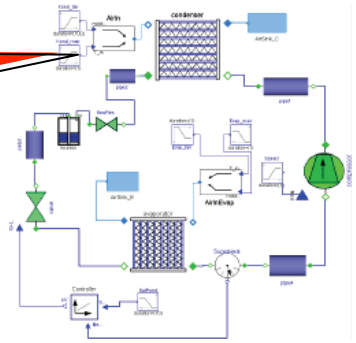


Low-level controllers
Simulink

How these tool interact?



Supervisory controllers
Rhapsody/
SysML



Physical dynamics
Modelica

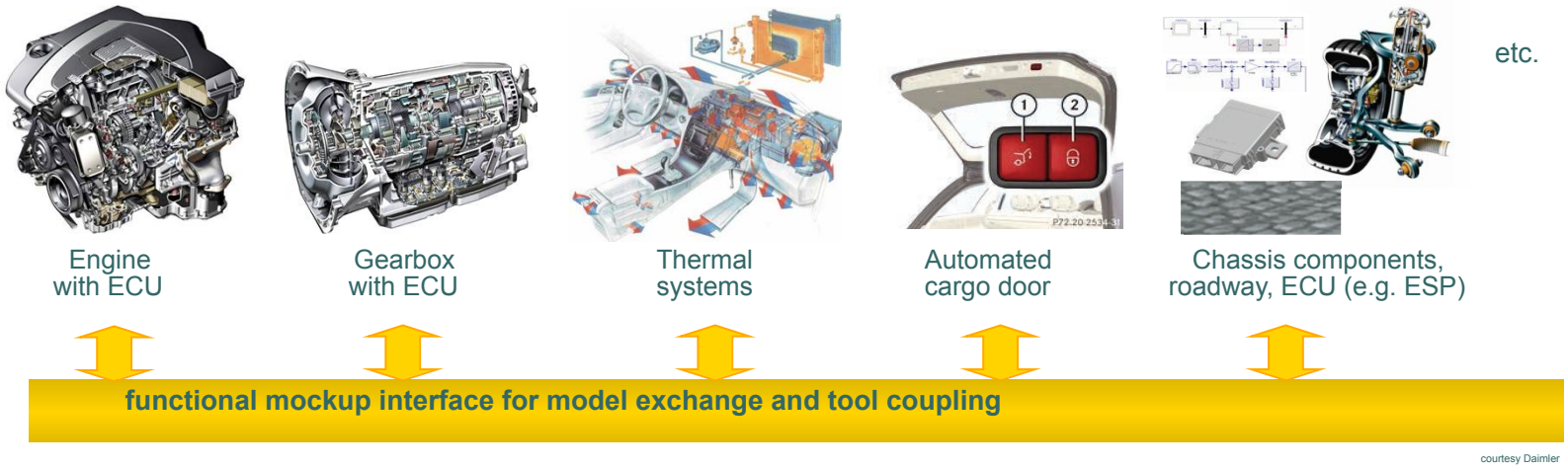
Slide courtesy of Stavros Tripakis

Co-Simulation

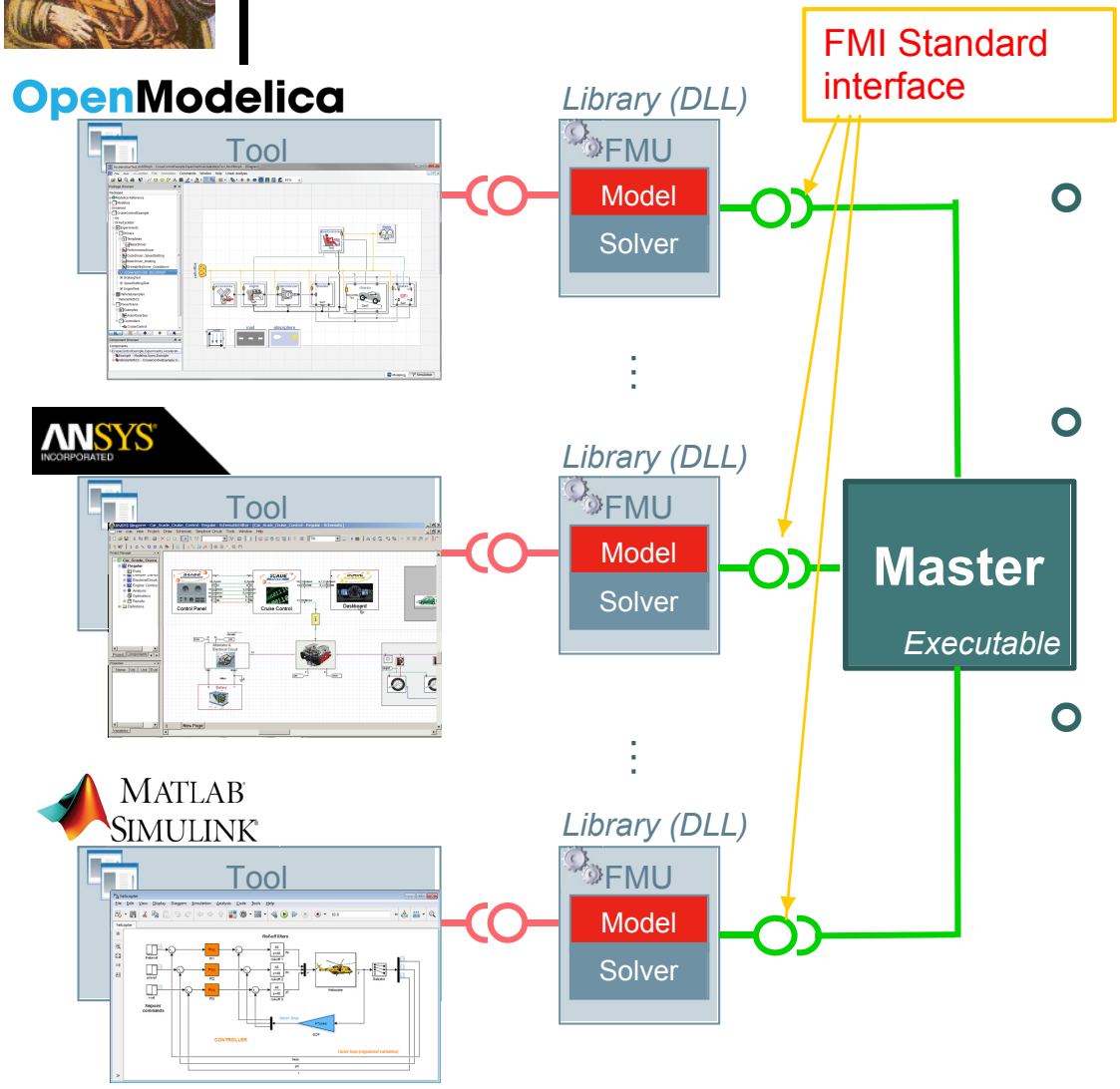
- Co-Simulation permits simulating individual components using different simulation tools simultaneously and collaboratively.
- Co-simulation does not require agreement between tools on the semantics of models.
- Requirements for Co-Simulation.
  - Semantics: a model of computation to orchestrate the exchange of data and the advancement of time.
  - Software engineering: a standard interface for the components.

Functional Mock-up Interface (FMI)

- A tool independent standard for interoperable models
  - **Model exchange**: Model descriptions (FMUs) from one tool are interpreted and executed in another.
  - **Co-simulation**: Models created in one tool (FMUs) are executed within another tool.



FMI 2.0 Co-Simulation



- Requires agreement on the semantics of the interface.
- FMU is a self-contained object: model + simulation engine provided by the design environment.
- The FMUs are orchestrated by a MA that deterministically exchange data and advance the simulation time.

Limitations in FMI 2.0 Co-Simulation

