Co-simulation of FMUs and Distributed Applications with SimGrid

Benjamin Camus, Anne-Cécile Orgerie, Martin Quinson

Co-simulation of FMUs and Distributed Applications with SimGrid

Benjamin Camus
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes
benjamin.camus@inria.fr

Anne-Cécile Orgerie
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes
anne-cecile.orgerie@irisa.fr

Martin Quinson
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes
martin.quinson@irisa.fr

ABSTRACT

The Functional Mock-up Interface (FMI) standard is becoming an essential solution for co-simulation. In this paper, we address a specific issue which arises in the context of Distributed Cyber-Physical System (DCPS) co-simulation where Functional Mock-up Units (FMU) need to interact with distributed application models. The core of the problem is that, in general, complex distributed application behaviors cannot be easily and accurately captured by a modeling formalism but are instead directly specified using a standard programming language. As a consequence, the model of a distributed application is often a concurrent program. The challenge is then to bridge the gap between this programmatic description and the equation-based framework of FMI in order to make FMUs interact with concurrent programs. In this article, we show how we use the unique model of execution of the SimGrid simulation platform to tackle this issue. The platform manages the co-evolution and the interaction between IT models and the different concurrent processes which compose a distributed application code. Thus, SimGrid offers a framework to mix models and concurrent programs. We show then how we specify an FMU as a SimGrid model to solve the DCPS co-simulation issues. Compared to other works of the literature, our solution is not limited to a specific use case and benefits from the versatility and scalability of SimGrid.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; **Simulation tools**; *Continuous simulation*; *Distributed programming languages*; • **Computer systems organization** → *Embedded and cyber-physical systems*;

KEYWORDS

co-simulation, FMI, distributed system, cyber-physical system

ACM Reference Format:
Benjamin Camus, Anne-Cécile Orgerie, and Martin Quinson. 2018. Co-simulation of FMUs and Distributed Applications with SimGrid. In *SIGSIM-PADS '18 : 2018 SIGSIM Principles of Advanced Discrete Simulation, May 23–25, 2018, Rome, Italy*. ACM, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3200921.3200932

SIGSIM-PADS '18 , May 23–25, 2018, Rome, Italy

1 INTRODUCTION

Cyber-Physical Systems (CPS) can be defined as *"physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core"* [27]. In this article, we focus on a specific (although common) class of CPS called Distributed-CPS (DCPS) that are CPS equipped with a geographically distributed computing application –i.e. an application that consists of several concurrent processes, possibly remotely located, and interacting through message exchanges. Such systems include notably smart-grid [21], smart-home [16], cloud infrastructure [12] and smart-city [32].

In most cases, a Modeling and Simulation (M&S) process is required to design and study DCPS systems. However, several expert skills belonging to different scientific fields may be required. In this multidisciplinary approach, each domain comes with its own try and tested models and tools. The challenge is then to have a unified approach with a set of heterogeneous M&S tools (i.e. models and simulation pieces of software). A growing strategy to tackle this challenge is co-simulation [17] which consists in coupling different standalone M&S tools, so that they simulate the whole system together. The advantages of co-simulation include that (1) it enables to study the global behavior of the system, (2) it enforces a clear separation of concerns in the M&S process and (3) it enables to reuse and factorize efforts put into the development and validation of M&S tools. Yet, co-simulation raises two main challenges. First, managing interoperability [14] consists in ensuring that simulation software codes – that may have different API and be written in different programming languages – can exchange usable data during the simulation. Then, the multi-paradigm challenge [29] requires to bridge he different modeling formalisms that are used by the M&S tools. In the case of CPS, this often implies managing a hybrid simulation that combines discrete dynamics (for the computing systems) and continuous dynamics (for the physical systems) [10].

Since 2010, the Functional Mock-up Interface (FMI) standard [4] of the Modelica Association is becoming an essential solution toward co-simulation. It offers a unified framework and an API to control equation-based models of multi-physical systems (e.g. electrical, mechanical, thermal systems). The strength of the standard is that it is supported by

over 100 M&S tools[1]. These tools enable (1) to design a model and export it as an FMU (Functional Mock-up Unit) –i.e. a simulation unit compliant with FMI– (2) and/or to import and use an FMU as a component in their modeling environment. Several frameworks have been proposed to perform co-simulation of FMUs [3, 11, 15], and to integrate continuous FMUs into discrete event formalism environments [7, 13, 24].

