Toward an Accurate and Fast Hybrid Multi-Simulation with the FMI-CS Standard

Jean-Philippe Tavella[*], Mathieu Caujolle[*], Stephane Vialle[‡], Cherifa Dad[‡], Charles Tan[*], Gilles Plessis[†], Mathieu Schumann[†], Arnaud Cuccuru[§], Sebastien Revol[§]

[*]EDF Lab Saclay, 91120 Palaiseau, France
[†]EDF Lab Les Renardières, 77250 Ecuelles, France
[‡]UMI 2958 GeorgiaTech-CNRS, CentraleSupelec, University Paris-Saclay, 57070 Metz, France
[§]CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems, P.C. 174, Gif-sur-Yvette, 91191, France

Abstract—**Multi-simulation in the context of future smart electrical grids consists in associating components modeling different physical domains, but also their local or global control. Our DACCOSIM multi-simulation environment is based on the version 2.0 of the FMI-CS (Functional Mock-up Interface for Co-Simulation) standard maintained by the Modelica Association. It has been specifically designed to run large-scale and complex systems on a single PC or a cluster of multicore nodes. But it is quite challenging to accurately simulate FMUs-composed systems involving predictable and unpredictable events while preserving the system overall performance. This paper presents some additions to the FMI-CS standard aiming to improve the accuracy and the performance of distributed multi-simulations involving a mix of both time steps and various kinds of events. The proposed FMI-CS primitives are explained, as well as the Master Algorithm strategies to exploit them efficiently.**

I. Motivations and related works

A smart power grid is an example of complex *Cyber-Physical Systems*. For EDF (the major French utility company), it 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. We consider vital to develop agile modeling and simulation in order to design and validate novel distributed operating functions a long time before performing tests on experimental sites, and to reuse models and tools often limited to specific areas of expertise.

Heterogeneity of smart power grids led EDF to investigate coupling standards such as the *Functional Mock-up Interface* (FMI) initiated by Daimler AG within the ITEA2 MODELISAR project and now maintained by the Modelica Association[1]. More precisely, EDF chose the FMI-CS (FMI for Co-Simulation) 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 with the additional benefit of protecting the model intellectual property (IP). Simulation of a smart power grid requires to associate event-based and time stepped components to achieve *hybrid multi-simulations*, but managing events with the current FMI-CS standard is not straightforward.

EPOCHS [1] was an example of distributed multi-simulation environment based on the well known HLA middleware standard[2]. It mixed events and time steps, but delayed event processing at the end of each computation time step (in simulated time) and lacked accuracy. At the opposite, GECO [2] accurately processed all events in a global event queue without introducing delay in the simulated time. But it did not support distribution on several PCs, and so can not scale largely. In 2013, some researchers started to mix distributed HLA standard and FMI 1.x standards [3] but could not manage events before the end of the time steps. In 2014 the C2WT environment mixed again HLA and FMI standards [4], and interconnected the telecom network simulator OMNET++ with FMUs modeling continuous time systems, but it used only constant time steps and its execution algorithm did not address time step interruption for unexpected event processing.

In 2016, some members of our consortium proposed a generic computation strategy and algorithm [5] to achieve accurate event location while respecting the FMI-CS 2.0 standard. But despite optimization, the current FMI-CS features can lead to numerous rollbacks and reruns of FMU computations to process unexpected events. So, such a solution could limit the performance and scalability achieved on multi-core PC clusters by our pure FMI-based distributed multi-simulations [6]. Moreover, discrete signals often used by cyber-physical components in electrical grids cannot be well addressed by this strategy.

This paper proposes new FMI-CS functionalities, in order to improve both the accuracy of cyber-physical co-simulations involving a mix of time steps and various kinds of event signals, and their performance and scalability on distributed architectures.