- Super-dense time? **NO!**
- $t \in \boldsymbol{R}$
- FMI uses **`double`** as data type for Real Numbers
  - Timing precision is dependent on the magnitude of time, and operations on time incur complex quantization effects.

```
double r = 0.8;
double k = 0.7;
k = k + 0.1;
printf("%f,%f,%d\n",r,k,r==k);
```

```
0.800000,0.800000,0
```

Limitations in FMI 2.0 Co-Simulation

- FMI defines five platform dependent data types for I/O and state variables:
  - fmi2Real, fmi2Integer, fmi2Boolean, fmi2String, fmi2Char, fmi2Byte
- These are continuous time variables.
- FMI lacks a notion of "**absent**" value, something that is essential for discrete-event and synchronous-reactive systems.

Limitations in FMI 2.0 Co-Simulation

- How to advance time in FMI?
- In Ptolemy II: **`fireAt()`**… it is pro-active! An actor asks to a director to be executed.
- In FMI: **`doStep()`**… the MA propose an advancement of time.
- An FMU can rejected or partially accept a step size!
- Roll-back?

Limitations in FMI 2.0 Co-Simulation

- What MoC for the Master Algorithm?
  - Not specified.
- Discrete Events?
  - No “absent”, no super-dense time
- Synchronous Reactive?
  - No “absent”, no super-dense time and no notion of “unknown”.
- Only sampled-data systems are well supported:
  - The step size can be fixed!

