Synthesis and feedback on the distribution and parallelization of FMI-CS-based co-simulations with the DACCOSIM platform

Cherifa Dad, Jean-Philippe Tavella, Stéphane Vialle

Version of Record: https://www.sciencedirect.com/science/article/pii/S0167819121000545
Manuscript_0beffd2694e1f5a9488a58aea08b6889

Synthesis and feedback on the distribution and parallelization of FMI-CS-based co-simulations with the DACCOSIM platform

Cherifa DAD[a,b], Jean-Philippe TAVELLA[c], Stéphane VIALLE[d,*]

[a] *CentraleSupélec & UMI GT-CNRS, Metz, France*
[b] *University of Lorraine & LORIA, Nancy, France*
[c] *EDF Lab Paris-Saclay, Paris, France*
[d] *CentraleSupélec, University Paris-Saclay & LRI, Metz, France*

Abstract

A co-simulation applied to Smart Grids consists of grouping in the same setting models of physical components (among other electrical ones) and models of control units (including communication devices). Combining these models needs to use a generic and robust co-simulation environment instead of developing a specific one. In this context, we developed the DACCOSIM 2017 co-simulation platform based on FMI-CS (Functional Mock-up Interface for CoSimulation) standard to simulate the physical components of a Smart Grid. These components represent the most CPU-consuming part of the co-simulation. However, the tasks of FMI-CS-based applications (FMUs) are exposed as heterogeneous gray boxes with no information concerning their computation and communication volumes. Moreover, all these FMUs frequently communicate with each other by sending a lot of small messages. Consequently, the deployment of an FMI-CS based co-simulation on a distributed architecture is a complex task carried out by DACCOSIM 2017. This paper introduces the development of DACCOSIM-2017, and its experiment on distributed architectures.

Keywords: Distributed Co-Simulation, Multicores, Clusters, Performances, Scaling, FMI standard.

1. Introduction

The emergence of Smart Grids [1] offers new opportunities to design today's and tomorrow's energy systems, from producers to consumers through energy networks (electricity, heat, ...) and telecommunications, to optimize production, distribution, and consumption of electricity. This kind of system represents a concrete example of cyber-physical systems, and is an industrial challenge for EDF[1] and its subsidiary ENEDIS[2]. The transformations of the energy world and the associated uncertainties require efficient prediction tools to understand and address the impact of the various energy scenarios on EDF's investments and thus best organize the modernization of the energy system in order to: (1) guarantee efficient and high-quality low-carbon energy production (i.e. resilient to hazards) and (2) develop innovative services based on new uses (e.g. self-consumption, electric vehicles...). From a system point of view, a Smart Grid contains many heterogeneous cyber and physical components that interact with each other: smart buildings (residential, tertiary and industrial), district heating network, distribution electrical grid, local production (combined heat and power plant, gas boilers, ...), telecommunication infrastructure, and smart operation control units ensuring buildings self-consumption through photo-voltaic panels on the roofs, incentive cut offs, storage devices (thermal, batteries). But, their implementation on the ground is difficult and the investment costs remain expensive. That's why we must first validate their proper functioning by co-simulation.

A co-simulation [2] is a well known technique for simulating and validating complex systems. It facilitates the work to associate different weakly coupled models designed from heterogeneous development tools, in favor of evaluating their relevance and the global system performance before any implementation in the field. To co-simulate a Smart Grid, the electrical utility company we worked with had adopted the Functional Mock-up Interface (FMI) standard [3] to normalize the exchanges of data between all the components. The FMI project was initiated by the ITEA2 MODELISAR project (2008-2011) and is now kept up to date by the Modelica Association[3]. FMI is based on a continuous model for simulating the whole tasks of complex systems, e.g. Smart Grids. The adopted standard makes possible to combine different models by encapsulating them into gray boxes appointed FMUs (Functional Mock-up Units) with two modes: *FMU for Model Exchange* mode (FMU-ME) for strong coupling of FMUs and *FMU for co-simulation* mode (FMU-CS) for weaker coupling. In the FMU-ME mode,

⋆This article presents the results and lessons learned from a 5-year research project funded by the French institute RISEGrid

*Corresponding author: Stéphane Vialle, CentraleSupélec, 2 rue Edouard Belin, 57070 Metz, France ; phone: +33 6 66 63 32 16 ; fax: +33 3 87 76 47 00

Email addresses: dad.cherifa@gmail.com (Cherifa DAD), jean-philippe.tavella@edf.fr (Jean-Philippe TAVELLA), stephane.vialle@centralesupelec.fr (Stéphane VIALLE)

[1] The major French electrical utility company.

[2] Company in charge of managing the electricity distribution network in France.

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

an FMU simply contains a model and it is up to the user to provide a solver or to develop his own solver, while the FMU-CS mode allows to encapsulate in the same box (FMU) a model with a previously chosen numerical solver. We use the FMU-CS mode because each model becomes an easy-to-use executable while respecting intellectual property.

In general, an FMU is a zipped folder that contains a description XML file (*modelDescription*) for describing external and internal variables of FMUs, and a set of binary codes depending on the operating system offering the FMU calculation functions as well as the primitives of its interface. But, unfortunately, it requires defining and implementing a "Master Algorithm" which (1) interfaces all FMUs of the system, (2) establishes connections and, (3) exchanges data between different interacting models [4]. The simulation of physical components relies on solvers that discretize continuous time in time steps, which are either constant or variable knowing that a time step should be large enough so the co-simulation is not dramatically slowed down, but not too significant to avoid important approximation errors [5]. Note that for EDF, real-time or Hardware In the Loop (HIL) executions are not an objective, the main goal being to be able to run large simulations within a reasonable time.

Project objectives and work presented

The goal of our co-simulation project was, therefore (1) to develop an FMI-based co-simulation platform adapted to Smart Grids which can distribute and simulate on a distributed multi-core PC cluster its different components, (2) to propose efficient approaches for the placement of these tasks to increase performance on multi-core PC clusters, and (3) to achieve co-simulation scale-up.

The work presented in this paper focusses on simulating the most computationally intensive part of the Smart Grid, which is the co-simulation of physical devices of a power system using constant time steps. For that, we used the 2017 version of DACCOSIM (*Distributed Architecture for Controlled COSIMulation*), a platform to design and run simulation applications based on FMI-CS in the form of a graph of heterogeneous FMUs, frequently communicating. When the application has a large number of components, we cannot launch the DACCOSIM graph on a single physical node with a limited memory size and computing power, but rather on a multicore PC cluster offering both a large amount of memory and a set of computing cores.

A DACCOSIM graph can be represented as a *periodic task graph*, repeating simulation time steps. But, inside a time step, it becomes a *direct acyclic graph* (DAG): vertices are both FMUs and master tasks, and edges are both inter-FMU and FMU-hierarchical master communications. Some of these communications also introduce synchronizations between tasks. In this present work, we summarize all the efforts we made for distributing efficiently and rapidly the FMI-CS-based applications on a distributed architecture. We also identify and discuss the various difficulties that we encountered to achieve an accurate and efficient size-up of such applications.

This paper is structured as follows: section 2 summarizes related works about co-simulation platforms applied to cyber-physical systems and section 3 depicts our innovative DACCOSIM platform. In section 4, we evaluate the behavior of FMU graphs on multicore PC clusters. Some methodologies and approaches to distribute the heterogeneous FMUs on multicore PC clusters are introduced in section 5. Section 6 and 7 interpret the scaling and the size-up of small and large industrial applications based on FMI-CS. The main conclusions and discussions of current results are provided in the last section.

Framework	Synchronization	Distributed exec.
EPOCHS [6]	*Time-stepped*	Yes
ADEVS [7]	*Dedicated orchestration*	Limited
GECO [8]	*Global event-driven*	No
INSPIRE [9]	*Time-stepped*	Yes
C2WT [10]	*Time-stepped*	Yes
PowerNet [11]	*Dedicated orchestration*	Limited
VPNET [12]	*Time-stepped*	Limited
DACCOSIM [13]	*Time-stepped*	Yes (large scale)
MECSYCO [14]	*SMA event-driven*	Yes
MOSAIK [15]	*SMA event-driven*	Yes
HELICS [16]	*Event-based*	Yes

Table 1: Comparison between co-simulation solutions.

This research is carried out by the Research Institute for Smarter Electric Grids (RISEGrid)[4], founded by EDF and CentraleSupelec, and ultimately aims to simulate the smart electric grids of the future.

2. Related works and positioning

Co-simulations of Smart Grids use both event-based simulators to simulate the telecom networks and information systems, and discretized continuous-time simulators to simulate the physical components. These different kinds of simulators usually exchange data through a middleware, like an HLA bus [17], or an implementation of an FMI Master Algorithm [4], or a mix of an HLA bus and an FMI Master Algorithm. In any case, communications between components can be done either directly or across a centralized communication mechanism. This section introduces the solutions adopted by the main co-simulation environments applied to Smart Grids.

2.1. Synchronization types for co-simulation

We can identify three main approaches when designing a co-simulation:

- *Time-stepped* approach: Individual simulators of the global system execute independently their simulations during constant or variable time steps, and exchange and synchronize some output values at the end of each time step (which are determined by the Master Algorithm). FMI-based co-simulations follow this approach. However, taking into account some events (like an alarm) appearing during time steps requires to enhance the pure time-stepped model of continuous time based simulators.

[4] https://www.centralesupelec.fr/fr/risegrid-institute-research-institute-smarter-electric-grids

- *Global event-driven* approach: A global ordered list saves both the *state events* emitted by continuous time based simulators of physical systems (ex: alarm signals), and the *time events* emitted by telecom network and information system simulators. Then, an event logical bus drives each event to the relevant components. HLA-based co-simulations follow this approach.