II. FMI-CS standard and DACCOSIM environment

A. Functional Mockup Interface features and limits

In FMI-CS a component is a self-contained object that includes the model description and a numerical solver provided

[1]https://www.fmi-standard.org/downloads

[2]http://standards.ieee.org/findstds/standard/1516-2000.html

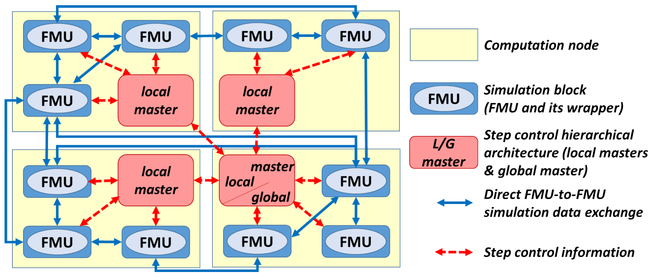
FMU
local master
master local global
Computation node
Simulation block (FMU and its wrapper)
L/G master
Step control hierarchical architecture (local masters & global master)
Direct FMU-to-FMU simulation data exchange
Step control information

Fig. 1. DACCOSIM distributed architecture

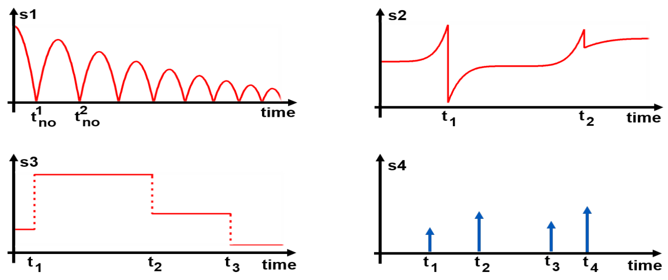

Fig. 2. Different kinds of almost everywhere differentiable signals

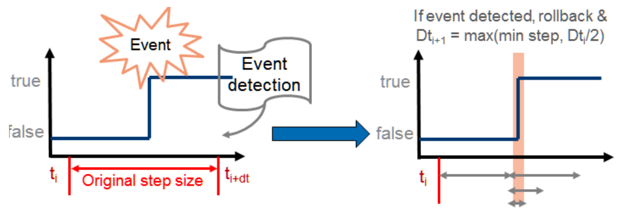
Event
Event detection
true
false
t_i
Original step size
t_{i+dt}
If event detected, rollback & Dt_{i+1} = max(min step, Dt_i/2)

Fig. 3. Approximate event detection with FMI-CS 2.0 (bisectional algorithm)

by the design environment where it was conceived. Co-simulation can encompass a large number of active components (FMUs) potentially developed from different tools. The data exchange between these components is controlled by a program called Master Algorithm (MA) whose behavior is not specified by the FMI standard. The MA controls the time progress of a system co-simulation with a sequence of step by step calculations. During a computation step, each FMU independently simulates a part of the system and, at the end of the step, the outputs from some FMUs provide new input values to some other FMUs.

However, our practice shows that all kind of signals are not correctly handled by the current version 2.0 of the FMI-CS standard, especially in an hybrid co-simulation context mixing discrete and continuous components (see section III-B).

B. *DACCOSIM, an FMI-based co-simulation software*

DACCOSIM has been designed to achieve multi-simulations of continuous time systems discretized with time steps, and run solvers using constant or variable time steps, the latter maintaining the requested accuracy with the minimal amount of computations whatever the system dynamics. DACCOSIM *parallel and distributed runtime architecture* is performance- and scalability-oriented, and takes maximal advantage of any cluster of multi-core nodes. Each simulation executes a series of time steps composed of three stages: the time step computations of the FMUs (independent computations), the communication of the computed FMUs outputs values to the connected inputs (many small communications), and the simulation control by the hierarchical control masters (information gathering, next operations decision and order broadcasting). DACCOSIM architecture distributes FMUs on cluster nodes, encapsulates each FMU in a multithreaded wrapper, and implements a *hierarchical (and distributed) control master* to manage all threads and FMU operations (Fig. 1). A unique global master controls local ones, and each local master controls a set of FMUs on its node. All master tasks run concurrently. Each multithreaded *wrapper* runs the computation function of its FMU to achieve time step progress, manages inter-FMUs and FMU-Master communications, and stores simulation results on disk. It parallelizes and optimizes the orchestration of these tasks to minimize the global execution time.