Some extensions to FMI 2.0 Co-simulation

Determinate Composition of FMUs for Co-Simulation

David Broman[1,2] Christopher Brooks[1] Lev Greenberg[3] Edward A. Lee[1]
Michael Masin[3] Stavros Tripakis[1] Michael Wetter[4]
{broman,cxh,eal,stavros}@eecs.berkeley.edu, {levg,michaelm}@il.ibm.com, mwetter@lbl.gov
[1]University of California, Berkeley, USA [2]Linköping University, Sweden
[3]IBM [4]LBNL, Berkeley, CA, USA

ABSTRACT

In this paper, we explain how to achieve deterministic execution of FMUs (Functional Mockup Units) under the FMI (Functional Mockup Interface) standard. In particular, we

model are either memoryless or implement one of rollback or step-size prediction. We show further that such a model can contain at most one "legacy" FMU that is not memoryless and provides neither rollback nor step-size prediction.

Requirements for Hybrid Cosimulation Standards*

David Broman
KTH Royal Institute of Technology & UC Berkeley

Lev Greenberg
IBM Research – Haifa, Israel

Edward A. Lee†
UC Berkeley

Michael Masin
IBM Research – Haifa, Israel

Stavros Tripakis
UC Berkeley & Aalto University

Michael Wetter
Lawrence Berkeley National Laboratory

ABSTRACT

This paper defines a suite of requirements for future hybrid cosimulation standards, and specifically provides guidance for development of a hybrid cosimulation version of the

schema for describing components. An **FMU** (Functional Mock-up Unit) is a component, typically exported from a modeling and simulation tool, that can be instantiated and used as part of a simulation in another modeling tool. To

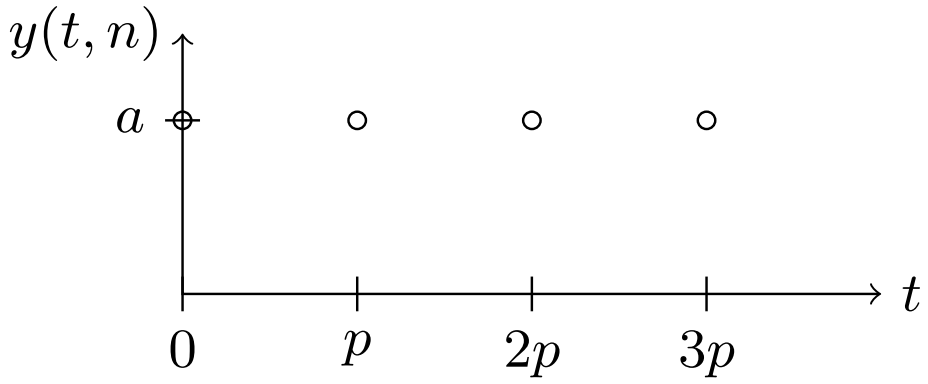
Super-dense time

- FMU contract for DE semantics.

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

The need for **absent** to simulate DE dynamics.

- Each FMI data type has been enriched with the absent value:
  - $V' = V \cup \{\boldsymbol{\varepsilon}\}$
- This signal is always absent except at **t = k*p**, $k \in N$.



Efficient roll-back

- An efficient roll-back is the one you don't have to do!
- FMU contracts for efficient rollback: predictable step size!

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

+

(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''.

=

No need for roll-back

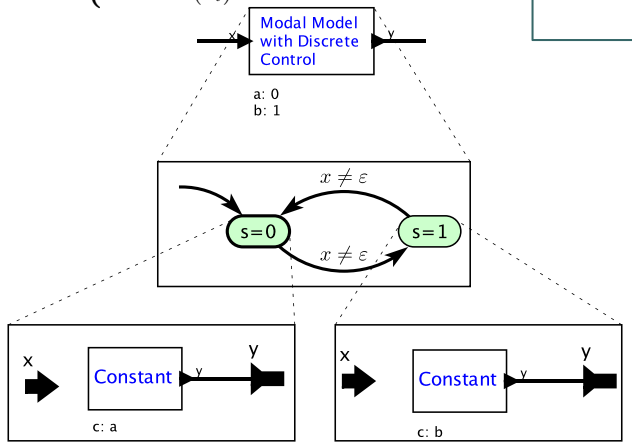
Some test components

- Test components and test compositions to evaluate the hybrid behavior.

