Technical Report – April 07, 2016

Toward an Hybrid Co-simulation with the FMI-CS Standard

Jean-Philippe Tavella[1], Mathieu Caujolle[1], Charles Tan[1],
Gilles Plessis[2], Mathieu Schumann[2],
Stéphane Vialle[3], Cherifa Dad[3],
Arnaud Cuccuru[4], Sébastien Revol[4]

[1] EDF Lab Paris-Saclay, 91120 Palaiseau, France *firstname.lastname@edf.fr*
[2] EDF Lab Les Renardières, 77250 Écuelles, France *firstname.lastname@edf.fr*
[3] UMI 2958 GeorgiaTech-CNRS, CentraleSupelec, Université Paris-Saclay, 57070 Metz, France *firstname.lastname@centralesupelec.fr*
[4] CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems, P.C. 174, Gif-sur-Yvette, 91191, France *firstname.lastname@cea.fr*

Abstract

FMI (Functional Mock-up Interface) is a standard initiated by Daimler AG within the ITEA2 MODELISAR project, and is now maintained by the Modelica Association. It has been designed to enable the exchange of source models and the co-simulation of executable models exported from more and more modeling tools.

In FMI-CS (FMI for Co-Simulation), a component is a self-contained object that besides the model description also includes a numerical solver provided by design environment from where it comes.

A co-simulation may implies lots of FMUs and during a macro-step of the system simulation, each FMU independently simulates part of the system and at the end of each macro-step, the outputs from some FMUs provide new initial values (or inputs) to some other FMUs.

Unhappily, the current version 2.0 of the FMI-CS standard does not handle correctly all kind of signals especially in an hybrid co-simulation context.

The purpose of this paper is to propose some extensions to the FMI-CS 2.0 standard and to detail a simple use case in order to test these extensions in a co-simulation involving some major tools implementing the FMI-CS standard at the FMU side and DACCOSIM at the Master Algorithm side.

This work has been presented in the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

Contents

1 Introduction

Complex systems are characterized by the interconnection of numerous and heterogeneous cyber components (e.g. controllers) and physical components (e.g. power grids).
For EDF (the major French utility company), the smart power grids will extensively rely on new control functions (i) to increase the grid efficiency, reliability, and safety, (ii) to enable better integration of new assets (e.g. distributed generation and alternative energy sources), (iii) to support market dynamics and manage new interactions between established and new energy players.

Cyber-Physical Systems (CPS) design involves multiple teams working simultaneously on different aspects of the system. Especially, the development of a smarter electrical grid requires to reuse models and tools often based on separate areas of expertise.
This heterogeneity led EDF to investigate coupling standards such as the *Functional Mock-up Interface* (FMI[1]) initiated by Daimler AG within the ITEA2 MODELISAR project and now maintained by the *Modelica Association*[2]. More precisely, EDF chose the FMI-CS (FMI for Co-Simulation) part of the standard because this operation mode allows to export models as active components called FMUs (Functional Mock-up Units), each FMU being a self-contained archive file including a model and a numerical solver. As an additional benefit, the model IP is readily protected when models are exported as FMUs from FMI-CS compliant modelling tools.

For EDF, it is vital to develop agile modelling and simulation in order to design and validate distributed operating functions a long time before performing tests on experimental sites. The *Distributed Architecture for Controlled CO-SIMulation* (DACCOSIM[3] software) developed by EDF and CentraleSupelec is a part of the answer to this problematic. It is aimed at simulating large and complex multiphysics and hybrid systems on multi-core PC clusters fully exploiting all the potential of the FMI-CS standard. Unhappily the current FMI-CS 2.0 is not enough for hybrid co-simulation and this paper introduces improvements to be proposed to the FMI user group in charge of maintaining the FMI standard at the *Modelica Association*.
These researches are carried out by the RISEGrid[4] institute, founded by EDF and CentraleSupelec with the ultimate goal to design the smart electric grids of the future. Some others parties may also collaborate with RISEGrid, especially CEA List.

[1] https://www.fmi-standard.org/downloads
[2] https://www.modelica.org/
[3] https://daccosim.foundry.supelec.fr/
[4] http://www.supelec.fr/342_p_36889/risegrid.html

2 Challenges and requirements

FMI-CS is an attempt to define a unifying kernel for interoperability between active components through a standardized interface. Information about the standard with a list of compatible tools, the original specification of the standard, and a list of related publications can be found on the FMI website[5].
The problem with the FMI standard latest version FMI 2.0 is that it has been designed for dynamic systems with only *continuous and differentiable time signals*.