The Java version of DACCOSIM relies on JavaFMI[3] developed by SIANI institute (Las Palmas University), and is available for both Windows and Linux operating systems, either 32-bit or 64-bit.

III. FMI EVOLUTION TO CONSIDER VARIOUS SIGNALS

A. *Almost everywhere differentiable signals*

Fig. 2 illustrates the different kinds of almost everywhere differentiable signals we need to consider in our cyber-physical systems. Signal $s1$ is a *continuous & piecewise differentiable signal*. It is present at each time $t_i \in \mathbb{R}^+$, continuous on $\mathbb{R}^+$, but not differentiable at all the points t^j_{no} having a null ordinate. Signal $s2$ is a *piecewise continuous & differentiable signal*. It is present at each time $t_i \in \mathbb{R}^+$, but is neither continuous nor differentiable at time $t1$ and $t2$. Signal $s3$ is a *piecewise constant signal*. It is present at each time $t_i \in \mathbb{R}^+$, and is constant on disjoint and continuous time slots I^j so that $\bigcup_j I^j = \mathbb{R}^+$, with a discontinuity appearing at each time slot switch t_j. Signal $s4$ is a *discrete event signal*. It is present for a set of definition $\mathfrak{D}$, being a discrete time set $t_i \in \mathfrak{D}$, with $\mathfrak{D} \subset \mathbb{R}^+$. This signal can be confused with the events it generates.

Later in this article, we only focus on signals that are always present all the time of the simulation (signal $s1$, $s2$ and $s3$ on figure 2) as these cases correspond to the ones encountered in cyber-physical systems like smart grids, smart buildings and smart factories.

B. *Proposal of FMI-CS standard evolution*

The FMI-CS standard defines an application programming interface (API) that components conforming to the standard must implement. The environment interacts with the FMU only by means of its API, which consists primarily of the

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

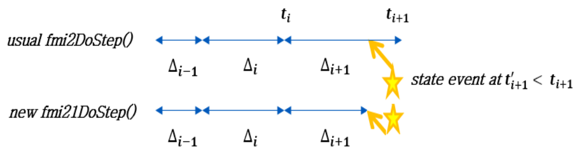

Fig. 4. Behavior of the new primitive *fmi21DoStep(...)*

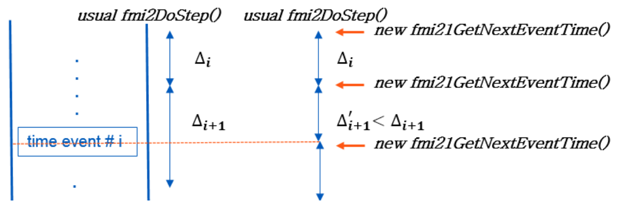
usual fmi2DoStep()
usual fmi2DoStep()
new fmi21GetNextEventTime()
new fmi21GetNextEventTime()
new fmi21GetNextEventTime()
Δ_i
Δ_{i+1}
$\Delta'_{i+1} < \Delta_{i+1}$
time event # i

Fig. 5. Getting next known event time with *fmi21GetNextEventTime(...)*

fmi2DoStep() primitive which attempts to perform a simulation step on the FMU, given a specific time step. The problem with the current version of the FMI-CS standard is that event handling is not very accurate: when an event occurs during a step calculation, the event is only reported at the end of the current step. If the simulation step is too large, events could even be missed. Different strategies can be considered to approximately locate an event (progress with minimum step size [7], bisectional method [5]). All of them require rollbacks and decrease the overall computational performance (Fig. 3).