Modal Model with Discrete Control

$$y(t,n) = \begin{cases} a & \text{if } s(t,n) = 0 \\ b & \text{otherwise.} \end{cases}$$

$$s(t,n) = \begin{cases} 0 & \text{if no such } i \text{ exists} \\ 1 & \text{if } s(d_i) = 0 \\ 0 & \text{if } s(d_i) = 1 \end{cases}$$



Adder

Input signals x_1 and x_2. **Output** signal y.

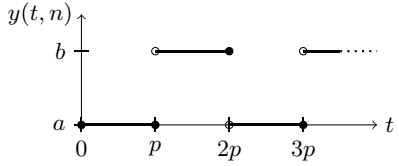
For all $\tau \in T$,

$$y(\tau) = \begin{cases} x_1(\tau) + x_2(\tau) & \text{if } x_1(\tau) \neq \varepsilon \text{ and } x_2(\tau) \neq \varepsilon \\ x_1(\tau) & \text{if } x_1(\tau) \neq \varepsilon \text{ and } x_2(\tau) = \varepsilon \\ x_2(\tau) & \text{if } x_1(\tau) = \varepsilon \text{ and } x_2(\tau) \neq \varepsilon \\ \varepsilon & \text{otherwise} \end{cases}$$

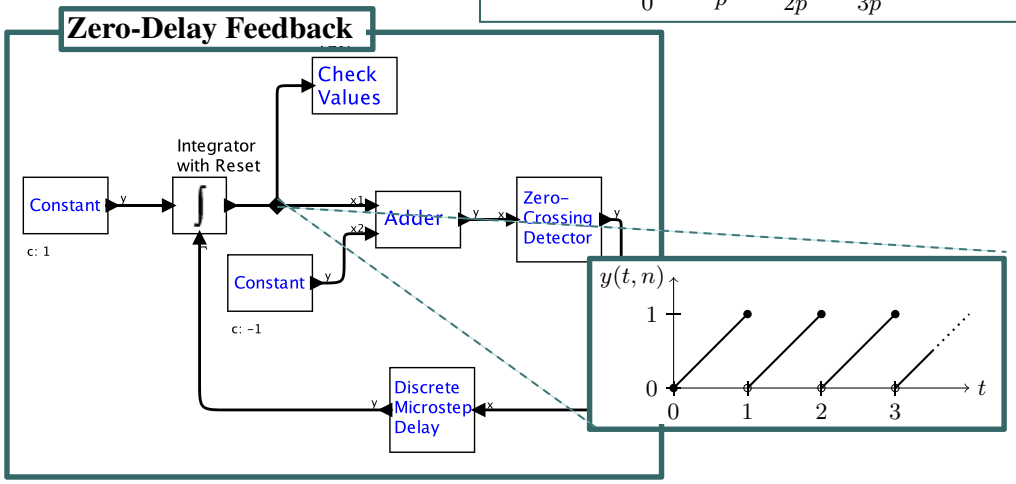
Periodic Piecewise Constant Signal Generator

For all $\tau \in T$,

$$y(t,n) = \begin{cases} a & \text{if } kp < t < (k+1)p \text{ and } k \in \mathbb{N} \text{ is even;} \\ b & \text{if } kp < t < (k+1)p \text{ and } k \in \mathbb{N} \text{ is odd;} \\ b & \text{if } t \text{ is an odd multiple of } p \text{ and } n \geq 1; \\ b & \text{if } t \text{ is an even multiple of } p,\ t > 0, \quad n = 0; \\ a & \text{otherwise.} \end{cases}$$



Zero-Delay Feedback



Open problem

- How to represent **time**?
  - This is still not solved.
  - What we know is that floating point numbers are not suitable to encode simultaneity of events.

FMI in a Nutshell

Notation:

C	set of FMU instances in a model
$c \in C$	FMU instance
S_c	set of states of FMU c
U_c	set of input ports of c
Y_c	set of output ports of c
$\mathbb{V}$	set of values that a port can take

API's main functions:

$$\texttt{init}_c : \mathbb{R}_{\geq 0} \to S_c \qquad \texttt{init}_c(t) \mapsto s$$

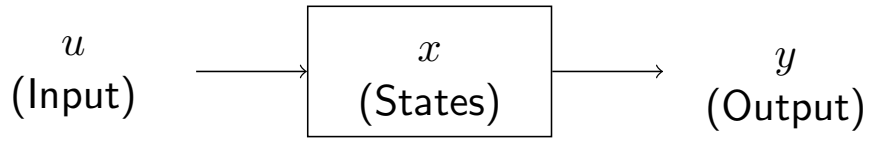
$$\texttt{set}_c : S_c \times U_c \times \mathbb{V} \to S_c \qquad \texttt{set}_c(s, u, v) \mapsto s$$

$$\texttt{get}_c : S_c \times Y_c \to \mathbb{V} \qquad \texttt{get}_c(s, y) \mapsto v$$

$$\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c \times \mathbb{R}_{\geq 0} \qquad \texttt{doStep}_c(s, h) \mapsto (s', h')$$

What is an FMU?