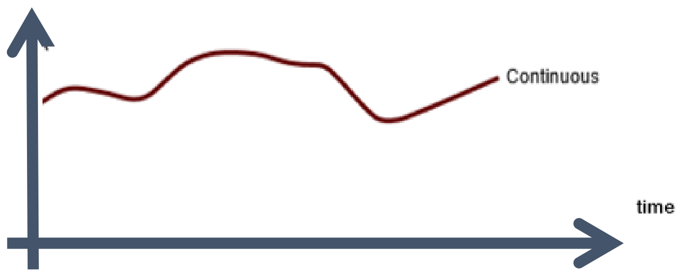


Figure 1: this signal is continuous and differentiable at any point in its range of definition

In a more general context of co-simulation, hybrid signals are to be also considered. Different examples are given in the next paragraphs.

2.1 Continuous & piecewise differentiable signals

In this case, the signals are:

- Present at each time $t_i \in \mathbb{R}^+$;
- Continuous on $\mathbb{R}^+$;
- Not differentiable in some points t_i : $\lim_{\varepsilon \to 0} f'(t_i - \varepsilon) \neq \lim_{\varepsilon \to 0} f'(t_i + \varepsilon)$.

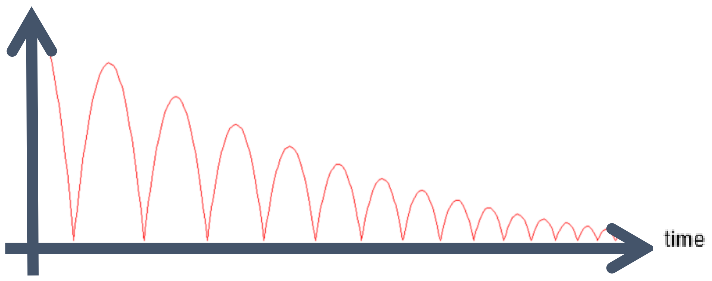


Figure 2: this signal is not differentiable at all the points having a null ordinate

2.2 Piecewise constant signals

In this case, the signals are:

- Present at each time $t_i \in \mathbb{R}^+$;
- Constant on disjoint and continuous time slots I^i so that: $\bigcup_i I^i = \mathbb{R}^+$;
- With a discontinuity appearing at each time slot switch.

[5] https://www.fmi-standard.org

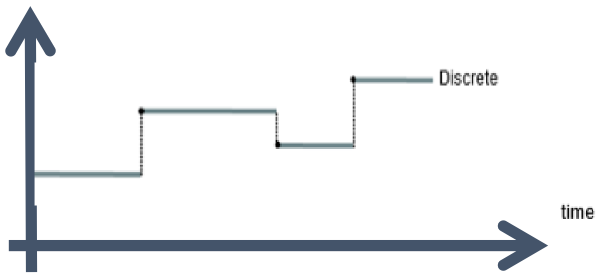


Figure 3: this integer (or real) signal is defined on 4 time slots

2.3 Piecewise continuous & differentiable signals

In this case, the signals are:

- Present at each time $t_i \in \mathbb{R}^+$;
- Not continuous and then not differentiable at some points t_i : $\lim_{\mathcal{E}\to 0} f(t_i - \mathcal{E}) \neq \lim_{\mathcal{E}\to 0} f(t_i + \mathcal{E})$.

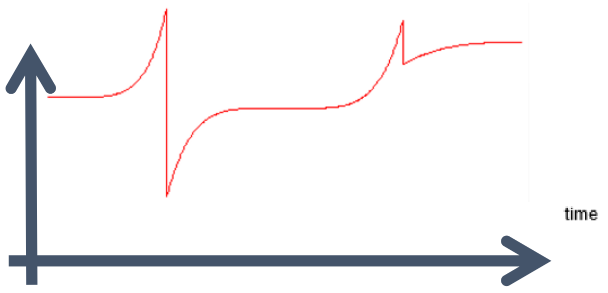


Figure 4: this signal is not continuous and not differentiable at two points

2.4 Discrete event signals

In this case:

- The signals are present for a set of definition $\mathcal{D}$ being a discrete time set $t_i \in \mathcal{D}$ with $\mathcal{D} \subset \mathbb{R}^+$;
- These signals can be confused with the events they generate.