Specifically, events can be either *state events* or *time events*. It is assumed here that *state events* are *unpredictable break-points* (they cannot be predicted without exploring the future) while *time events* are *predictable break-points*.

For unpredictable break-points, an evolution of the *fmi2DoStep(...)* is proposed in the form of a new primitive ***fmi21DoStep(stepSize, nextEventTime)***. It allows the FMU solver to stop at the first unpredictable event with a new return code *fmi21Event* and the time event is given by the returned value *nextEventTime* (Fig. 4). If no event occurs, *nextEventTime* is the computation step end time. So, the MA can go on computing state variables without performing any rollback on this component.

But when one FMU prematurely stops, the other FMUs involved in the co-simulation will proceed with their calculation causing a computation time waste. The ***fmi21BreakPendingStep(earlierTime, stopTime)*** primitive we propose is meant to reduce this computational burden by interrupting an FMU pending calculation before its normal end. A pending step execution will be stopped by an asynchronous call of the primitive at *earlierTime* if it is not yet reached, or upon receipt in the opposite case returning the actual stop time value *stopTime*. It will also ignore any *earlierTime* earlier than its current start time.

For predictable break-points, a primitive named ***fmi21GetNextEventTime(currentTime, stopTime, eventTime)*** is proposed to report in advance the time event precise date (Fig. 5). This time information could be exploited to optimize the value of the next step size in view of the exact date of the next predictable event.

To be complete, we propose ***four*** additional primitives ***fmi21GetXXX()*** to get the value of variables according to their type *XXX* (*Boolean*, *Integer*, *Real* or *String*) at a discontinuity point. Prototyping of these primitives makes sense in particular in the context of super-dense time system [8], but not only. After a *fmi21DoStep()*, the FMU stops at t_i^+, allowing the new *fmi21GetXXX()* to get the values of variables (at t_i^+) after all the events occurring at the same Newtonian time t, exactly as the current *fmi2GetXXX()* do. But in addition and without changing the reached Newtonian time all previous signal values are also returned by the new primitive whatever the events that happened between t_i^- and t_i^+. This mechanism respects the constraint of a positive non zero step size for step computation as required by the current FMI standard but contradicts a recent proposal from UC Berkeley [9].

All these new primitives have been introduced by EDF in June 2016 in the Working Group *Clock and Hybrid Co-simulation* of the FMI project, now hosted by the Modelica Association[4], and discussions are ongoing.

IV. Expected impact on multi-simulation

A. Illustrative use-case presentation

The academic use case we intend to use in order to implement and test the proposed FMI primitives is a barrel-filler factory (Fig. 6). Involved models, detailed in [5], consist of:

- A **queue of barrels** waiting on a conveyor to be filled.
- A **tank** storing the water to fill the barrels.
- A **controller** $c1$ managing the opening of the valve between the tank and the barrel.
- A **controller** $c2$ regulating the whole filling process.

It mixes continuous dynamics (water flow from the tank) and several types of non-continuous signals discussed in III-A, such as piecewise continuous and differentiable signals (water level in the tanks), piecewise constant and potentially discrete events (command signals), see Fig. 7.

B. Expected impact on result accuracy

Results showed in Fig. 8 come from a DACCOSIM multi-simulation. They highlight the current accuracy insufficiencies of FMI-CS 2.0. Even when applying adaptive time step strategies relying on rollbacks to ensure accurate simulation and correct detection of events, the location of events can only be approximated. Accuracy is limited by the minimum time step size authorized by the masters. But, when using the proposed *fmi21DoStep()*, the accuracy of the event time will no longer be limited by the master algorithm control strategies, it will only depend on the ability of the FMU internal solver to manage these events.