- It is a black-box.
- It is a Mealy machine.
- A standard API to interact with the black box: set inputs, get outputs, advance state.



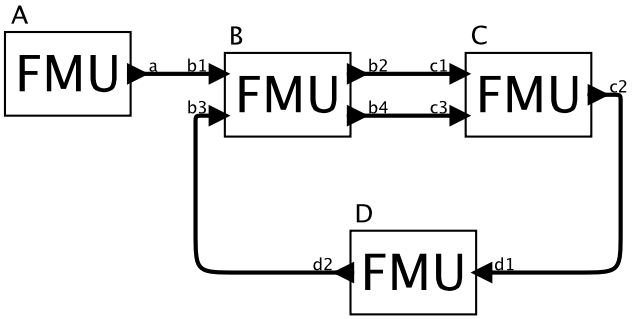
$\texttt{set}_c(s, u, v) \mapsto s$

$\texttt{get}_c(s, y) \mapsto v$

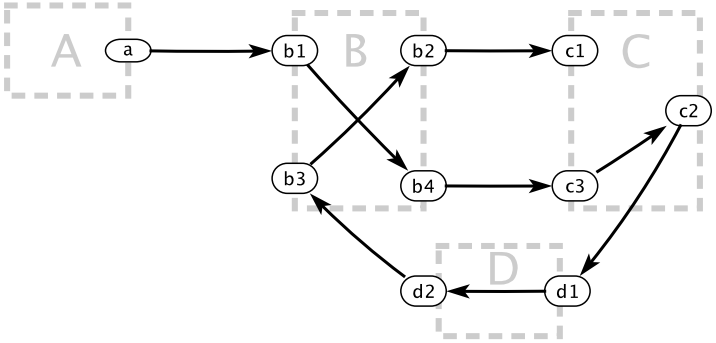
$\texttt{doStep}_c(s, h) \mapsto (s', h')$

Models with feedback

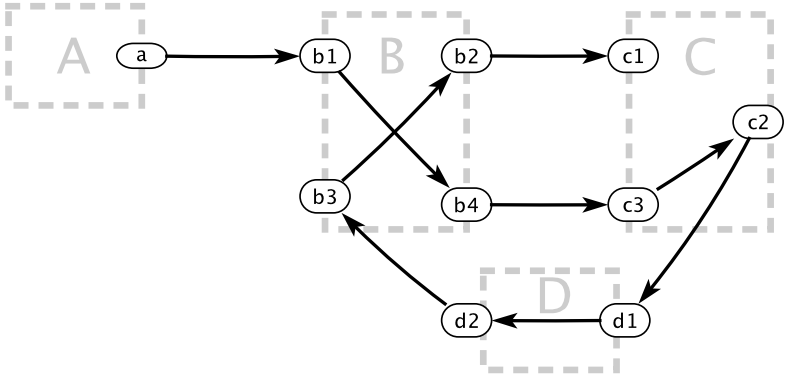
- How to execute a model with feedback?



- Restriction to models with no cyclic dependencies.