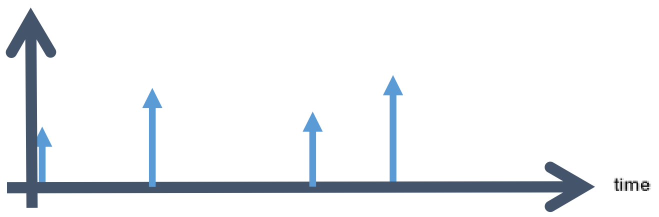


Figure 5: an event is generated at each discontinuity of this discrete signal

3 DACCOSIM environment

As stated in [1], DACCOSIM has been designed to achieve multi-simulations of continuous time systems, discretized with *time steps*, and running solvers using constant or variable time steps (to maintain the requested accuracy with the minimal amount of computations, whatever the dynamic of the system). It consists in two complementary parts: a *Graphic User-friendly Interface* (GUI) and a *dedicated computation package*.
The GUI developed in Java facilitates the complex systems studies by designing the multi-simulation graph (Fig. 6), i.e. the FMUs involved and the variables exchanged in-between, defining the resources used by the simulation (local machine or cluster), configuring the simulation case (duration, co-initialization method, time step control strategy...) and implementing the graph into DACCOSIM master tasks managing the simulation.
The *dedicated computation package* controls all task execution issues relative to the multi-simulation: co-initialization, local or distributed computation steps, fixed or variable time step control strategies, detection of state events generated inside FMUs, inter-FMU communications, distributed and hierarchical decision process...The Java version of DACCOSIM relies on JavaFMI[6] and is available for both Windows and Linux operating systems, whether 32-bit or 64-bit.

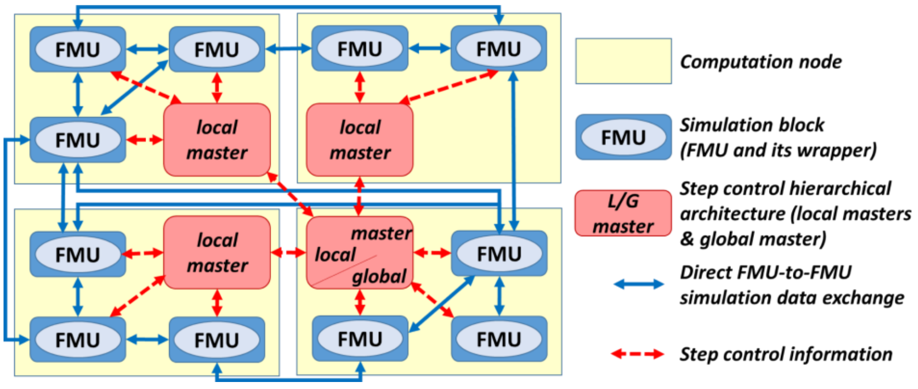

FMU
local master
master local global
Computation node
Simulation block (FMU and its wrapper)
L/G master
Step control hierarchical architecture (local masters & global master)
Direct FMU-to-FMU simulation data exchange
Step control information

Figure 6: distributed DACCOSIM architecture with a hierarchical *master*

With DACCOSIM as with any other Master Algorithm based on the FMI-CS 2.0, the event handling is approximate even with the help of the rollback feature. Typically, events are approached by small steps or a bisectional search (Fig. 7).

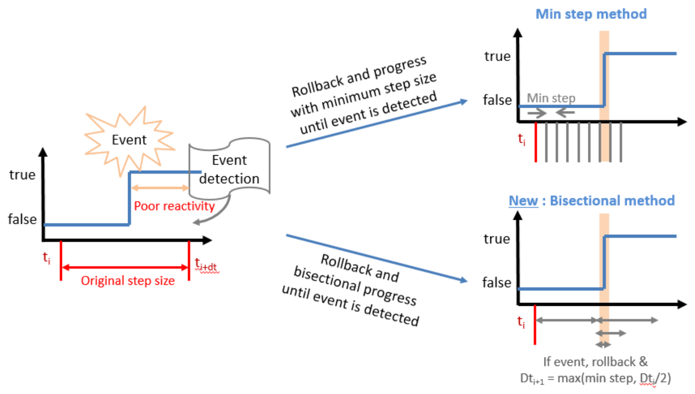

Event
Event detection
true
false
Poor reactivity
t_i
t_i+dt
Original step size
Rollback and progress with minimum step size until event is detected
Rollback and bisectional progress until event is detected
Min step method
Min step
New : Bisectional method
If event, rollback &
Dt_i+1 = max(min step, Dt_i/2)

Figure 7: the FMI-CS 2.0 only allows to approach the event handling

[6] https://bitbucket.org/siani/javafmi/wiki/Home

4 Some proposals to improve the FMI-CS 2.0

Hybrid co-simulation mix components whatever the nature of the associated input or output signals. A particular case is given when all the signals are present at each time:

- With continuous simulators:
  - These components cannot indicate in advance the date of their *state events*
  - It's typically the case with FMUs exported from Dymola
- With pure discrete simulators:
  - Each component can theoretically indicate in advance the date of its *time events*
  - It's potentially the case with FMUs exported from Control Build or Papyrus
- With mixed simulators:
  - In the most general case, mixed (discrete-continuous) components can generate *state events* and *time events*

A generalization of this case involves to be able to handle signals that can be present only at some times. It's typically the case when telecom simulators are to be added to the co-simulation.