In this paper, we address another issue which arises in the context of DCPS co-simulation where FMU components need to interact with distributed application models. The core of the problem is that, in general, complex distributed application behaviors cannot be easily and accurately captured by a modeling formalism (e.g. finite automata) [9]. Instead, the most common approach consists in directly specifying a distributed application using a standard programming language. As a consequence, the model of a distributed application corresponds often a concurrent program which runs on a single computer. The challenge is then to bridge the gap between this programmatic description and the equation-based framework of FMI in order to make FMUs interact with concurrent programs. Considering the diversity of DCPS previously cited, an ad-hoc solution should be avoided in favor of a more versatile approach.

In this article, we show how we use the unique execution model of the SimGrid M&S platform [9] –and more precisely its concept of separated entities' virtualization– to tackle this issue. SimGrid is a versatile platform for the simulation of distributed systems which embeds a set of rigorously validated IT models (e.g. CPU, IP network, disk, energy consumption). The platform manages the co-evolution and the interaction between these models and the different concurrent processes which compose a distributed application code. Thus, SimGrid offers a framework to mix models and concurrent programs. Our contribution is then to specify an FMU as a SimGrid model to ease the simulation of DCPS. Compared to other works in the literature [6, 16, 21, 32], our solution is not limited to a specific use case and benefits from the versatility of SimGrid and its validated IT models.

The rest of the article is organized as follow. In Section 2, we describe for illustration purpose the simple yet representative use-case of a chiller failure in a data-center. This use-case illustrates all along the article the challenges of DCPS simulation and our contributions. Section 3 presents the FMI standard and the SimGrid platform. Section 4 details how we integrate FMU into the SimGrid framework. Finally, Section 5 shows how we validate our proposition with the co-simulation of our use-case.

2 REPRESENTATIVE USE CASE

For illustration purpose, we consider the simulation of a failure in the chiller of a data-center (DC) called DC1. We consider that the failure occurs when the chiller demand (which depends on the heat dissipation induced by computations) becomes too high. After the failure occurrence, a safety mechanism shuts down the power supply if the temperature becomes too high to preserve machines. We want to simulate the computing processes which cause and handle the failure. This requires to model both the distributed application deployed in the DC, and the physical processes of heat transfers.

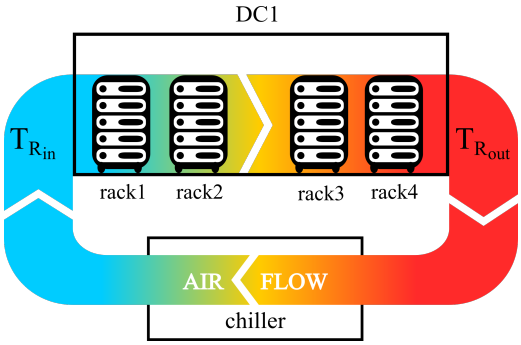


Figure 1: Physical model of the data-center.

This use case is representative because it implies:

(1) coupling different M&S tools (OpenModelica and SimGrid),
(2) which use different modeling paradigms (algebraic/differential/discrete equations and concurrent programs),
(3) with discrete (distributed application execution) and continuous (the temperature evolution) dynamics in interaction (the distributed application changes the computers' heat dissipation, and the room temperature triggers power shutdown that kills the running programs).

In the following we describe the different models, their co-simulation and the faced challenges.