The proposed *fmi21GetXXXEvent()* primitives complement the *fmi21DoStep()* function by allowing to fetch the FMU output values computed by the internal solver just before and after a time event. Exact characterization of the FMU dynamics

[4]https://www.modelica.org/

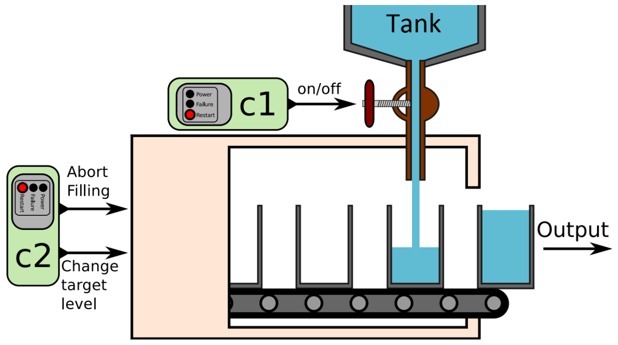

Fig. 6. Use case mixing discrete and continuous components

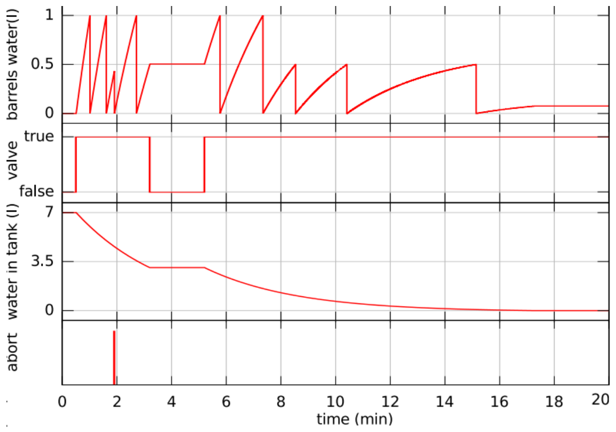

Fig. 7. Example of signals considered in the use case

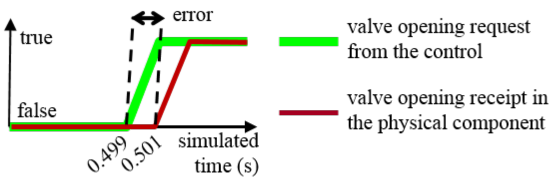

Fig. 8. DACCOSIM error on the gate output command (constant time step)

before and after an event can thus be achieved without having to approximate them by performing small step computations. Signals such as the barrel water volume or the abort signal (Fig. 7-a and -d) will thus be accurately characterized.

Thanks to the proposed add-ons to the FMI-CS standard, similar accuracy performance should be achieved within a domain-specific simulation tool and within a FMI 2.0 based generic multi-simulation tool such as DACCOSIM.

C. Expected impact on performances and parallelization

One of the major challenges when managing events with the FMI standard in hybrid multi-simulations is avoiding CPU time waste. The new FMI functions we propose (see section III-B) will help to avoid some rollbacks and re-computations:

- With function ***fmi21GetNextEventTime()*** a control FMU will be able to communicate the date of its next known *time event*. Master algorithms, such as the one implemented in DACCOSIM, should be able to exploit this information when defining the size of the new computation step: its end will be aligned with the exact event time if initially included in the considered time step. So, at the end of the time step, the event will be routed to other FMUs without any delay nor rollback.
- With function ***fmi21DoStep()***, an FMU will be able to stop its computation just after it has generated a *state event*, and with function ***fmi21BreakPendingStep()*** it will be able to request all other FMUs they stop their computations as soon as they reach this event time. Thus the new DACCOSIM master algorithms will take advantage of this behavior and ask only the fastest FMUs, i.e. those whose computation process had already exceeded the earliest event time before getting the corresponding break signal, to roll back and rerun a computation up to the event time.