4.1 A new function *fmi21DoStep()*

The goal here is to precisely achieve *state events* whose date cannot be predicted without an exploration of the future (unpredictable break-point).

Here are some details of the proposal :

- Prototyping : *fmi21DoStep(stepSize, nextEventInstant) ;*
- The solver integration stops at the first unpredictible event with a new return code *fmi21Event* and the time event is given by the returned value *nextEventInstant;*
- The Master Algorithm can go on to integrate state variables without rollback on this component.

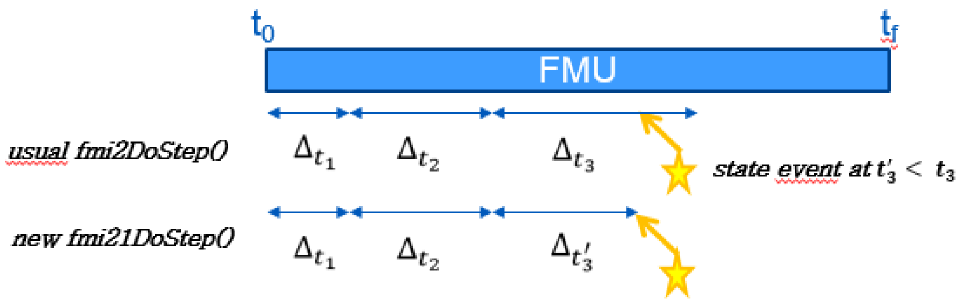


Figure 8: illustration for the new *fmi21DoStep()*

This new primitive *fmi21DoStep()* could be tested on the use case described further.

4.2 A new function *fmi21GetNextEventTime()*

The goal here is to report in advance the *time event* precise date (predictable break-points).

Here are some details of the proposal :

- Prototyping: *fmi21GetNextEventTime(currentTime, stopTime, eventInstant);*
- The Master Algorithm can know in advance the maximum value of the next step size which is wedged on the exact date of the next predictable event.

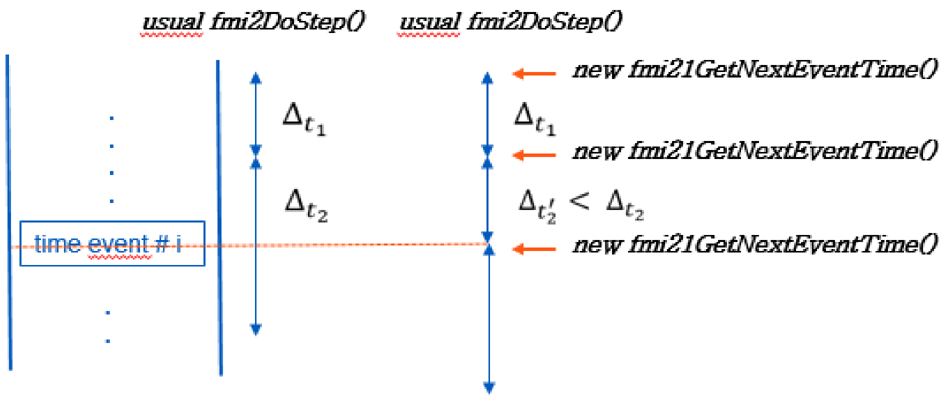

usual fmi2DoStep()
usual fmi2DoStep()
new fmi21GetNextEventTime()
Δ_{t_1}
Δ_{t_1}
new fmi21GetNextEventTime()
Δ_{t_2}
$\Delta_{t'_2} < \Delta_{t_2}$
time event # i
new fmi21GetNextEventTime()

Figure 9: illustration for the new *fmi21GetNextEventTime()*

This new primitive *fmi21GetNextEventTime()* could be tested on the use case described further.

4.3 Several new functions *fmi21GetXXXEvent()*

The goal here is to get the value of variables according to their type XXX at a discontinuity point.