2.1 Physical system of the DC

To describe the nominal behavior of the physical systems, shown in Figure 1, we use a simplified version of the model of [12]. We consider a Computer Room Air Handler (CRAH) that sends an airflow through the computing units (called Physical Machines – PM) racks to cool the DC. Thanks to the chiller, the inlet air temperature is always equal to the same temperature $T_{R_{in}}(t)$. As the air passes through the rack, its temperature increases because of the heat dissipation of the DC $Q_{DC}(t)$. This quantity is defined as follows:

$$Q_{DC}(t) = P_{load_{DC}}(t) + Q_{others_{DC}}(t) \quad (1)$$

$P_{load_{DC}}(t)$ is an input of the model which corresponds to the power consumption and heat dissipation of the PMs. $Q_{others_{DC}}(t)$ corresponds to the heat dissipation of the other devices of the DC (e.g. lighting, Power Distribution Unit) and is equal to:

$$Q_{others_{DC}}(t) = \alpha \times P_{load_{DC}}(t) \quad (2)$$

[1] according to http://fmi-standard.org

Considering the mass of the air in the room m_{air} and its specific heat C_p, the outlet air temperature $T_{R_{out}}(t)$ corresponds to:

$$T_{R_{out}}(t) = T_{R_{in}} + \frac{Q_{DC}(t)}{m_{air} \times C_p} \quad (3)$$

The cooling demand of the chiller $Q_{cooling}(t)$ is defined as follows:

$$Q_{cooling}(t) = Q_{DC}(t)/\eta_{cc} \quad (4)$$

With η_{cc}, the inefficiency in the coil of the CRAH.

When the chiller stops working, the inlet and outlet air temperatures become equal and they start increasing according to the following equation:

$$\frac{dT_{R_{out}}}{dt} = \frac{Q_{DC}(t)}{m_{air} \times C_p} \quad (5)$$

A boolean discrete variable *power* determines the status of the power supply in DC1. It is initially set to 1 meaning that power supply is working. When the temperature reaches a critical threshold *tempThreshold*, the power supply is shut down –i.e. $power = 0$. The following discrete equation models the behavior of the safety mechanism:

$$\textbf{when } T_{R_{out}}(t) >= tempThreshold \textbf{ then } power = 0 \quad (6)$$

The status of the chiller is modeled by a boolean variable *chiller*. The chiller is initially in state 1, meaning that it is working. When the load of the chiller reaches a critical threshold *criticalLoad*, the failure occurs and the status of the chiller switches to 0, meaning that it stops working. This is modeled by the following discrete equation:

$$\textbf{when } Q_{cooling}(t) >= criticalLoad \textbf{ then } chiller = 0 \quad (7)$$

The expected behavior of the system is that when the chiller is working, the inlet temperature remains constant and the load of the chiller varies with the power consumption of the computing units. When that computing load is too high, the chiller load will eventually reach the critical threshold, inducing a chiller failure. The room temperature will then increase with a rate proportional to the PMs' power consumption. Once the temperature reaches the critical threshold, the power supply is shut down, stopping all computing units.

2.2 Distributed application of the DC

We consider a scheduling algorithm which sequentially deploys Virtual Machines (VM) on the PM of the data center DC1, while another data center DC2 is used as a backup solution. The whole IT system is shown in Figure 2.

Each VM is a computer system emulation which requires a given amount of the computing resources (i.e. CPU and memory) of its PM to execute. When creating a VM, the user specifies its size –i.e the maximum amount of computing resources it can use on the PM. Thus, several VMs can run on a single PM with sufficient resources. We consider for the sake of simplicity that a VM always runs at full capacity

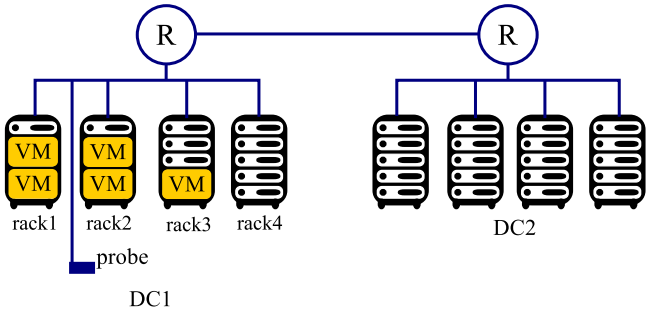


Figure 2: IT model of the data-centers.