Fig. 9 illustrates this last mechanism, considering 3 FMUs running their *doStep()* functions concurrently on different CPU cores. FMU1 generates a state event at the simulated time t'_{i+1} and signals to all others FMU to stop their computation at t'_{i+1} (calling ***fmi21BreakPendingStep()***). But FMU2 achieves very light computations and has already reached the end of the simulated time step (t_{i+1}). Then, FMU2 will roll back to t_i and re-run its computation up to t'_{i+1}. At the opposite, FMU3 is more computation intensive and benefits from the FMU2 break signal to stop its computation when reaching t'_{i+1}, avoiding unnecessary computations (up to t_{i+1}), rollback and re-computation (up to t'_{i+1}). Finally, complete execution time on a multi-core processor will be shorter when resizing the ongoing FMU computations. The ***fmi21BreakPendingStep()*** function is also under discussion in the Working Group *Clock and Hybrid Co-simulation* of the FMI project.

Better support of events in FMI-CS will allow to introduce more FMU components devoted to control. These components consume very few CPU time (fast control algorithms) but increase communications (reading many sensor outputs and sending command signals). So, this evolution should impact FMU distribution algorithms on PC clusters, which optimize both load balancing and inter-node communications [6].

V. Future works on Smart Grid simulations

Until now the FMI-CS standard allowed us to compute different kinds of Smart Grid simulation from building thermics [6] to MV network voltage management based on state-estimation. These simple use cases permitted us to identify the limits of the current standard and design new ways to overcome them. In fact, integrating the proposed primitives into the standard will be required in order to enable efficient and accurate large, multi-physics and control-centric simulations.

In a near future, complex Smart Grid use cases involving (i) distribution networks to control Distribution System Operators (DSO) assets with new smart functions to ensure the absence of constraints on the power grid, (ii) buildings and their energy systems with control over elements such as heating, ventilation and air conditioning, and (iii) energy actors to control the flexibility of buildings and electrical vehicles, will be performed (Fig. 10). So, important work is going to be

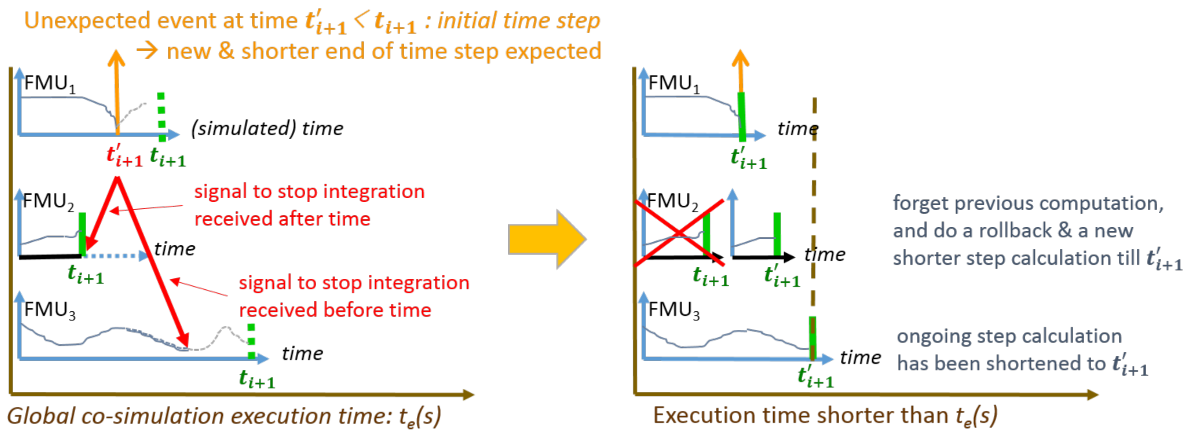
Unexpected event at time $t'_{i+1} < t_{i+1}$: initial time step
→ new & shorter end of time step expected
FMU$_1$
(simulated) time
t'_{i+1} t_{i+1}
FMU$_2$
signal to stop integration received after time
time
t_{i+1}
signal to stop integration received before time
FMU$_3$
time
t_{i+1}
Global co-simulation execution time: $t_e(s)$
FMU$_1$
time
t'_{i+1}
FMU$_2$
time
t_{i+1} t'_{i+1}
forget previous computation, and do a rollback & a new shorter step calculation till t'_{i+1}
FMU$_3$
time
t'_{i+1}
ongoing step calculation has been shortened to t'_{i+1}
Execution time shorter than $t_e(s)$