Here are some details of the proposal :

- Prototyping of f*mi21GetXXXEvent()* is in conformance with the proposal recently done by the UC Berkeley (« FIDE – An FMI Integrated Development Environment », SAC'16, April 2016) with a specific *fmi21SignalStatus* typed as an enumeration value 'present' or 'absent';
- But semantics is slightly different: each time *fmi21GetXXXEvent()* is called at $^{-}t_i$, time moves till t_i^+ and an updated value is available for all the referenced variables, whatever their variability.

These new primitives *fmi21GetXXXEvent()* could be tested on the use case described further.

5 Some basic examples to illustrate our proposals

This paragraph give some examples to illustrate the use of the new primitives presented before.

5.1 Example 1: continuous & piecewise differentiable signals

Fig. 10 gives the Modelica model and the desired output waveform for this example.

```
model M1
  parameter Real g = 9.18;
  parameter Real c = 0.9;
  Real y(start = 12), dery(start = 0);
equation
  der(y) = dery;
  der(dery) = -g;
  when y <= 0.0 then
    reinit(dery, -c * dery);
  end when;
end M1;
```

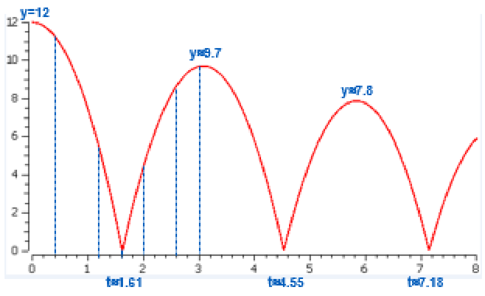


Figure 10: the Modelica model and the desired output waveform for continuous & piecewise differentiable signals

The following array shows a theoretical chronological sequence at the Master Algorithm side:

time	fmi21 DoStep() call	fmi21 DoStep() return	fmi2 GetReal() return
0.0			12.0
0.0	0.4		
0.4		0.4	
0.4			≈ 11.3
0.4	0.8		
1.2		1.2	
1.2			≈ 5.5
1.2	1.5		
≈ 1.61		≈ 1.61	
≈ 1.61			0.0
≈ 1.61	0.4		
≈ 2.01		≈ 2.01	
≈ 2.01			≈ 4.5
≈ 2.01	0.6		
≈ 2.61		≈ 2.61	
≈ 2.61			≈ 8.2
≈ 2.61	0.4		
≈ 3.01		≈ 3.01	
≈ 3.01			≈ 9.7
≈ 3.01	0.2	≈ 3.21	
≈ 3.01			≈ 9.5

5.2 Example 2: piecewise constant signals

Fig. 11 gives the Modelica model and the desired output waveform for this example.

```
model M2
  discrete output Real y(start = 0.0)
equation
  when time >= 0.5 then
    y = 7.0;
  elsewhen time >= 3.5 then
    y = 2.0;
  elsewhen time >= 5.5 then
    y = -2.0;
  end when;
end M2;
```

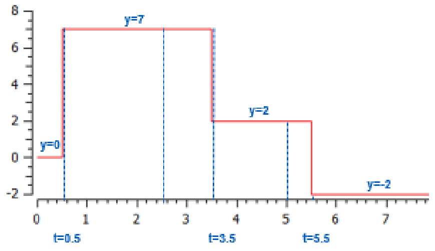


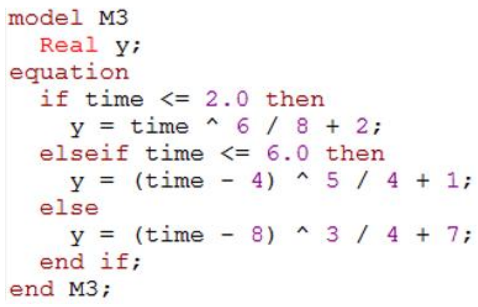
Figure 11: the Modelica model and the desired output waveform for piecewise constant signals

The following array shows a theoretical chronological sequence at the Master Algorithm side:

super dense time	**fmi21 GetNextEventTime() return**	**fmi2 DoStep() return**	**fmi2 GetReal() return**	**fmi21 GetRealEvent() return**
$^{-}$0.0			0.0	
$^{-}$0.0	0.5			
$^{-}$0.5		0.5		
$^{-}$0.5			0.0	
0.5^{+}				7.0
0.5^{+}	3.5			
$^{-}$2.5		2.0		
$^{-}$2.5			7.0	
$^{-}$3.5		1.0		
$^{-}$3.5			7.0	
3.5^{+}				2.0
3.5^{+}	5.5			
$^{-}$5.0		1.5		
$^{-}$5.0			2.0	
$^{-}$5.5		0.5		
$^{-}$5.5			2.0	
5.5^{+}				-2.0
5.5^{+}	t_{max}			