- *Dedicated orchestration* approach: Some co-simulations implement specific orchestration mechanisms, like in [11] where two simulators call each other on-demand and run alternatively, or in [12] where a dedicated orchestrator interconnects and manages only two simulators.

Implementation of the last approach does not require a middleware and does not facilitate the design of generic and scalable co-simulation environments. We have therefore focused our research on the first and second approaches, and the blending of these approaches to include both event-based and time-stepped components. Such a hybrid approach is required for the simulation of *cyber-physical* systems like Smart Grids but it remains complex.

2.2. Co-simulation environments applied to Smart Grids

In 2006, EPOCHS [6] was the first distributed multi-simulation environment applied to the simulation of an electric power and communication system. It was based on the HLA middleware standard and mixed event-based and time-stepped components, but delayed state event processing at the end of each time step and lacked accuracy. Based on Discrete EVent System formalism (DEVS [18]), the ADEVS platform [7] improved the synchronization algorithm of EPOCH. It successfully modelled a power system with DEVS formalism, coupled with the NS2 telecom simulator. However, this platform was designed for a specific application and cannot apply to more generic energy systems. At the opposite, in 2012 GECO [8] was an attempt for a more accurate simulation of an electric power and communication system. It accurately processed all events in a global event queue without introducing a delay in the simulated time. But, it was not designed on top of a generic and distributed middleware (it implemented a dedicated orchestrator), so it did not support distribution on several PCs and could not scale largely. Approximately at the same date (2012-2013), the INSPIRE framework [9] implemented a more detailed and realistic co-simulation of a telecommunication network and an electric power system, to investigate the impact of telecommunication network delays. Interesting results were an encouragement to use co-simulation to design electrical systems driven by telecommunications networks. Moreover, INSPIRE was implemented on top of HLA middleware, but no information was reported about the distribution on several machines and the scalability of the co-simulations.

In parallel, the FMI standard emerged in 2010 [3], and in 2013 researchers of the Austrian Institute of Technology (AIT) have experimented with several strategies to mix FMI 1.x standards and HLA standard. They developed several FMI Master Algorithms to manage and interface a set of FMUs with an HLA event bus (the Run-Time Infrastructure - RTI), which favored the accuracy or the parallelism of the execution of the FMUs, or supported variable stepping [19, 20, 21]. However, they could not manage events before the end of a time step, and no performance measurements of these FMI+HLA solutions were reported. In 2014, the C2WT environment mixed again HLA and FMI standards [10], and interconnected the telecom network simulator OMNET++ with FMUs modeling continuous time systems. But, the distributed control of the FMUs supported only constant time steps and again delayed some event processing at the end of the FMU time steps. C2WT did not address time step interruption for unexpected event processing, which would not be acceptable in our Smart Grid use cases (ex: unexpected alarms must be processed on time).

The development of the Mosaik [5] platform started around 2012 and aimed at designing and simulating complex large smart grid scenarios [15]. It adopted a multi-agent system approach, including several components to assist users in the design and validation of their co-simulation scenarios. It relies on communication mechanisms allowing distributed executions on several remote sites. However, the authors do not emphasize distribution on parallel architectures, nor performance measurements at runtime. More recently, Mosaik has added FMUs as components in its co-simulations [22]. Since 2015 MECSYCO [23] has been exploring another multi-agent approach, including a distributed event bus. It has been enhanced in 2016 with a *model artifact* relying on the DEV & DESS formalism to integrate FMUs and to offer a sound framework for describing hybrid systems [14]. But, the drawback of this multi-agent technology is that each time an event occurs in the system, a synchronization has to be performed between all agents, causing rollbacks to resynchronize them all. Tab. 1 summarizes some features of these environments.

Finally, the HELICS project is participating in the modernization of the US network, by developing a co-simulation framework [16], the first version of which was published in 2017. It aims at co-simulating very large systems (up to 100K components), including off-the-shelf simulators, FMI and HLA components. It relies on an ad hoc communication mechanism or standard middleware (MPI or ZMQ), and includes a hierarchical and distributed broker for co-simulation control. Its time model is based on events and time series, with a co-iteration step before any time advancement to improve signal consistency. However, no information is available on the hardware used for the experiments, nor on possible accelerations.

2.3. Positioning of DACCOSIM

We designed the foundations of our co-simulation framework (DACCOSIM) from 2013 up to 2015, with the objectives to support (1) accurate processing of unexpected state-events and adaptive time steps, both requiring the FMUs to roll back, and (2) massive executions involving many FMUs deployed on large computing clusters, requiring efficient parallel and distributed executions.

[5] https://mosaik.offis.de/

Like AIT researchers, we started experimenting with several FMI+HLA hybrid solutions. However, in order to achieve our objectives we preferred to develop an FMI-based solution relying on a hierarchical and distributed Master Algorithm (see section 3). Our first prototype was delivered in 2015 [13].

At the difference of other research projects, measuring and achieving good performances on realistic co-simulations of energy systems has always been one of our main concerns [24, 25]. So, we designed some strategies and algorithms to distribute and load balance numerous FMUs on large PC clusters, successfully experimented and included in DACCOSIM 2017 (see section 5).

3. Presentation of the DACCOSIM-2017 platform

DACCOSIM [13] is a physical system platform based on the FMI for co-simulation standard (FMI-CS) used in its 2017 version. It has been developed by CentraleSupélec and EDF R&D as part of the RISEGrid institute. DACCOSIM allows running many heterogeneous gray boxes, called FMU-CS, on multicore PC clusters that interact with each other. This platform integrates wrappers that encapsulate FMI API, in order to manipulate these FMUs. So, it uses the JavaFMI[6] package in Windows and Linux versions. Moreover, DACCOSIM 2017 includes a user-friendly Graphical User Interface (GUI) for designing and performing a complex system. Note that DACCOSIM 2017 allows designing the physical system either through a "DSL" (Daccosim Script Language) code or its GUI.

3.1. Global architecture

DACCOSIM 2017 achieves parallel and distributed co-simulations, dedicated to physical systems, with a hierarchical co-simulation master [24]. In our attempts to design, distribute and co-simulate on cluster nodes very wide systems composed of thousands of FMUs, we needed both to synchronize all the communication points (for accuracy purpose) and decentralize the usual control function of the Master Algorithm (for performance purpose). As shown in Fig. 1, the overall architecture of DACCOSIM consists of a set of local masters supervised by a global master. The leaves of this architecture represent blocks of co-simulation, containing a wrapper code connected to the FMUs following the FMI standard. Each local master controls its simulation blocks located in the same compute node. In practice, DACCOSIM generates a set of Java codes corresponding to local master codes located on the different compute nodes, and each of them calls one or more FMUs launched by their wrappers.

According to FMI-CS 2.0 standard, DACCOSIM notably offers for the co-initialization of its calculation graph:

- Automatic construction of the global causal dependency graph, built both from the FMUs internal dependencies and the calculation graph external dependencies. An acyclic view of the graph is generated by aggregating each cycle as a super-node composed of Strongly Connected Components (SCCs);
- The first version of a generalized distributed co-initialization algorithm, mixing a sequential propagation method applied to the acyclic dependency graph, and a Newton-Raphson method solving its SCCs. These co-initialization features of DACCOSIM 2017 are not detailed in this paper focusing on the co-simulation mechanisms.

And for co-simulation, DACCOSIM 2017 offers among others:

- Implementation of each FMU wrapper as two threads allowing to concurrently run computations and send messages (FMU & control) while receiving incoming messages;
- Overlapped or ordered data synchronization inside distributed masters, that can operate with constant or variable time steps.
- Approximate event detection while waiting for a new version of the FMI standard able to correctly handle hybrid co-simulations [26].

An original feature of the DACCOSIM architecture lies in the fact that the FMU variable values to be exchanged at each communication step are directly transmitted from the senders to receivers without passing by a master. The masters, the wrapped blocks (mainly FMUs with wrappers) as well as the communication channels between them are automatically generated by DACCOSIM by translating the calculation graph defined by the user via its GUI. All communications are implemented using ZeroMQ[7] middleware, allowing direct communications between different threads located on the same or on different PC cluster nodes. For intra-node communications, a mechanism of shared message queue is also available.

3.2. Execution principles

With DACCOSIM, all FMUs execute the same constant or variable time step by applying one of two orchestration modes [24]: *Ordered* or *Overlapped* mode, as shown in Fig. 2 and 3 (will be detailed further). In the first mode, all FMUs of the system start their execution in parallel, and each of them executes successively the three substeps of the time step:

1. FMU *computation* substep,
2. *control* substep including two synchronization barriers between all tasks,
3. FMU *data-exchange* substep.

This last substep is chained with the first substep of the next time step according to a *relaxed synchronization*. In contrast, the *Overlapped* mode is an optimization of the first mode that

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

[7] ZeroMQ, 0MQ or ZMQ is a library that communicates between the distributed compute nodes, it is based on creating a socket for each entity (like client-server).

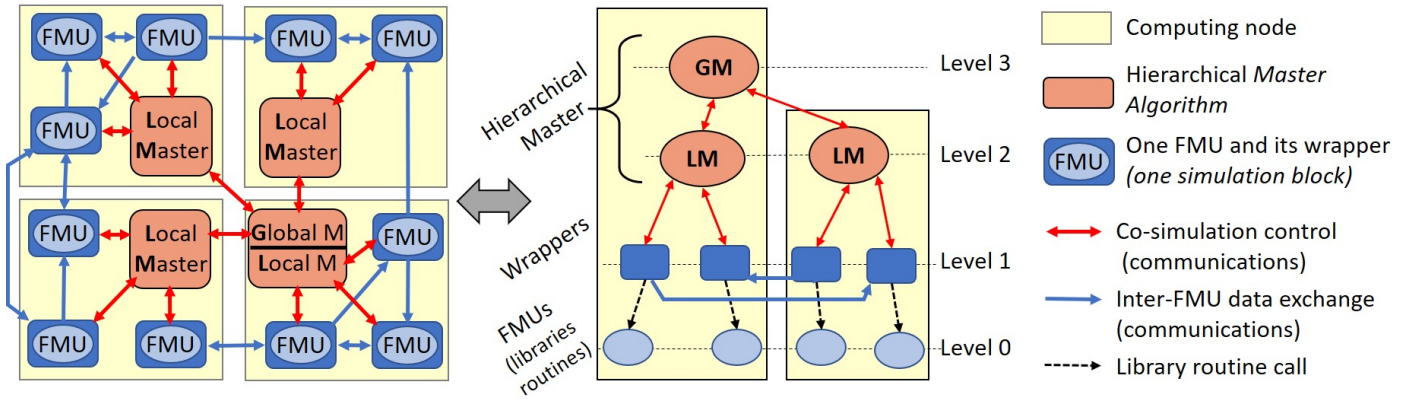

Figure 1: DACCOSIM distributed architecture.