Models with feedback

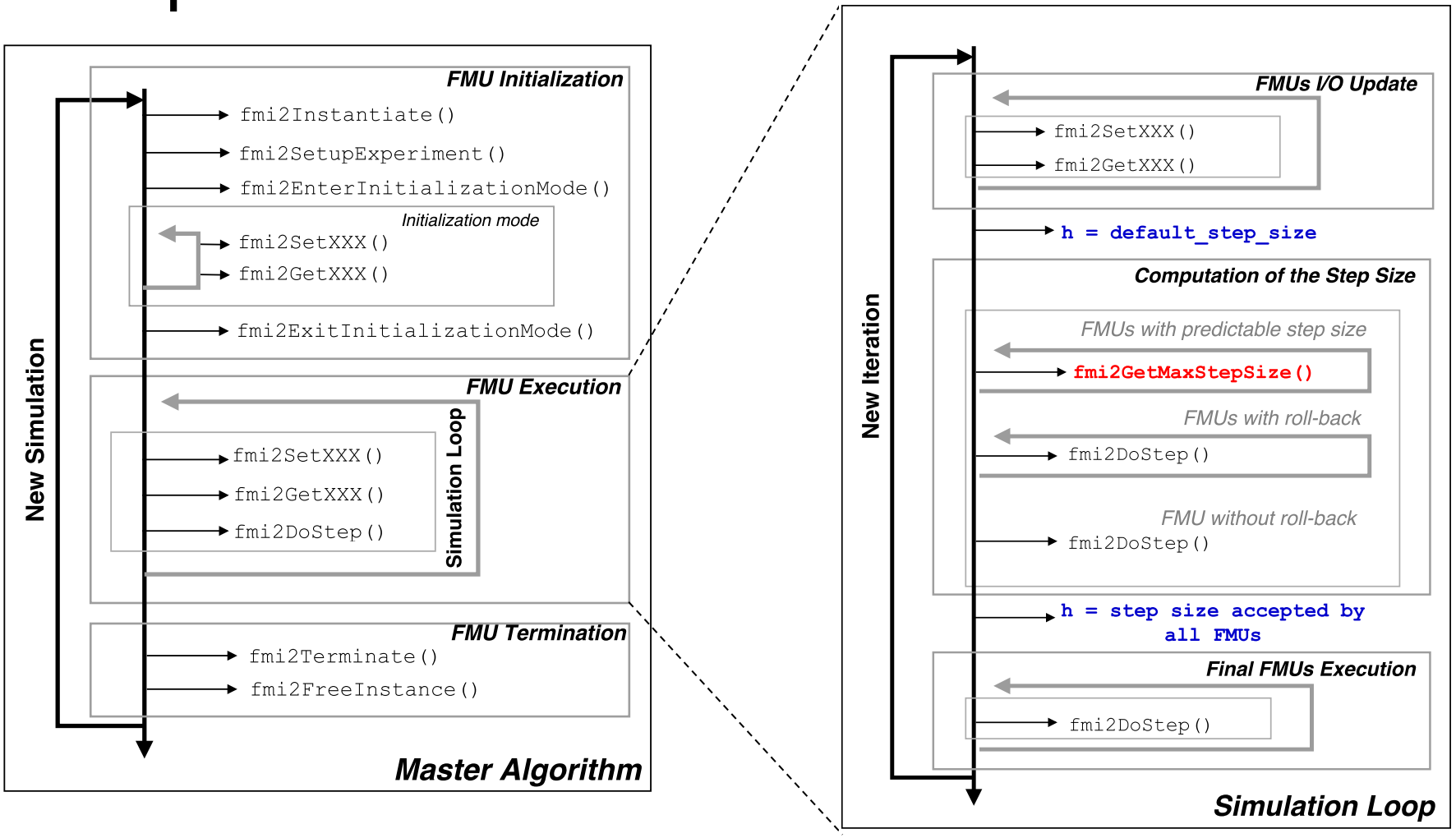


- **`get`** known outputs → **`set`** dependent inputs → repeat (while respecting the dependencies), until all I/O ports are set → update (**`doStep`**) the states of all FMUs by calling **`doStep`**s (we'll see how).
- FMI provides a mechanism for an FMU to declare I/O dependencies.
  - This allows the determination of a total order for I/O port update.