5.3 Example 3: piecewise continuous & differentiable signals

Fig. 12 gives the Modelica model and the desired output waveform for this example.



```
model M3
  Real y;
equation
  if time <= 2.0 then
    y = time ^ 6 / 8 + 2;
  elseif time <= 6.0 then
    y = (time - 4) ^ 5 / 4 + 1;
  else
    y = (time - 8) ^ 3 / 4 + 7;
  end if;
end M3;
```

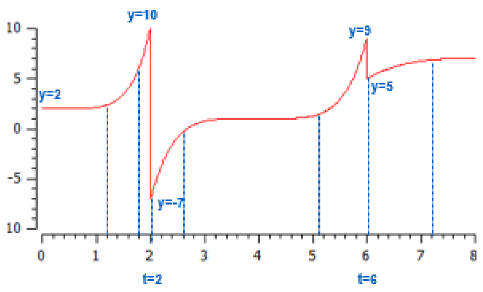


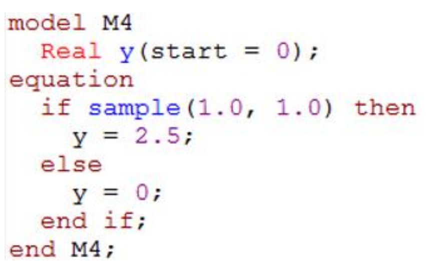
Figure 12: the Modelica model and the desired output waveform
for piecewise continuous & differentiable signals

The following array shows a theoretical chronological sequence at the Master Algorithm side:

super dense time	fmi21 DoStep() call	fmi21 DoStep() return	fmi2 GetReal() return	fmi21 GetRealEvent() return
$^{-}0.0$			2.0	
$^{-}0.0$	1.2			
$^{-}1.2$		1.2		
$^{-}1.2$			≈ 2.1	
$^{-}1.2$	0.6			
$^{-}1.8$		1.8		
$^{-}1.8$			≈ 5.9	
$^{-}1.8$	2.2			
$^{-}2.0$		2.0		
$^{-}2.0$			10.0	
2.0^{+}				-7.0
2.0^{+}	0.6			
$^{-}2.6$		2.6		
$^{-}2.6$			≈ -0.5	
$^{-}2.6$	2.5			
$^{-}5.1$		5.1		
$^{-}5.1$			≈ 1.0	
$^{-}5.1$	1.0			
$^{-}6.0$		6.0		
$^{-}6.0$			9.0	
6.0^{+}				5
6.0^{+}	1.2			
$^{-}7.2$		7.2		
$^{-}7.2$			≈ 6.4	

5.4 Example 4: discrete event signals

Fig. 13 gives the Modelica model and the desired output waveform for this example.



```
model M4
  Real y(start = 0);
equation
  if sample(1.0, 1.0) then
    y = 2.5;
  else
    y = 0;
  end if;
end M4;
```

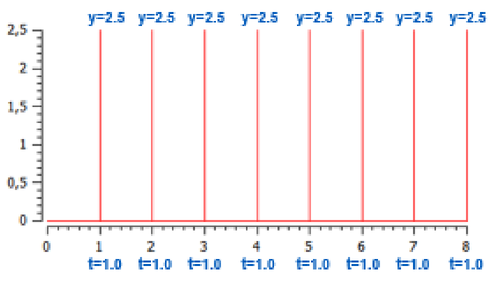


Figure 13: the Modelica model and the desired output waveform for discrete event signals

The following array shows a theoretical chronological sequence at the Master Algorithm side:

super dense time	fmi21 GetNextEventTime() return	fmi2 DoStep() return	fmi21 GetRealEvent() return
${}^{-}0.0$	1.0		
${}^{-}1.0$		1.0	
1.0^{+}			2.5
1.0^{+}	2.0		
${}^{-}2.0$		1.0	
2.0^{+}			2.5
2.0^{+}	3.0		
${}^{-}3.0$		1.0	
3.0^{+}			2.5
3.0^{+}	4.0		
${}^{-}4.0$		1.0	
4.0^{+}			2.5
4.0^{+}	5.0		
${}^{-}5.0$		1.0	
5.0^{+}			2.5
5.0^{+}	6.0		
${}^{-}6.0$		1.0	
6.0^{+}			2.5
6.0^{+}	7.0		
${}^{-}7.0$		1.0	
7.0^{+}			2.5
7.0^{+}	8.0		