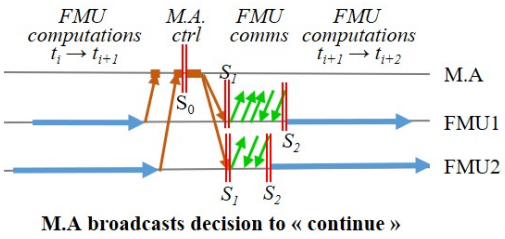

Figure 2: Ordered Orchestration mode.

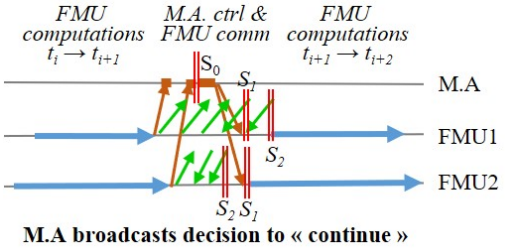

Figure 3: Overlapped orchestration mode.

runs the control and the data-exchange substeps in parallel, to attempt to accelerate the co-simulation.

In both orchestration modes, the *control* substep is when the Hierarchical Master decides if a *rollback* is necessary. Then, it drives the FMUs to the next time step, or returns them to the beginning of the current time step. The execution model of DACCOSIM and its implementation were therefore designed to manage the adaptation of the time stepping, the rollbacks and the resumption of the whole co-simulation process.

In the DACCOSIM 2017 platform, the management of variable time steps and events was not finalized in the Master Algorithm, which is responsible for changing the time steps and reacting to temporal events. However, the lower layer of DACCOSIM is ready to support advanced control algorithms. The following section introduces the events triggering the rollback mechanism before detailing the two orchestration modes implemented in DACCOSIM.

3.2.1. *Rollback mechanism triggering*

In DACCOSIM 2017, the rollback mechanism can be triggered for different reasons:

- a detection of too large imprecision of the FMU calculations within a variable time step. In this case, it is necessary to do a rollback and redo the FMU calculations with a reduced time step.
- the occurrence of a state-event within an FMU computation. A state-event is a discontinuity in the evolution of FMU, for example an alarm. In this case, the rollback mechanism will be triggered to redo the calculations with a shortened time step, to stop the FMU computation at the moment of the appearance of the state-event.
- the occurrence of a time-event to certain FMUs, representing a predetermined external event according to a simulation scenario (such as a control command).

For a given time step, the global master informs all local masters to start their simulations, and each one runs its FMUs in parallel (computation substep). At the end of this time step, each local master collects the information coming from its FMUs concerning the new time step for the next iteration (control substep). This data will be sent to the global master that will take one of the following decisions: (*i*) keep the same time step, (*ii*) increase the time step when all FMU computations are accurate enough or, (*iii*) decrease it and rerun this computation (rollback mechanism) when any FMU reports inaccurate results. If there is no rollback, each local master informs its FMUs to send and receive data between them (communication substep).

3.2.2. *Ordered orchestration mode*

Ordered orchestration of a time step is illustrated on Fig. 2, and follows a *relaxed synchronization* mechanism. All FMU computations are run in parallel to progress from t_i up to $t_{i+1} = t_i + h$ (h is a time step), and as soon as an FMU has finished its computation substep it sends its local requirements to the Master Algorithm (M.A.): to roll back and rerun with a smaller time step (to increase accuracy), to continue with the same time step, or to continue with a greater time step. Then, the M.A. processes each received requirement but awaits all requirements

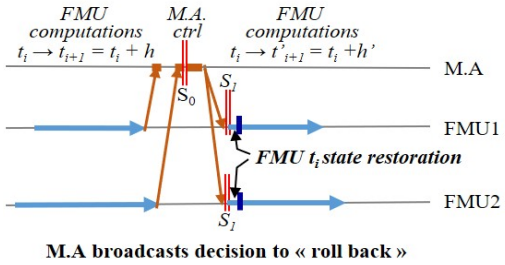

Figure 4: Ordered orchestration mode with rollback.

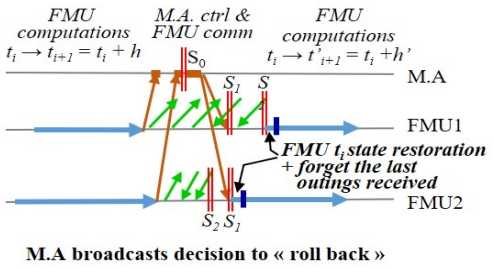

Figure 5: Overlapped orchestration mode with Rollback.

(synchronization point S_0) before taking a global decision, and broadcasting its decision to all FMUs. After that, all FMUs wait for the M.A. global decision, and as soon as an FMU receives the M.A. decision (sync. point S_1), it rolls back or continues its time step.

- If an FMU receives the command to continue (Fig. 2), it enters its communication substep, sending its output results to connected FMUs and waiting for the update of all its input values (sync. point S_2). Finally, as soon as it has updated all its inputs, it enters its next computation substep.
- If an FMU receives the command to roll back (Fig. 4), it restores its previous state at t_i and reruns its computation substep, but progresses from t_i up to $t'_{i+1} = t_i + h'$, with $h' < h$ the new time step broadcasted by the M.A.

So, the only synchronization point looking like a global synchronization barrier is the M.A. decision broadcast, which all FMUs are waiting for. Other synchronization points are relaxed ones, that stop only one task (the M.A. or one FMU). Each task going over a relaxed synchronization point carries on with its work independently of other tasks. *Relaxed synchronization* allows increasing performance, avoiding time consuming synchronization barriers, and avoiding to synchronize all FMUs on the slowest ones (the ones with the longest computations or communications at the current time step). Algorithms with relaxed synchronization schemes are usually more complex to implement and debug, but ZMQ middleware has allowed easy and efficient implementation of these communication and synchronization mechanisms between threads across a PC cluster.

3.2.3. Overlapped orchestration mode

To still reduce the communication cost, a solution consists in overlapping some of the communications with some FMU computations, and with the Master Algorithm decision pending. Fig. 3 illustrates these mechanisms. When an FMU has finished its computation substep, it sends its requirements to the M.A. and, not waiting for M.A. decision broadcast, enters its communication substep. So, FMUs update their input values while the M.A. collects their requirements and broadcasts its global decision. But, depending on the pending time of the M.A decision and on the number of inter-FMU communications, each FMU can cross its synchronization points S_1 (M.A. decision broadcast) and S_2 (all input update received) in any order (see FMU1 and FMU2 examples on Fig. 3). So, when both S_1 and S_2 synchronization points have been crossed, each FMU must consider the M.A. decision:

- If the M.A. issued a command to continue, then each FMU enters its new calculation substep (see Fig. 3), and saved some execution time by performing its inter-FMU communications while the M.A.'s decision was pending.
- If the M.A. has issued a rollback command, then each FMU waits for the end of its communications, restores its state at the beginning of the time step, and restarts its calculation from t_i to t'_{i+1} (see Fig. 5). In this case, the overlap mechanism slightly increased the execution time with unnecessary inter-FMU communications.

From a theoretical point of view, our overlapping mechanism reduces the execution time when there are few rollbacks, or when using constant time steps. But from a technical point of view, some threads will work to send and receive messages while some threads will achieve the end of long FMU computations (M.A. decision broadcast is no longer a synchronization barrier). The communication threads could disturb the ongoing computations and slow down the co-simulation, especially when running more threads than available physical cores. Nevertheless, our overlapped orchestration mode has appeared efficient on our co-simulation of heat transfer inside three-floor buildings, run on a 6-core node cluster with a 10 Gb/s Ethernet interconnect. Section 5 will show the performance achieved on our benchmark of a power grid co-simulation.

3.3. Timing model

We can depict the execution model of DACCOSIM as a task graph $G = (V, E)$, where the set of vertexes V denotes the tasks to be treated (FMUs or masters), and the set of edges E refers to the communications between FMUs, or the communications between the hierarchical master and the FMUs. Fig. 6 shows an example of an FMU graph with four heterogeneous FMUs executing a suite of time steps. All FMUs run their computation substep in parallel during each time step.