Updating the FMU States

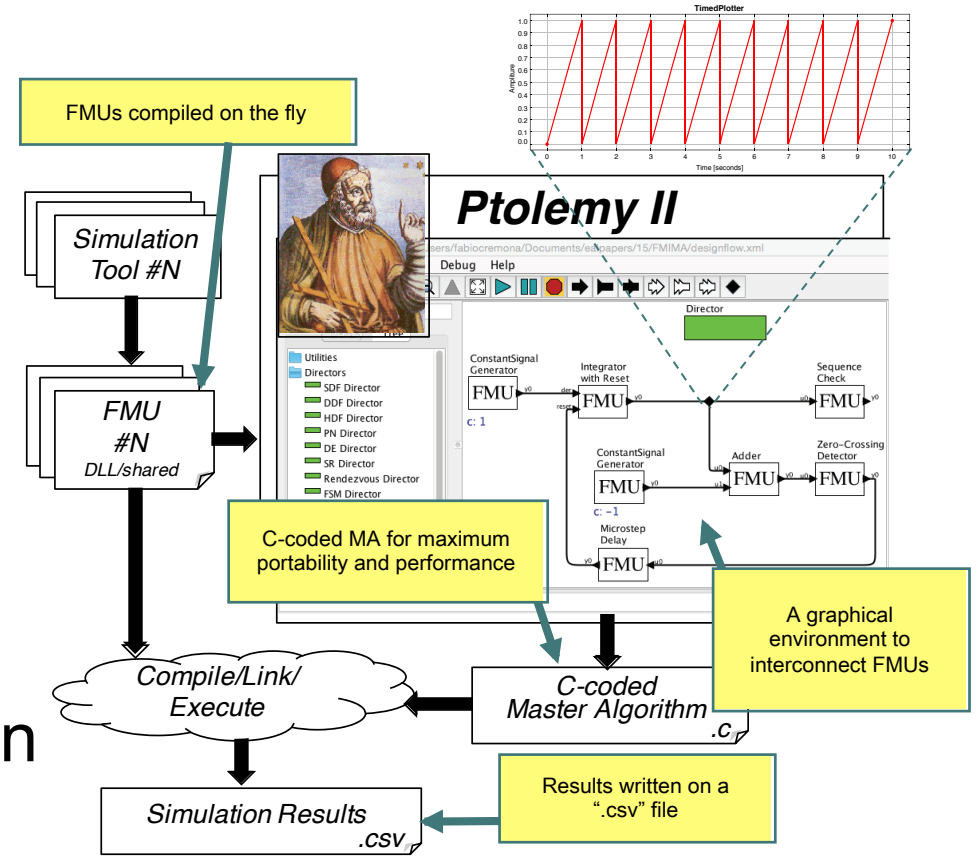
- In what order should **doStep**$_{\texttt{FMU1}}$ and **doStep**$_{\texttt{FMU2}}$ be called?
- Suppose **doStep**$_{\texttt{FMU1}}$ than **doStep**$_{\texttt{FMU2}}$:
  - What if FM1 accept ***h*** but FMU2 reject it?
- Suppose **doStep**$_{\texttt{FMU2}}$ than **doStep**$_{\texttt{FMU1}}$:
  - What if FM2 accept ***h*** but FMU1 reject it?
- That's why we need an efficient mechanism for roll-back!

A Master Algorithm for DE and CT MoC



FIDE – An FMI Integrated Design Environment

- Imports FMUs as FMU-actors with input and output ports.
- Arrange and interconnect FMU-actors through a graphical user interface.
- Co-simulate a composition of FMUs using an implementation of the MA. The MA is generated as C-code that can be compiled and executed outside Ptolemy II with benefits in performance and portability.



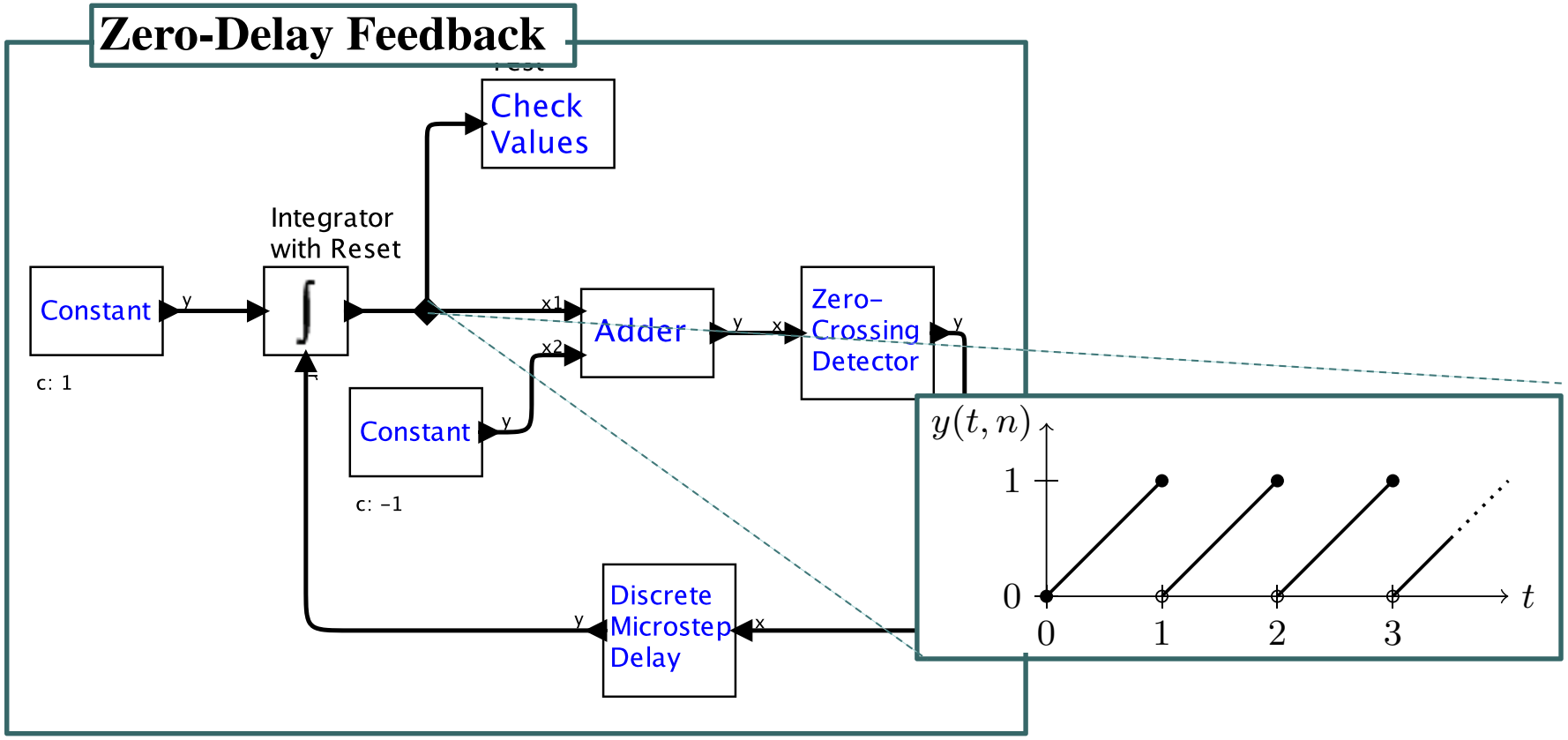
FIDE – An FMI Integrated Design Environment