6 Future application with an academic use case

The academic use case we intend to use in order to implement and test the new primitives is described in paper [2]. This case mixes *continuous time, piecewise constant* and potentially *discrete event.*

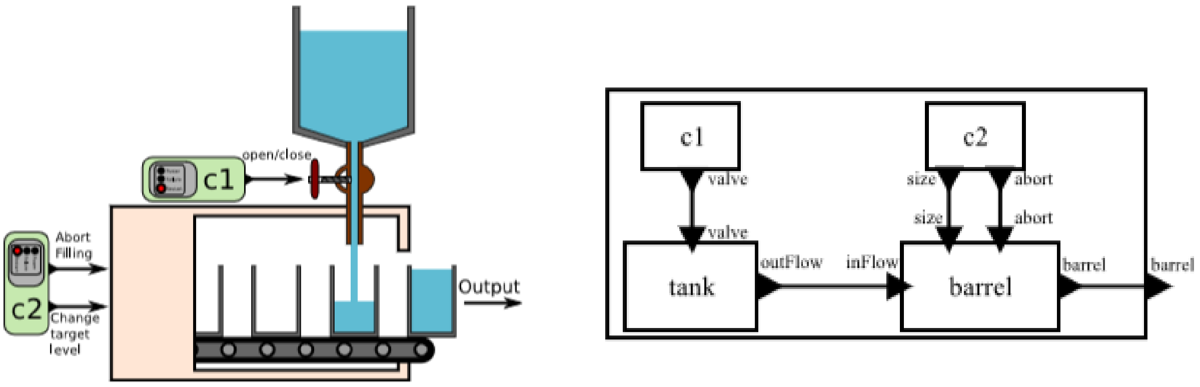


Figure 14: an overview of the use case mixing discrete and continuous components

For this case, a single reference model is available in Modelica.

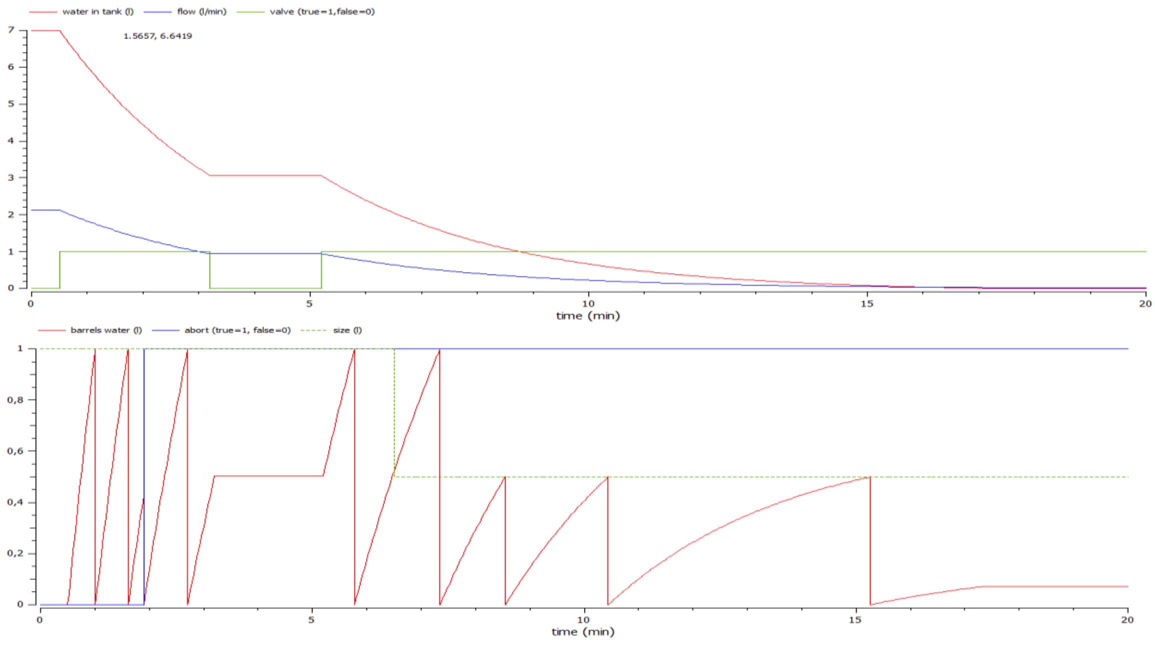


Figure 15: an overview of the use case mixing discrete and continuous components (OpenModelica)

FMUs exported from Dymola are available for components c1, c2, tank and barrel. Discrete models c1 and c2 are also available in IEC 61131-3 (with Control Build) and in UML/SysML (with Papyrus/Moka). Discrete FMUs are (with Control Build) or will be in June (with Papyrus) available.