In DACCOSIM, the execution time of each FMU encompasses the computation time of the FMU solver T_{Comput}, the control time $T_{Control}$ (including the synchronization times), and the communication time with other FMUs T_{Comm}. In this work, we used the *Ordered orchestration mode* because it makes it possible to model precisely and measure times of the different substeps of each FMU. At the j-th time step (i.e. at $t = j.h$), the

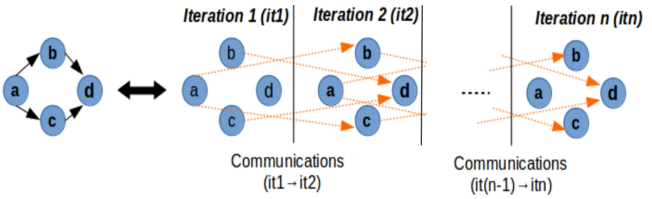

Figure 6: Example of DACCOSIM graph.

co-simulation time of an FMU k is measured as follows:

$$T_{CoSimOneStepFMU}(k,j) = T_{Comput}(k,j,h) + T_{Control} + T_{Comm}(k) \quad (1)$$

T_{Comput} and T_{Comm} depend respectively on the computation complexity and on the connectivity of the FMU k. But, $T_{Control}$ is the same for all FMUs, because the control substep includes synchronization barriers in *Ordered* mode. So, the co-simulation time for a multicore node n containing K_n FMUs, processing the computations of time step j is modeled as follows:

$$T_{CoSimOneStepNode}(n,j) = \max_{1\le k\le K_n} T_{Comput}(k,j,h) + T_{Control} + \max_{1\le k\le K_n} T_{Comm}(k) \quad (2)$$

We consider here an unsaturated execution: $\forall n, K_n \le C_n$, where C_n represents the number of computing cores of node n. This is a reasonable hypothesis as we track performances in our distributed co-simulations, avoiding to run several FMU per physical cores. Moreover, we will see in section 5.3 that sometimes we need to keep: $K_n \simeq C_n/2$. So each FMU computation can run freely on its own core, and the computation time of one step is limited to the computation time of the longest FMU ($\max_{1\le k\le K_n} T_{Comput}(k,j)$). Assuming we have at most one FMU per core, we consider the interconnection network is also unsaturated and the communication time of one step is limited to the most communicating FMU ($\max_{1\le k\le K_n} T_{Comm}(k)$).

Since all local masters (and their FMUs) execute in parallel on different nodes, the overall time of co-simulation of the system for a given time step j is calculated as follows:

$$T_{CosimOneStep}(j) = \max_{1\le n\le N} T_{CoSimOneStepNode}(n,j) \quad (3)$$

Where N is the total number of allocated physical nodes. And, the overall time of the entire co-simulation is calculated by the following equation:

$$T_{Cosim} = \sum_{j=1}^{iterations} T_{CoSimOneStep}(j) \quad (4)$$

Such as "*iterations*" represents the number of iterations of the whole co-simulation. In the case where we adopt a constant time step ($h_i = h = TimeStep$), the number of iterations will be calculated by $iterations = \frac{D}{h}$ such as the D depicts the overall duration of co-simulation. When using a variable time step, the number of iterations will only be known at the end of the co-simulation because it depends on the number of performed *rollbacks*, and the execution of each time step will vary with the step size. This model is suitable for constant time steps but not for variable ones.

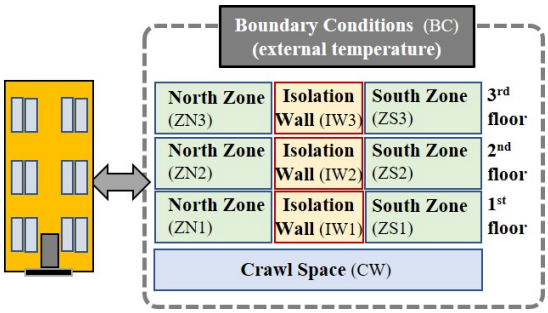

Figure 7: Enerbat use-case, to model heat transfers in a set of n 3-floor buildings with $1 + 10 \times n$ FMUs.

3.4. Toward reliable event management

In this section, we seek to accurately detect state-events at a low-cost by proposing improvements to FMI version 2.0. So, we have proposed to add new primitives in the FMI-CS standard [26] to integrate hybrid co-simulation in a pure FMI-CS environment. Our solution does not require any model adaptation and allows to couple physics models with continuous variability and controllers with discrete variability. Moreover, parallelism is not reduced by our approach, as all FMUs continue to run concurrently either when processing shorter time steps, or when executing rollbacks. Therefore, event handling by the FMI-CS evolution we have proposed does not require changing our parallel and distribution strategy of the FMU co-simulation graph. However, some improvements have been adopted since, but they did not impact the distributed execution model of DACCOSIM [25].

4. Hardware and software testbeds

This section introduces the industrial use-cases provided by EDF (named *Enerbat* and *Ideas*), and the different PC clusters used for benchmarks. The next sections will detail the run of experiments on these testbeds.

4.1. Tested co-simulations

Enerbat is a simplified industrial case, representing heat transfers in a set of n three-floor buildings. Each building is modeled with two zones per floor separated by an indoor wall (see Fig. 7). A single FMU *BC* is responsible for the thermal boundary conditions (e.g. actual temperatures recorded in a French suburb), which in our case are assumed to be the same for all floors of each building. Another FMU *CS* models the crawl space temperature for the lower floor of a building, and each building having a specific FMU *CS*. The FMUs *ZNx* (resp. *ZSx*) represent the Northern (resp. Southern) part of every floor. Each one is designed with Modelica and based on differential equations modeling physical phenomena such as conduction,

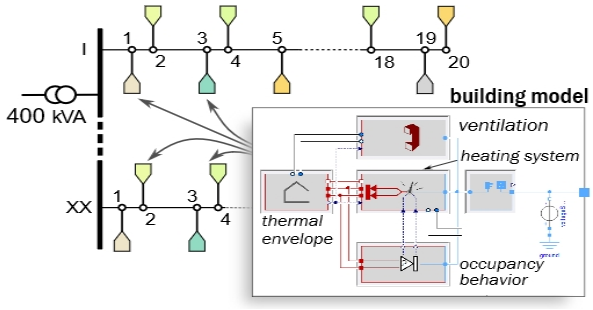

Figure 8: Ideas use-case, to model the thermal behavior and electricity consumption of 1000 buildings with 1042 FMUs.

convection and solar radiation. The FMUs *IWx* detail the behavior of the wall between the floor zones depending on different insulating properties.

All these FMUs are equation-based only, and modeled by encapsulated arrays of records with changeable size to propose 5 levels of complexity (size/weight) for each FMU. No control or temperature regulation is considered in this benchmark, the data exchanged between these FMUs are temperatures and thermal flows. We designed two benchmarks running $1 + 10 \times n$ FMUs. The *cl5* benchmark includes high complexity level FMUs, and exhibits negligible communication times compared to the computation ones. The *cl3* benchmark includes medium complexity level FMUs and has significant communications.

Ideas showed in Fig. 8 is a large industrial use-case [25], that allows to model building thermal behavior and heating system consumptions. It includes a total of 1042 heterogeneous FMUs exported from Dymola[8] 2016 FD01. It complies with the FMI-CS standard, it has also been fully implemented in Modelica using the OpenIDEAS library[9] introduced in [27]. Neither the electrical grid, the heating systems nor the building envelops have been simplified.

Ideas use-case is composed of 1000 buildings connected to low-voltage (LV) feeders, each of them including a thermal envelope, ventilation and heating systems and a stochastic occupancy behavior. The buildings are spread over 20 low-voltage LV feeders, each modeled as one FMU, noted I to XX on Fig. 8. These feeders are connected to a medium-voltage (MV) network that is also simulated with a single FMU. A data-reading FMU provides real medium-voltage measurements that are imposed at the MV substation busbar. The electric grid frequency is provided to different FMUs (buildings and feeders) by 20 additional FMUs. This distributed frequency FMU implementation is meant to reduce inter-node communications since the frequency has to be dispatched to all the FMUs of the use case. Finally, the co-simulation holds a total of 1042 FMUs exported from Dymola 2016 FD01 in conformance with the FMI-CS 2.0 standard. A smaller use-case with fewer buildings and only 442 FMUs, has also been designed to evaluate the scalability of our solution.

[8] A reference tool developed by Dassault Systèmes. It represents both a modeling and a simulating tool based on the Modelica language.

[9] EFRO-SALK project, with support of the European Union, the European Regional Development Fund, Flanders Innovation & Entrepreneurship and the Province of Limburg.

4.2. Execution platforms

To achieve our experiments, we used three (standard) Ethernet PC clusters at CentraleSupelec:

- *Skynet* cluster including 4-core nodes (Intel Core i7-920 CPU) at 2.6 GHz and a 1 Gb/s Ethernet network,
- *Cameron* cluster including 6-core nodes (Intel Xeon E5-1650 CPU) at 3.6 GHz and a 10 Gb/s Ethernet network,
- *Sarah* cluster composed of dual 4-core nodes (Intel Xeon E5-2637 v3 CPU) at 3.5 GHz (*Haswell* architecture) with a 10 Gb/s Ethernet network.

Also, we used a high performance PC cluster at EDF R&D:

- *Porthos* cluster composed of dual 14-core nodes (Intel Xeon E5-2697 CPU) at 2.60 GHz (*Haswell* architecture) with Infiniband FDR communication network.

5. FMU graph parallelization and distribution

5.1. Difficulties related to the nature of FMU graphs

A co-simulation realized by DACCOSIM is represented as a graph of FMUs that integrate their own solvers, but no information can be provided *a priori* by the FMU on the complexity of its calculations, or during execution on its computation load or its volume of communications. Those FMUs are seen as gray boxes, with load unbalanced and frequently exchanging a lot of small data. But achieving large co-simulations requires to run these task graphs on clusters of multicore PCs. So, achieving large and efficient co-simulation based on such tasks, poses some challenges:

1. To implement an adapted communication layer in DACCOSIM.
2. To run the optimal number of FMU per multicore node (one per processor, per physical core, per logical core ?).
3. To design efficient algorithms to distribute and load balance the computation of heterogeneous FMUs, while minimizing inter-node communications.

The first challenge has been dealt with the design of the DACCOSIM architecture, and its two orchestration modes (see sections 3.1 and 3.2). The second challenge is experimentally studied in section 5.3 and the last one is taken up in section 5.5 when designing our FMU distribution algorithms.