Fig. 9. Parallel execution of FMUs with different CPU loads, on a multi-core processor

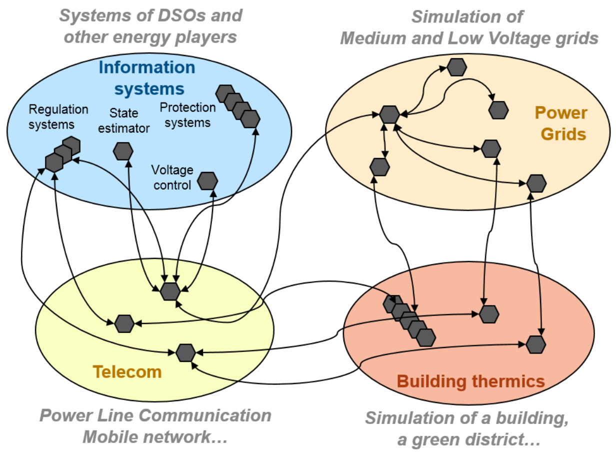
Systems of DSOs and other energy players
Simulation of Medium and Low Voltage grids
Information systems
Regulation systems
State estimator
Protection systems
Voltage control
Power Grids
Telecom
Building thermics
Power Line Communication Mobile network…
Simulation of a building, a green district…

Fig. 10. Illustration of model interactions of Smart Grid use cases

carried on several topics, among which: implementation of the proposed FMI-CS add-ons in control FMUs generated by Papyrus (https://eclipse.org/papyrus/), upgrade of the JavaFMI API[3] to support these evolutions, adaptation of the control logic of DACCOSIM Master Algorithms to exploit them, and design of new cluster deployment strategies.

A preliminary version of this work has been presented in May 2016 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, FMI and BIM standards, contributing to smart power grid design. Then, a more complete version has been introduced in June 2016 in the Working Group *Clock and Hybrid Co-simulation* of the FMI project, and discussions are ongoing to attempt to improve the FMI-CS standard.

Acknowledgment

Authors thank Region Grand Est and RISEGrid institute for their support to this research.

References

[1] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, "EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Transactions on Power Systems*, vol. 21, no. 2, May 2006.

[2] H. Lin, S. S. Veda, S. K. Shukla, L. Mili, and J. S. Thorp, "GECO: Global Event-Driven Co-Simulation Framework for Interconnected Power System and Communication Network," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1444–1456, 2012.

[3] M. U. Awais, P. Palensky, A. Elsheikh, E. Widi, and S. Matthias, "The high level architecture RTI as a master to the functional mock-up interface components," in *International Conference on Computing, Networking and Communications (ICNC)*, San Diego, USA, Jan. 2013.

[4] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, and C. Sureshkumar, "Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems," in *10th International Modelica Conference*. Modelica Association and Linkoping University Electronic Press, 2014.

[5] B. Camus, V. Galtier, and M. Caujolle, "Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO," in *Proceedings of the 2016 Spring Simulation Multiconference (TMS/DEVS'15)*, 2016.

[6] C. Dad, S. Vialle, M. Caujolle, J-Ph. Tavella, and M. Ianotto, "Scaling of Distributed Multi-Simulations on Multi-Core Clusters," in *Proceedings of 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2016)*, Paris, June 13-15 2016.

[7] 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.

[8] X. Liu, E. Matsikoudis, and E. A. Lee, "Modeling timed concurrent systems," in *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR)*, ser. LNCS, C. Baier and H. Hermanns, Eds., vol. 4137. Springer, 2006.

[9] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, "FIDE - An FMI Integrated Development Environment," in *Symposium on Applied Computing*, April 2016.