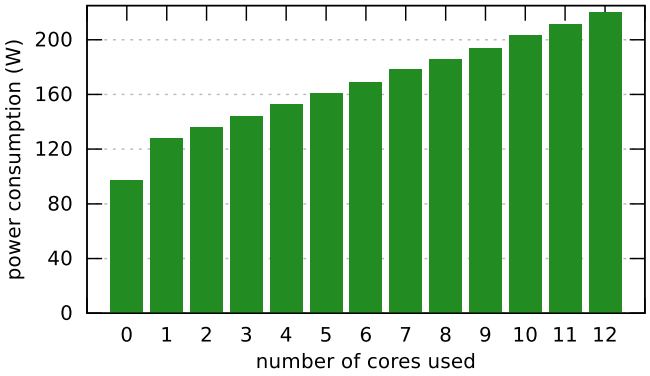


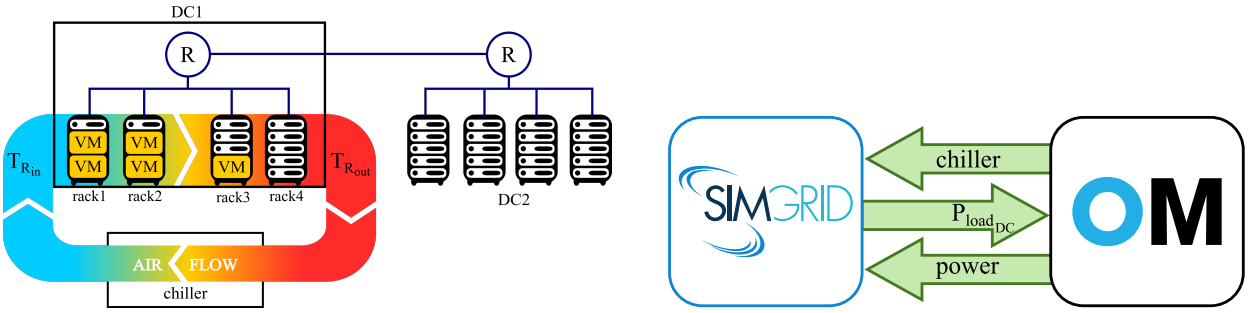
Figure 3: Consumption model of the PM.

when deployed –i.e. 100% of the PM resources required by the VM are used.

The scheduler behavior corresponds to a simple first-fit algorithm (described by Algorithm 1). We consider that it takes 10 seconds to deploy a VM on a PM. PMs consume 0W when turned off. When turned on, we use the power consumption model of [20] which is based on real watt-meter measurements and consider a PM to consumes 97W when idle, 128W when using 1 core, and 220W when working at full capacity. As shown in Figure 3, the power consumption of a PM depends linearly on its CPU usage between 1 core used and max [18]. As a consequence, with Algorithm 1 the power consumption of DC1 will increase with the number of deployed VMs.

Algorithm 1: Scheduling algorithm to deploy VM in DC1

```
while DC1 is not full do
  Let pmList be the list of PM of DC1.
  for pm in pmList do
    if pm as enough cores available to run a VM then
      deploy VM on pm
      break
    end if
  end for
end while
```



(a) Global system studied.

(b) Data exchanges between SimGrid and OpenModelica (OM)

Figure 4: Co-simulation of the chiller failure.

Algorithm 2: Emergency algorithm managing chiller failure in DC1

```
wait for emergency message
shut down unused PM in DC1
Let deployedVM be the list of all VMs deployed on DC1.
for vm in deployedVM do
  Let pm0 be the PM hosting vm.
  Let pmList2 be the list of PMs of DC2.
  for pm1 in pmList2 do
    if pm1 as enough cores available to run vm then
      migrate vm on pm1
      if pm0 is empty then
        shut down pm0
      end if
      break
    end if
  end for
end for
```

A probe monitors the chiller status. When a failure is detected, the probe sends a message through an IP network to an emergency manager deployed in another DC called DC2. Upon reception, the emergency manager kills the scheduling algorithm to stop the deployment of VMs in DC1. It also immediately shuts down unused PMs to limit the room temperature. Then, the emergency manager relocate the VMs to DC2, to save as many as possible of them before the power supply shutdown. We perform live migrations of VM to ensure a continuity of service. With a live migration, the memory of a VM is progressively sent to the destination PM while the VM is running on the source PM. Once the transfer is done, the VM is shut down on the source PM and restarted on the destination PM. The time to migrate a VM strongly depends on the memory transfer time [1]. The manager shuts down the PM once all its VMs are migrated. The behavior of the emergency manager is described by Algorithm 2.