5.2. Stopping the frequency auto-tuning of the computing core

In recent years, to minimize its energy consumption, a processor adjusts the frequencies of its cores according to their loads and its overall load. Unfortunately, these energy optimization mechanisms disturbed our study because the calculation load of an FMU depends on its input values, the size of the time step and the start time of the step. Variations in this computational load have an impact on the frequency of the core,

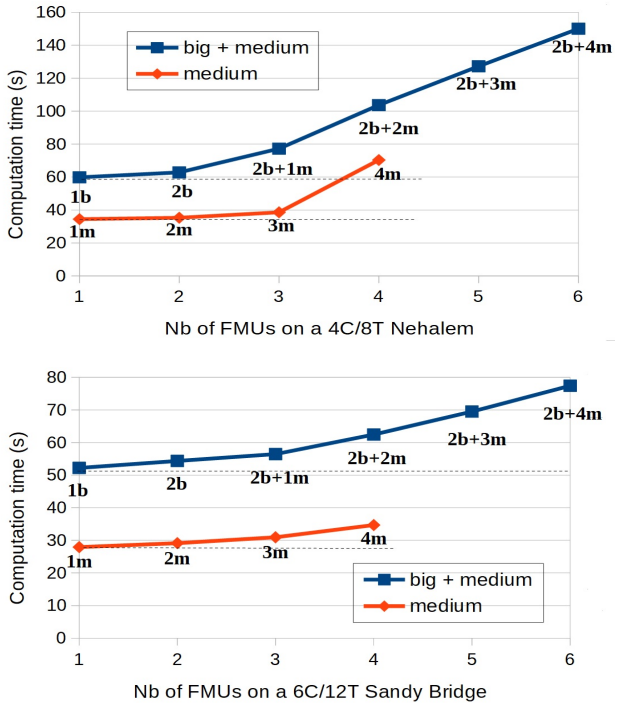

Figure 9: Execution time of concurrent FMUs on a multicore node on *Skynet* (upper) and *Cameron* (bottom) clusters.

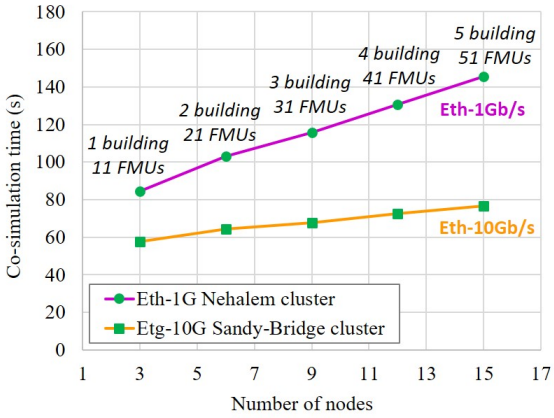

Figure 10: Importance of the interconnection network.

which in turn has an impact on the computational time of the FMU. The performances observed during the experiments fluctuated a lot and masked our load balancing efforts: on *Cameron* cluster, we have observed cores that could operate at 1.33 GHz or 3.1 GHz depending on the overall load of the compute node. When the frequency of the processors varied, it affected the performances of our co-simulations and it became very difficult to study the impact of our FMU distribution algorithms.

So, for our investigations and experiments we forced the cores of the processors to adopt the same frequency by using the *performance mode* (high frequency all the time imposed with the Linux cpufreq-set command).

5.3. Efficiency of FMU concurrent runs on a multicore node

To identify the behavior of FMUs on a multicore PC cluster, we measured the total execution time while running up to k FMU, on a k-core node. Theoretically, this total execution time should remain constant, or bounded by the longer FMU. Therefore, the overall idea of this experience is to put one FMU on a k-core node, and to gradually add other FMUs on the same node until running k FMU on the same k-core physical node.

Fig. 9 introduces the computation time of the simulation according to the number of FMUs on one node of *Skynet* and *Cameron* clusters. Measures were done in realistic conditions: a complete Enerbat use-case was co-simulated on a cluster, FMUs were interconnected, received real input data, and run with constant time steps. The Frequency of all computing cores was fixed to the maximum and remained unchanged. The subset of the FMUs under analysis was located on one specific node and the durations of the computation substeps (see section 3.3) were measured and cumulated on this node. No disk I/O or communications interrupted the computation substeps. Finally, we observe that the computation time does not remain constant but increases up to 20% when running 3 and 4 FMUs respectively on a hyperthreaded quadri-core (*Nehalem* 4C/8T) and a hyperthreaded Hexa-core (*Sandy Bridge* 6C/12T) cluster node. Beyond this number of FMUs, the increase becomes very significant even when running no more than 1 FMU per core. This phenomenon is common on multi-core processors, where the cores share the L2 cache memory and some memory access channels and thus interfere with each other. So, even keeping the maximum number of FMUs M_{FMU} lower than the number of physical cores and fixing their frequency (see the previous section), we have not been able to maintain the computation time of the set of FMUs equal to the most expensive FMU.

In this work, we realized our experiments with homogeneous (blue color) and heterogeneous FMUs (red color) on old processors with *Nehalem* and *Sandy-Bridge* architectures. But, in any case, the overall computation time increases significantly when $M_{FMU} > k/2$. So, we decided to limit the number of FMUs located on our old physical nodes to $k/2$, when we have enough compute nodes to accommodate all FMUs of the system:

$$M_{FMU} \leqslant NbrPhysicalCores/2 \quad (5)$$

On newer and more powerful processors with *haswell* architecture (*Porthos* HPC cluster of EDF), we were able to push this limit up to:

$$M_{FMU} \leqslant NbrPhysicalCores - 2 \quad (6)$$

5.4. Impact of the interconnection network

We also evaluated the impact of the network performances on the overall co-simulation time. We performed some *Enerbat cl3* benchmarks (see section 4.1), and we carried out a *size up experiment* on our 1 Gb/s and 10 Gb/s Ethernet clusters (*Skynet* and *Cameron*). We simulated from 1 up to 5 buildings on 1×3 up to 5×3 multicore nodes, increasing both the number of FMUs and the number of multicore PCs in the same proportions, as shown in the Fig. 10. In the ideal case, we should obtain a straight line: identical co-simulation times for all the problems treated in order to keep perfect parallelism (see section 6.2).

But, we observe through Fig. 10 that each time we increase the size of the problem and the number of nodes, the co-simulation time becomes longer than the previous one, especially if we use a poor communication network (1 Gb/s). On the other hand, the communication network of 10 Gb/s does not lead to perfect size up but gives better performance over the communication network of 1 Gb/s. This test shows that (1) performances of FMU-based applications are sensitive to the interconnection network quality and, (2) such a good quality network is required to achieve size up on our *Enerbat-cl3* benchmark.

5.5. Apportionment on multicore PC clusters

When developing DACCOSIM, one of our objectives was to integrate algorithms of FMU placement on the computing nodes, to easily reach a reasonable co-simulation time. Our approach is based on *experimental heuristic methods* and on *business expert methods*. They belong to *static off-line methods* [28] that define a fixed placement of tasks at the time of program compilation. However, the gray-box facet of the FMUs forces us to design methods not requiring detailed information.

Finally, we designed and identified the following methods, which are applicable to medium and large use-cases and reach reasonable co-simulation times on our benchmarks:

- **Experimental heuristic-based methods:** these heuristics require some preliminary investigations and are designed for small and medium use-cases. We have tested a collection of simple and generic heuristics requiring either (1) only studies of the FMU graph or (2) both FMU graphs and benchmarks on the target cluster. We can quote the most interesting ones: **LB** focuses on load balancing between compute nodes, **COMS** places on the same node the tasks exchanging a lot of data, **LB&COMS** maximizes communications inside each node while balancing the load between nodes.

- **Family Round Robin method (FRR):** when knowledge on the co-simulation is limited and the FMU graph is too large to quickly conduct some investigations, we use the **FRR** method that does not require any study of graphs or benchmarks. **FRR** method regroups FMUs into families with equal or similar computational loads, based on previous experiences of the users. As an example, building thermal calculation FMUs are always significantly more CPU-consuming than power flow calculation FMUs. Each family will form a list of FMUs, and each resulting list will be distributed on the physical nodes using a round-robin algorithm. The purpose of this method is to balance the computational load between the compute nodes without knowing either the exact FMU loads or the communication costs.

- **Business Experts methods (BE):** this method proposed by the EDF team exploits business knowledge on the simulated global system. For example, it proceeds by a division of the FMU graph according to the (geographical) organization at the electrical power net. It leads to loosely-coupled branches with similar simulation complexities, to get both load balancing and limited communications between computational nodes. However, the **BE** method is limited by the natural structure of the FMU graph.

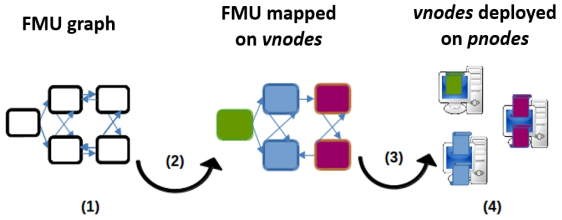

Figure 11: FMU graph mapping on virtual and physical computing nodes.

5.6. Generic distribution & deployment approach

As shown in Fig. 11, the approach taken for the distribution and deployment of DACCOSIM co-simulations on multicore PC clusters considers: FMUs, virtual nodes (*vnodes*) and physical nodes (*pnodes*). It targets cluster with homogeneous nodes, and goes through four steps which are:

1. Design of co-simulation graph with k FMUs, using DACCOSIM platform.
2. Mapping of the k FMUs on n_v virtual nodes (*vnodes*), by applying one of the distribution methods proposed in the previous section to load balance the computations between the *vnodes*. Here, we consider $k \geq n_v$: at less one FMU per *vnode* (no unused *vnode*).
3. Creation of n_v Java codes corresponding to the different local master codes (one Java file per *vnode*).
4. Deployment of v virtual nodes (*vnode*) on p physical nodes (*pnode*) by balancing the number of *vnodes* per *pnode*. computing load, using the DacRun tool[10].