DACCOSIM can highlight co-simulation accuracy insufficiencies with the FMI-CS 2.0.

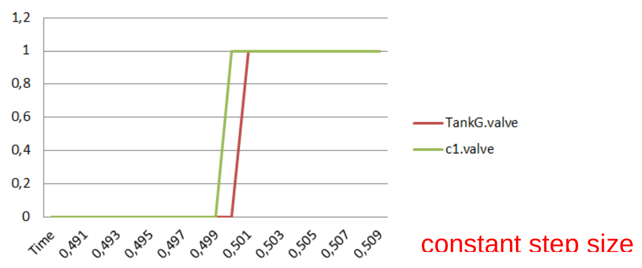


Figure 16: an example of inaccuracy for a discrete signal

7 Conclusion and future work

The partners (EDF, CentraleSupélec and CEA List) intend now to:

- Either
    - Implement all the new primitives in FMUs exported from their tools (e.g. Papyrus)
    - And slightly modify the Master Algorithm in DACCOSIM
- Or
    - Only implement *fmi21DoStep()* in FMUs exported from their tools (e.g. Papyrus)
    - And implement all the other primitives in DACCOSIM at the Master Algorithm side;
- Check for accuracy improvements with DACCOSIM;
- Possibly write a common paper e.g. for the next « Modelica and FMI conference » early 2017.

For pure discrete FMUs, a simplified implementation of the new *fmi21GetNextEventTime()* to explore the future could be something like:

```
proc fmi21GetNextEventTime($t_{cur}$, $t_{max}$, eventInstant)
fmi2GetState(S)                                  // internal FMU state copy
fmi21DoStep(stepSize = $t_{max}$-$t_{cur}$, nextEventInstant)   // instantaneous calculation with discrete FMUs
if cr == fmi21Event then
eventInstant := nextEventInstant                 // event date to return to the Master Algorithm
else                                             // when no event will occur in the next future
eventInstant := $t_{max}^{+}$                    // no event date to return to the Master Algorithm
fmi2SetState(S)                                  // rollback
endproc
```

In this case:

- *fmi21GetNextEventTime()* could be implemented at the DACCOSIM side only or in discrete FMUs;
- *fmi21DoStep()* should be implemented in all FMUs and rollback should be implemented in FMUs.

For pure discrete FMUs, a simplified implementation of the new *fmi21GetXXXEvent()* to get the value of variables at discontinuity points could be something like:

```
proc fmi21GetXXXEvent(varlist, valuelist)
fmi2GetState(S)                    // internal FMU state copy
fmi2DoStep(stepSize = Ɛ)           // microstep (instantaneous calculation with discrete FMUs)
fmi2GetXXX(varlist, valuelist)     // event updates to return
fmi2SetState(S)                    // rollback
finproc
```

In this case:

- f*mi21GetXXXEvent()* could be implemented at the DACCOSIM side only or in discrete FMUs;
- Rollback should be implemented in FMUs.

To summarize, this paper potentially proposes six new primitives:

- *fmi21DoStep()*
- *fmi21GetNextEventTime()*
  - Similar to *fmiGetMaxStepSize()* proposed in paper [4]
- *fmi21GetRealEvent(), fmi21GetIntegerEvent(), fmi21GetBooleanEvent() and fmi21GetStringEvent()*
  - Similar to a proposal done by UC Berkeley in paper [3] but avoiding to implement a *fmi2DoStep()* with a null step size

A new variability *variability = event* is also used as suggested in paper [4] but proposed here in a wider context.

Different possible implementations are suggested in DACCOSIM 2017 and in FMUs from partners.

References

[1] V. Galtier, S. Vialle, C. Dad, J.-Ph. Tavella, J.-Ph. Lam-Yee-Mui, and G. Plessis. *FMI-Based Distributed Multi-Simulation with DACCOSIM. In Proceedings of the 2015 Spring Simulation Multiconference (TMS/DEVS'15)*, 2015.

[2] B. Camus, V. Galtier, M. Caujolle. *Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO (TMS/DEVS'16)*, 2016.

[3] F. Crémona, M. Lohstroh, S. Tripakis, C. Brooks, E. A. Lee. *FIDE – An FMI Integrated Development Environment (SAC'16)*, 2016.

[4] Haifa Research Lab. *FMI working group, Industrial Cyber-Physical Systems consortium (IBM, UTC, CalTech, UC Berkeley)*, 2015.