The distributed application is then composed of several processes, namely the scheduling algorithm, the emergency manager processes, all the VM deployed on the PM, and the probe monitoring the chiller. We implement this application in SimGrid, and rely on its IT models of IP network, CPU usage and energy consumption for the simulation.

2.3 Co-simulation challenges

To study the considered use-case, we employ dedicated simulation tools: OpenModelica for the equation-based chiller failure model, and SimGrid for the distributed application and its power consumption over the DC. We need to couple our OpenModelica and SimGrid models as shown in Figure 4. The PM power consumption computed by SimGrid is used as input (i.e. $P_{load_{DC}}(t)$) of the physical models of OpenModelica. On the other side, any change to the model's discrete variables (i.e. *chiller* and *power*) triggers events in the distributed application.

This coupling rises the following issues:

- How to manage interoperability between OpenModelica and SimGrid ?
- How to make the distributed application processes co-evolve with the physical system models ?
- How to manage interactions between the computing processes and the physical models ?
- How to detect discrete state changes in the continuous system evolution, and trigger the induced discrete events in the distributed application ?

In the following, we detail the FMI standard and the SimGrid platform that we want to combine to address these issues.

3 CONTEXT

3.1 The FMI standard

FMI [4] aims at proposing a generic way to export, import and control equation-based models and their solvers. Using FMI, a model which may be composed of a mixture of differential, algebraic and discrete-time equations, can be exported under a standard format as an FMU. This FMU is a black-box (thus protecting the intellectual property of the model) with input and output ports which correspond to the input and output variables of the model. Each FMU can then be controlled

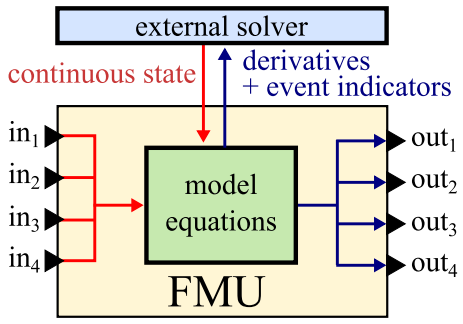


Figure 5: FMI for model-exchange.

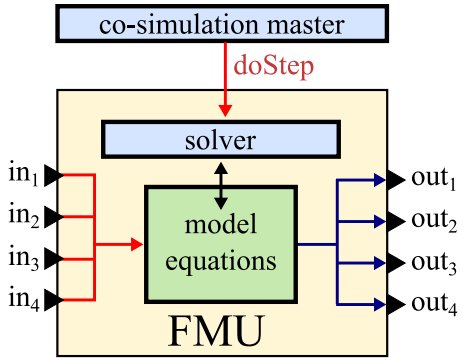


Figure 6: FMI for co-simulation.

using a standardized API, regardless the M&S tool used to generate it. In this way, FMI addresses the interoperability issue between different M&S tools.

FMI provides two ways of exporting and importing a model: FMI for co-simulation (FMI-CS) and FMI for model-exchange (FMI-ME). With **FMI-ME** (depicted in Figure 5), only the model (i.e. the equations) is exported. Thus an FMU-ME requires a numerical solver in order to be simulated. As a consequence, this export mode is mainly interesting in the context of model exchanges between equation-based tools, which is out of the scope of this article.

With **FMI-CS** (depicted in Figure 6), a model is exported with a passive solver that can be controlled by any M&S environment importing this FMU-CS. Because they do not require any external solver, FMU-CS can be integrated into a discrete M&S environments. This fits particularly well into our context of DCPS simulation. In the following, we detail this FMI-CS standard and its limits to make our proposition fully understandable for non-specialist.

The co-evolution of an FMU-CS with its environment is based on the concept of communication points. These communication points, which have to be set by the environment of the FMU, correspond to points in the simulated time where (1) the FMU simulation must be stopped, and (2) exchanges of data can be performed between the FMUs and its environment. Between two communication points, an FMU evolves independently of its environment.