The DACCOSIM virtual node mechanism has not been designed to overload physical nodes but to obtain good performances on any compute node architecture. For instance, launching one *vnode* (i.e. one JVM) per multi-core processor (instead of one per node) can increase performances, improving data locality and cache memory usages (especially on NUMA[11] machines).

6. First *scaling* experiment

In this section, we used the EDF *Enerbat-cl3* benchmark and two 17-nodes PC clusters (see section 4) to conduct our first *scaling* experiment with DACCOSIM 2017. We start measuring some *speedup*, then we investigate a *size up* approach,

[10] A parallel python tool that deploys and runs distributed DACCOSIM co-simulations on a distributed platform, and gathers results.
[11] Non Uniform Memory Architecture.

Use-case complexity	Heuristic	Best speedup	Nb. of nodes
Cl5	LB	2.49	4
Cl3	LB&COMS	1.38	3

Table 2: Best speedup reached on small *1-building Enerbat* benchmark.

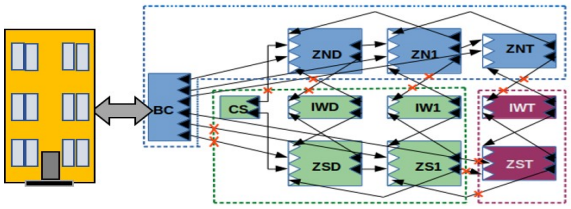

Figure 12: Best distribution of the 11 FMUs of the *1-building Enerbat-Cl3* benchmark on 3 nodes.

and finally we evaluate DACCOSIM 2017 on a complete experiment of *scaling*. As explained in section 3.2, no co-simulation control algorithm using variable time steps was available in DACCOSIM 2017. Thus, all our experiments used constant time steps, but were run on lower-layer mechanisms compatible with variable time steps.

6.1. Measurement of fixed size speedup

Definition: The *fixed size speedup* metric allows us to evaluate the parallelism of a given application, considering a fixed size of data and fixed total amount of computations. It is defined by the following equation:

$$S(p) = T(1)/T(p) \tag{7}$$

where $T(1)$ represents the simulation time obtained on a single computing resource, and $T(p)$ depicts the one obtained on p computing resources with an identical data set. For example, we can compare some runs on p nodes with a run on 1 node inside a PC cluster, or some runs on p cores with a run on 1 core inside a multi-core PC.

Depending on the obtained value, we get: (1) an ideal speedup if $S(p) = p$, (2) a normal speedup, if the result is between 1 and p, (3) a slowdown, if we have a result less than 1 and, (4) an hyper-acceleration when $S(p) > p$ (which has always a logical explanation). See [29] for an introduction to the hyper-acceleration phenomenon.

Experiment: This *speedup* experiment focuses on the distribution of a *1-building Enerbat* co-simulation, including 11 FMUs, on several computing nodes of the *Cameron* cluster (see section 4). This *Speedup* is measured function of the number of used nodes.

As this co-simulation is small (11 FMUs), we tested the set of basic heuristics for FMU distribution presented in the previous section (LB, COMS, LB&COMS), and we experimentally searched for the best distribution, leading to the highest speedup. Tab. 2 shows the best speedup achieved on *Cameron* cluster by running the *cl3* and *cl5* benchmarks according to different heuristics. In *cl5* benchmark, when the computing load of the FMUs is large compared to the communication costs,

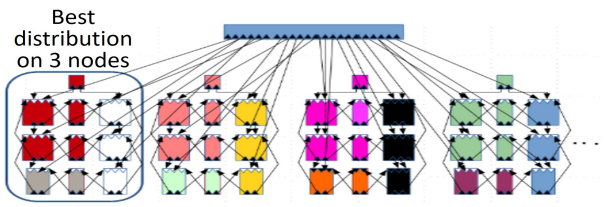

Figure 13: Scaling by replication of 4 buildings on 4×3 physical nodes.

Nb. of buildings	Nb. of nodes	Co-simulation time
1 (11 FMUs)	3	57,69 s
2 (21 FMUs)	6	64.54 s
3 (31 FMUs)	9	67.86 s
4 (41 FMUs)	12	73.43 s
5 (51 FMUs)	15	76.55 s

Table 3: Co-simulation time of n buildings on $3 \times n$ nodes of a Eth-10G cluster.

the pure load balancing strategy LB appears adapted. But, in the case where communications are not negligible, the strategy LB&COMS is (logically) the most desired. Finally, the best distribution of the *1-building Enerbat-cl3* benchmark is achieved on 3 nodes and is illustrated in Fig. 12.

The obtained speedup when simulating one building with 11 FMU is average: only 1.38 on 3 nodes for *Enerbat-cl3* benchmark. This small 1-building co-simulation is too small to reach a good speedup on a PC cluster. However, the following sections evaluate *size up* and *scaling* based on replication of this elementary parallelization, and achieve significant performances.

6.2. Experiment of size up based on replication

Definition: A *size up* experiment handles larger applications by using more computing resources. The *size up* metric checks (1) if we succeed to process bigger problems and, (2) if we can keep constant the co-simulation time regardless of the size of the problem treated. This means to find out the assumption of Gustafson's laws [30, 31] such as:

$$T(q_1, p_1) = T(q_2, p_2) = ... = T(q_k, p_k) = C^{st} \tag{8}$$

where q presents the size of the problem and p, the number of computing nodes.

Maintaining $T(q_k, p_k) = C^{st}$ allows to study larger problems without disturbing the business planning. For example: running larger co-simulations in the morning and still analyzing the results in the afternoon.

Experiment: The initial *Enerbat* use-case that we distributed on 3 nodes did not exceed 11 FMUs (1 building). In this experiment, we increased the number of simulated buildings, *replicating* the best distribution found for one building. We still used 3 nodes per building, as illustrated in Fig. 13 where each color of FMU is associated with one computing node, while one FMU modeling the external temperature was located on one of the nodes and connected to almost all other FMUs.

This first *size up Enerbat-cl3* benchmark exhibits interesting results, already introduced on the bottom curve of Fig. 10

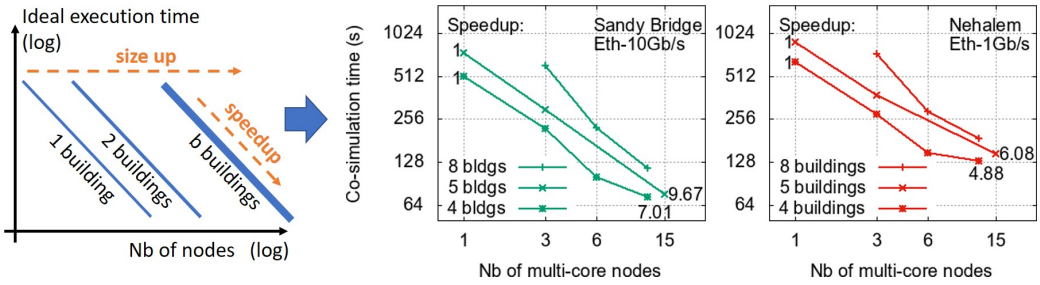

Figure 14: Scaling of *Enerbat-cl3* benchmark on Eth-10G and Eth-1G clusters.

and detailed in Tab. 3: execution times remain almost constant. Despite frequent small communications and the limited speedup obtained when simulating only one building on 3 nodes, we succeeded to obtain a good size up when simulating n buildings on $n \times 3$ nodes of an Eth-10G cluster. Direct FMU-to-FMU communications, not crossing the Master Algorithm, is a key feature of DACCOSIM architecture (see section 3.1). It avoids the bottleneck around a central communication supervisor, which would cause a *size up* benchmark to fail.

6.3. Experiment of scaling based on replication

Definition: After a successful *size up*, we are now expecting a good *scalability* when running larger applications, cumulating *speedup* and *size up* approach. Ideal execution time on p nodes of a q size problem is defined by:

$$T^{ideal}(q, p) = T(q, 1)/p \tag{9}$$

Considering a fixed problem size ($q = q_0$), the ideal curve $T^{ideal}(q_0, p)$ becomes a straight line with -1 slope in log scales, and ideal execution time curves for different problem sizes appear as parallel straight lines (see Fig. 14 left).

So, experimental times curves will be plotted and compared to the ideal ones, and we aim to observe:

- Straight lines: a regular decrease in execution time is obtained, which makes it possible to predict the performance as a function of the number of nodes used (for a given size of the problem).
- Slopes as close as possible to -1: this means that parallelization is very efficient, without significant additional parallelization costs.
- Parallel lines: this means that we observe the same parallelization profiles for different sizes of problems, and that the parallelization strategy remains effective.

Moreover, when experimental curves are close to the ideal ones, it becomes easy to predetermine the minimal number of computing nodes required to process a problem with a given size respecting a time constraint. This property allows keeping the work plan even when studying larger problems.

Experiment: In this first scaling experiment, we continued to reuse our optimal distribution of one building (10 FMU + 1 common FMU) over a group of 3 nodes. But considering a total of $b = k \times n$ buildings, we used $b = k \times 3$ nodes, each group of 3 nodes containing n buildings and the load of the groups remaining balanced. However, we stopped when reaching a maximum of 17 nodes available on our cluster.

$12 = 1 \times 12$ buildings	1×3 nodes	12 buildings per group of 3 nodes
$12 = 2 \times 6$ buildings	2×3 nodes	6 buildings per group of 3 nodes
$12 = 3 \times 4$ buildings *Not experimented*	3×3 nodes	4 buildings per group of 3 nodes
$12 = 4 \times 3$ buildings	4×3 nodes	3 buildings per group of 3 nodes

Table 4: FMU distribution in *Enerbat-cl3* scaling experiment.