- Super-dense time
  - Multiple iterations at the same time synchronization point
- "**absent**"
  - The FMI API has been extended to introduce

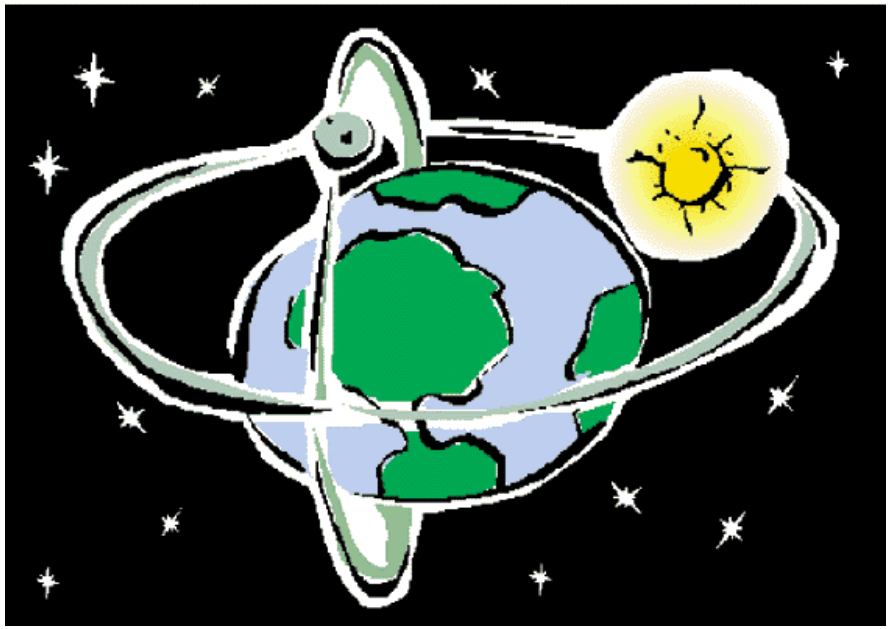
```
fmi2Status fmiSetHybridXXX (fmi2Component c, const fmi2ValueReference vr[],
    int nvr, const fmi2XXX value[], const fmi2SignalStatus status[]);


fmi2Status fmi2GetHybridXXX (fmi2Component c, const fmi2ValueReference vr[],
    int nvr, fmi2XXX value[], fmi2SignalStatus status[]);

typedef enum {
  PRESENT,
  ABSENT,
  UNKNOWN,
  PRESENT_THEN_ABSENT
} fmi2SignalStatus;
```

Though for future extensions

Example


Zero-Delay Feedback
Check Values
Integrator with Reset
Constant
c: 1
Adder
Zero-Crossing Detector
Constant
c: -1
Discrete Microstep Delay
x1
x2
y
x
$y(t,n)$
1
0
t
0
1
2
3

Thanks!



11th Biennial Ptolemy Miniconference

Berkeley, CA
October 16, 2015