For example, we carried out a co-simulation of 12 buildings on 1×3 nodes (12 buildings on one node group), on 2×3 nodes (6 buildings per group) and on 4×3 nodes (3 buildings per group), see Tab. 4. Similarly, we run a co-simulation of 5 buildings on 1×3 nodes and on 5×3 nodes, and we plotted the *T(b buildings, p nodes)* curves.

As we can see in Fig. 14, measured performances are not ideal. But, the curves are closed to straight lines with -1 slope: this first scaling experiment of DACCOSIM 2017 was successful. Also, we have got a good acceleration of 9.67 on *Cameron* and 6.08 on *Skynet* by co-simulating 51 FMUs on 15 computing nodes. For instance, it has not freely allowed to load balance computations and to use 7 or 11 parallel nodes. Our next experiment, on a larger scale, will use the more generic distribution methods introduced in section 5.5.

7. Large power network co-simulation

The *Ideas* industrial benchmark (see section 4.1) on the thermal behavior of buildings and the consumption of heating systems, presents a total of 442 or 1042 heterogeneous FMUs, and is available in both Dymola and DACCOSIM 2017 environments. In this section, we begin by comparing the numerical accuracies of Dymola and DACCOSIM. Then, we identify the most efficient DACCOSIM configuration for our benchmark *Ideas* and, finally, we conduct several experiments on a low cost Eth-10G PC cluster and on a large HPC cluster from EDF.

7.1. Accuracy of numerical results

Some executions have been performed for a one-day simulation with a constant step of one minute. The co-simulation

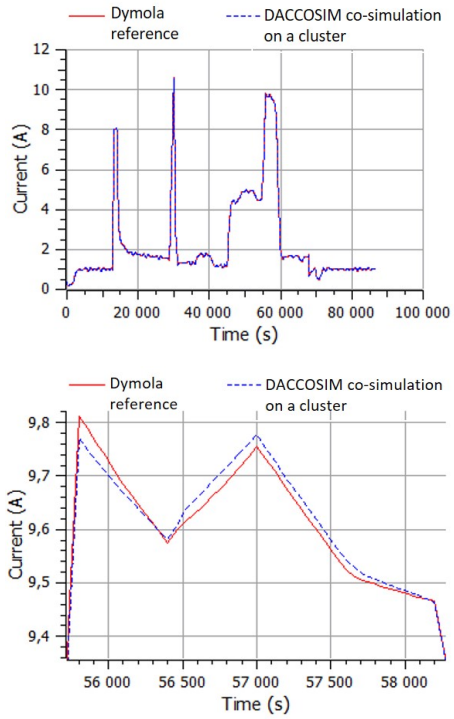

Figure 15: Current from a building of the DACCOSIM co-simulation on a cluster and its Dymola counterpart.

gives realistic results according to expert judgment. Moreover, the energy consumption of the buildings follows the same trend as the one observed on a Dymola reference simulation limited to one 20-building feeder.

To assess the correctness of the co-simulation on a cluster, we selected a single building of the test case and simulated it with Dymola by injecting sampled voltage data obtained from the cluster co-simulation. The power consumed by the building simulated with Dymola and the one co-simulated on the cluster should be the same as the two selected buildings have the same environment: same input voltage, same weather data and same occupancy data. The root mean square error on the current between those two simulations is 1.16×10^{-2} A, with current mainly in the range $1 - 10$ A. The two currents for the one-day simulation are plotted on top of Fig. 15 with a close-up on its bottom. The dynamic of the power consumption is well reproduced thus the DACCOSIM co-simulation on the cluster seems reliable.

7.2. Optimal configuration setting

Before launching a long performance measurement campaign on two different clusters, we have run some limited experiments on our 32-nodes Eth-10G cluster to point out the most efficient configuration of DACCOSIM for this co-simulation. Measured execution times are represented in Fig. 16, and allow to identify: (1) the fastest *orchestration mode* (see section 3.2) and, (2) the most adapted and efficient *distribution method* (see section 5.5).

Orchestration mode: DACCOSIM can overlap inter-FMU communications with longer FMU computations. But an overlapping strategy can speed up some applications and slow down others, depending on many parameters (like the nodes load balancing and the communication times). In Fig. 16, we can observe the *overlapped* orchestration mode slows down the execution of the *Ideas* co-simulation on our Eth-10G cluster. But it could be different when using an Infiniband FDR cluster.

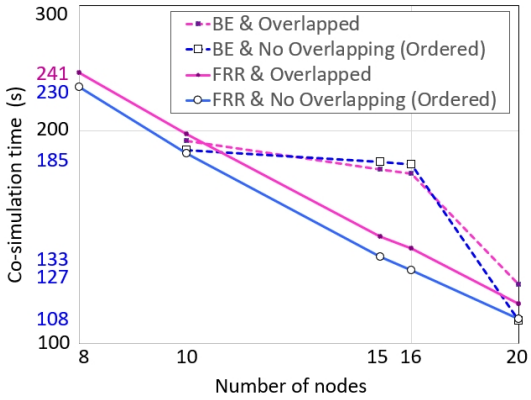

Figure 16: Co-simulation time of FRR *vs* BE methods, on a Eth-10G PC cluster.

So, we plan to use first the ***ordered* orchestration mode** for our performance measurement campaign, and to check again the *overlapped* mode on our Infiniband cluster.

Distribution method: An experiment-based heuristic for FMU graph distributions remains hard to design when dealing with large co-simulations on large computing platforms. For large scale use cases, we focus on methods not requiring deep and expensive prior study: FRR and BE methods.

We observe in Fig. 16, that the BE method gives poor execution times for 15 and 16 nodes. This method consists in cutting out the FMU graph, according to the technical structure it models and the analysis of an expert user. This can result in subgraphs with limited connections and communications, but can also lead to load imbalance, like on 15 and 16 nodes. At the opposite, performances achieved with the **FRR method** appear always satisfying even for 15 and 16 nodes, and for both orchestration modes: the execution times are straight lines with a slope of -0.86 in logarithmic scale. Moreover, the user simply needs to group the FMUs into families with similar loads (see section 5.5), which is easy to do in the *Ideas* use case. The FRR method will therefore be used in our performance measurement campaign.

7.3. Successful scale-up

We executed the *Ideas* use case with respectively 442 FMUs and 1042 FMUs, applying the FFR distribution method on both our cheap and HPC PC clusters. However, we still experimented with the *ordered* and *overlapped* orchestration modes. Fig. 17 illustrates the observed scale-up using 32 to 256 (physical) cores on *Sarah*, and 112 to 1792 cores on *Porthos*.

Experimental performances: The lower half of Fig. 17 shows a very good *scaling* when using our HPC cluster (*Porthos*), according to the criteria introduced in section 6.3:

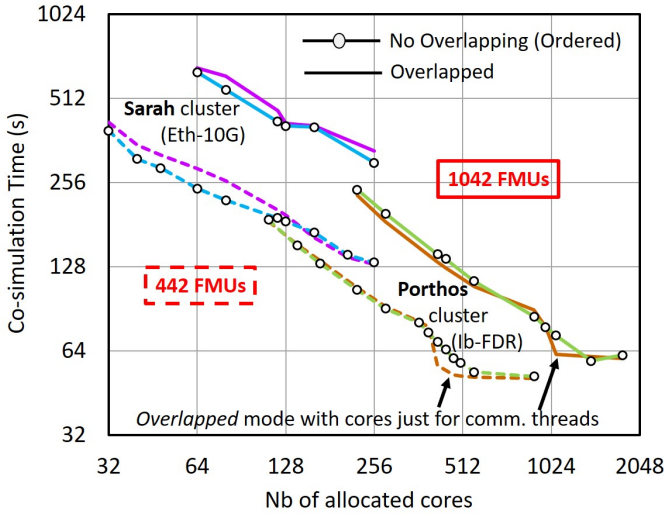

Figure 17: Large scale experiments of *Ideas* use-case.

- regular and almost ideal decrease of the execution time, close to a straight line with -1 slope,
- parallel execution time curves for medium and large configurations, illustrating the same behavior when parallelizing, independently of the problem size and the orchestration mode.

The upper half of Fig. 17 summarizes similar experiments on a less performing cluster (*Sarah*):

- We still observe quasi-straight lines for execution times, but with slopes reduced to -0.5. The communication times are more significant on the Eth-10G network of *Sarah* than on the Infiniband FDR network of *Porthos*. Communications limit the parallelization efficiency on *Sarah* cluster.
- Execution time curves are also parallel on *Sarah* cluster for medium and large problems. Parallelization remains insensitive to the problem size.

Thus, during the execution of the use case *Ideas*, DACCOSIM succeeds to scale up on both our low-cost and high-performance PC clusters, up to a thousand FMUs on a thousand cores.

Impact of orchestration mode: The *overlapped* mode was the fastest in a previous use case on another Eth-10G cluster with smaller nodes [24]. But in the first tests of the *Ideas* use case on the *Sarah* cluster, it appeared slower than the *ordered* mode, which is confirmed by the detailed experiments shown in the upper half of Fig. 17.

On the *Porthos* cluster with the Infiniband FDR interconnect, the two orchestration modes have similar performance until a core is allocated for each FMU (lower half of Fig. 17). When using a high performance interconnect, overlapping *Ideas* communications and computations has a minor impact. However, when allocating more cores than FMUs, it becomes possible to use some cores just to host and execute the communication threads. The *overlap* mode then becomes totally efficient (it no longer disrupts calculations) and performance improves sharply. This phenomenon can be seen in Fig. 17, at the bottom of the curves obtained on *Porthos* cluster. So, with this use-case, the *overlap* mode requires fewer nodes to reach the minimal execution time than the *ordered* mode.

Finally, both orchestration modes are interesting to improve scaling, but the strategy to foresee the right one remains complex and still needs investigation.

8. Conclusion and perspectives

Smart Grids occupy a strategic place in the future multi-energy system, combining heat, gas and electricity. The IT technologies are adding to the outdated electrical networks an ability to improve their performance in a new context based on increasing decentralized production and uncertainties on the production-consumption balance. But Smart Grid development requires large scale co-simulations, aggregating heterogeneous components.

In this paper, we introduced DACCOSIM, our distributed co-simulation platform, based on the FMI standard and designed to run and scale on multi-core PC clusters. Its software architecture implements a hierarchical Master Algorithm controlling a graph of FMUs. Its execution model and implementation support constant or variable time steps, rollbacks of the FMU graph, direct FMU-to-FMU communications and two orchestration modes. Its current co-simulation control algorithm manages constant time steps. We achieved many experiments to measure and optimize performances of co-simulations with DACCOSIM on different multi-core PCs and different PC clusters. Then, we designed some FMU graph distribution strategies, according to the limited available information on each FMU, and we run scalability benchmarks. Finally, a large industrial use case co-simulation with DACCOSIM succeeded to scale up to 1042 FMUs on more than 1000 CPU cores.

Limitations and future works

A co-simulation control algorithm supporting variable time stepping is being added in the next versions of DACCOSIM. Within the French project ModeliScale[12], we also intend to manage this variable stepping with methods inspired by the QSS (Quantized State Systems) methods [32]. In parallel, the logging of results is being improved with interpolation techniques especially the Hermite interpolation.

The testbed used to challenge DACCOSIM 2017 is not large enough to be representative of complex systems. So, we plan to handle wider energy systems, and drive future evolutions of DACCOSIM whose code is being industrialized still under LGPL v3 license on a collaborative BitBucket platform[13].

DACCOSIM 2017 is being redesigned into DACCOSIM-NG (New Generation) to offer a more friendly GUI, to reach

[12] https://systematic-paris-region.org/success_story/modeliscale-for-large-scale-energy-cyber-physical-systems\-labelled-by-the-french-competitive-clusters-systematic

[13] https://bitbucket.org/simulage/daccosim

better performances and to include new functionalities [33]. Redesigning DACCOSIM and running wide systems are also a great opportunity to feedback on the use of the FMI standard and influence the development of the next major release of this standard expected in 2021. Two improvements that we advocate for FMI 3.0 are the efficient treatment of hybrid co-simulation (continuous and discrete) and the consideration of flow causal variables for FMU-to-FMU data exchanges. These improvements will also facilitate the implementation of variable stepping in the Master Algorithm of DACCOSIM.

Acknowledgment: Authors thank Region Grand-Est for its constant support, and RISEGrid institute for the PhD grant that supported this research project.

References

[1] X. Yu, Y. Xue, Smart grids: A cyber–physical systems perspective, Proceedings of the IEEE 104 (5) (2016) 1058–1070.

[2] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: A survey, ACM Comput. Surv. 51 (3) (May 2018).

[3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. v. Peetz, S. Wolf, A. S. Gmbh, Q. Berlin, F. Scai, S. Augustin, The functional mockup interface for tool independent exchange of simulation models, in: Proceedings of the 8th International Modelica Conference, 2011, pp. 105–114.

[4] M. U. Awais, P. Palensky, A. Elsheikh, E. Widi, S. Matthias, Master for co-simulation using FMI, in: Proceedings 8th Modelica Conference, Dresden, Germany, 2011.

[5] R. M. Fujimoto, Parallel and Distribution Simulation Systems, 1st Edition, John Wiley & Sons, Inc., New York, NY, USA, 1999.

[6] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, 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 21 (2) (2006) 548–558.

[7] J. Nutaro, P. T. Kuruganti, L. Miller, S. Mullen, M. Shankar, Integrated hybrid-simulation of electric power and communications systems, in: 2007 IEEE Power Engineering Society General Meeting, 2007, pp. 1–8.

[8] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, J. Thorp, Geco: Global event-driven co-simulation framework for interconnected power system and communication network, IEEE Transactions on Smart Grid 3 (3) (2012) 1444–1456.

[9] H. Georg, S. C. Müller, N. Dorsch, C. Rehtanz, C. Wietfeld, Inspire: Integrated co-simulation of power and ict systems for real-time evaluation, in: Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on, 2013.

[10] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, C. Sureshkumar (Eds.), Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems, Vol. Proceedings of the 10th International Modelica Conference, 2014.

[11] V. Liberatore, A. Al-Hammouri, Smart Grid communication and co-simulation, in: IEEE 2011 EnergyTech, 2011, pp. 1–5.

[12] W. Li, A. Monti, M. Luo, R. A. Dougal, Vpnet: A co-simulation framework for analyzing communication channel effects on power systems, in: 2011 IEEE Electric Ship Technologies Symposium, 2011, pp. 143–149.

[13] V. Galtier, S. Vialle, C. Dad, J. Tavella, J. Lam-Yee-Mui, G. Plessis, Fmi-based distributed multi-simulation with daccosim, in: Proceedings of the 2015 Spring Simulation Multiconference (TMS/DEVS'15), 2015.

[14] B. Camus, V. Galtier, M. Caujolle, V. Chevrier, J. Vaubourg, L. Ciarletta, C. Bourjot, Hybrid co-simulation of FMUs using DEV&DESS in MECSYCO, in: Proceedings of the Symposium on Theory of Modeling & Simulation, TMS/DEVS 2016, part of the 2016 Spring Simulation Multiconference, SpringSim '16, Pasadena, CA, USA, April 3-6, 2016, 2016, p. 8.

[15] S. Rohjans, S. Lehnhoff, S. Schütte, S. Scherfke, S. Hussain, mosaik - a modular platform for the evaluation of agent-based smart grid control, in: IEEE PES ISGT Europe 2013, 2013, pp. 1–5.

[16] B. Palmintier, D. Krishnamurthy, P. Top, S. Smith, J. Daily, J. Fuller, Design of the helics high-performance transmission-distribution-communication-market co-simulation framework, in: 2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2017, pp. 1–6.

[17] B. Möller, The HLA tutorial v1. 0, Pitch Technologies, Sweden (2013).

[18] B. P. Zeigler, T. G. Kim, H. Praehofer, Theory of Modeling and Simulation, 2nd Edition, Academic Press, Inc., USA, 2000.

[19] M. U. Awais, P. Palensky, A. Elsheikh, E. Widi, 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, 2013.

[20] M. U. Awais, W. Mueller, A. Elsheikh, P. Palensky, E. Widi, Using the HLA for distributed continuous simulations, in: The 8th EUROSIM Congress on Modelling and Simulation, Cardiff, UK, 2013.

[21] M. U. Awais, P. Palensky, W. Mueller, E. Widi, A. Elsheikh, Distributed hybrid simulation using the HLA and the Functional Mock-up Interface, in: 39th Annual Conference of the IEEE Industrial Electronics Society (IECON), Vienna, Austria, 2013.

[22] D. S. Schiera, L. Barbierato, A. Lanzini, R. Borchiellini, E. Pons, E. F. Bompard, E. Patti, E. Macii, L. Bottaccioli, A distributed platform for multi-modelling co-simulations of smart building energy behaviour, in: 2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe), 2020, pp. 1–6.

[23] J. Vaubourg, Y. Presse, B. Camus, C. Bourjot, L. Ciarletta, V. Chevrier, J.-P. Tavella, H. Morais, Multi-agent Multi-Model Simulation of Smart Grids in the MS4SG Project, in: Demazeau, Yves, Decker, K. S., B. Pérez, Javier, de la Prieta, Fernando (Eds.), PAAMS'15, Vol. 9086 of Lecture Notes in Computer Science, Springer, Salamanca, Spain, 2015.

[24] C. Dad, S. Vialle, M. Caujolle, J. Tavella, M. Ianotto, Scaling of distributed multi-simulations on multi-core clusters, in: 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2016, Paris, France, June 13-15, 2016, 2016, pp. 142–147.

[25] S. Vialle, J. Tavella, C. Dad, R. Corniglion, M. Caujolle, V. Reinbold, Scaling fmi-cs based multi-simulation beyond thousand fmus on infiniband cluster, in: M. Association (Ed.), 12th International Modelica Conference 2017, Prague, Czech Republic, 2017.

[26] J. Tavella, M. Caujolle, S. Vialle, C. Dad, C. Tan, G., M. Schumann, A. Cuccuru, S. Revol, Toward an accurate and fast hybrid multi-simulation with the FMI-CS standard, in: 21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016, Berlin, Germany, September 6-9, 2016, 2016, pp. 1–5.

[27] R. Baetens, R. De Coninck, F. Jorissen, D. Picard, L. Helsen, D. Saelens, Openideas - an open framework for integrated district energy simulations, 2015, pp. 347–354.

[28] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810 – 837.

[29] C. A. Navarro, N. Hitschfeld-Kahler, L. M. P. Mateu, A survey on parallel computing and its applications in data-parallel problems using GPU architectures, Communications in Computational Physics 15 (2) (2014) 285–329.

[30] X.-H. Sun, J. L. Gustafson, Toward a better parallel performance metric, Parallel Computing 17 (10) (1991) 1093 – 1109.

[31] J. L. Gustafson, G. R. Montry, R. E. Benner, Development of parallel methods for a 1024-processor hypercube, SIAM Journal on Scientific and Statistical Computing 9 (4) (1988) 609–638.

[32] E. Kofman, Quantization-Based Simulation of Differential Algebraic Equation Systems, in: Simulation: Transactions of the Society for Modeling and Simulation International 79(7):363-376, 2003.

[33] J. Évora Gómez, J. J. Hernández Cabrera, J.-P. Tavella, S. Vialle, E. Kremers, L. Frayssinet, Daccosim NG: co-simulation made simpler and faster, in: 13th International Modelica Conference 2019, Regensburg, Germany, 2019.