From a software perspective, the FMU interface is composed of a set of C functions, and an XML file. The C functions control the FMU, whereas the XML file describes the FMU capacities and interface. More precisely, the XML file describes names, types (i.e. Real/Integer/Boolean/String), variability (constant/discrete/continuous) and causality (input/output/parameter) of the variables. The C interface enables to control an FMU-CS with the following operations [22]:

- a `doStep` integrates the FMU until a given communication point. Note that, an FMU may enforce fixed communication step-size. This limitation is then specified in the FMU description file. In this case, if the step-size required by `doStep` is not compatible with the fixed step-size, the method will fail to execute.
- a `getOutput` gets the current outputs of the FMU.
- a `setInput` sets the inputs of the FMU. In case of instantaneous dependencies between inputs and outputs, this method may change the outputs of the FMU. Depending on the FMU, a `doStep` of duration 0 may also be required to update outputs.
- `getState` and `setState` are optional operations used to export/import the model state. They enable to perform a rollback during the simulation of the model.

In its current state, FMI-CS is not yet fully compliant with hybrid simulation requirements [5, 11, 28]. In particular, an FMU only supports continuous and piecewise continuous input/output time signals. As a consequence, an FMU-CS cannot produce discrete event signals which are essential to communicate with event-based models. Also, the date of the next discrete event cannot be obtained from a FMU-CS. Although the scheduled 2.1 extension of the standard might solve this issue[2], we need to adapt to this constraint in the meantime.

In the model of our use-case, we use FMI-CS to export our thermal system model as an FMU using the dedicated features of OpenModelica. Thus, we obtain a simulation unit (shown in Figure 7) which is ready to interact and co-evolve with other models in different M&S environments. However, due to the limitation of the standard, the exact dates of the power supply shutdown and the chiller failure cannot be known in advance from the FMU. Also, the FMU is not able to trigger an event at these moments to notify the distributed application.

3.2 The SimGrid Platform

SimGrid [9] is a versatile platform dedicated to the scalable simulation of distributed applications and platforms. It can notably be used to study cluster, grid, peer-to-peer, Cloud, wide/local-area networks. It is grounded on sound simulation models of CPUs [30], TCP/IP networks [31], VMs [19], and energy consumption [18] which are theoretically sound and experimentally assessed. Using these models, SimGrid accurately simulates the resources usage (i.e. CPU and bandwidth sharing), the execution time and the energy consumption

[2]according to http://fmi-standard.org/downloads/ as of 24. Jan 2018.

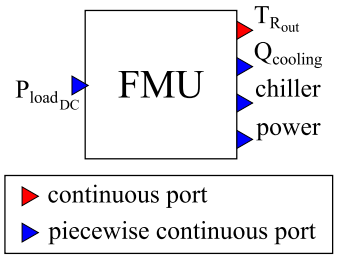


Figure 7: Block diagram view of the thermal system model of our use-case, exported as an FMU.

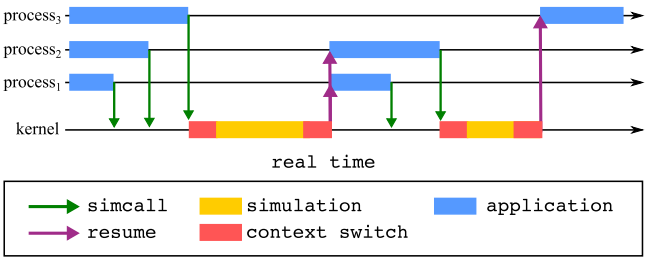


Figure 8: Example of simulation execution in SimGrid (according to the real time).

of a distributed application code. Thus, platform enables to simulate the behavior of a distributed system on a single-computer. SimGrid has grounded over 200 scientific works. SimGrid is implemented in C/C++ and available in open source at http://simgrid.org.

The SimGrid models are used to determine:

(1) how much computing resources are allocated to each action of the distributed application, e.g. how much bandwidth is used by each data transfer.
(2) when each action ends, e.g. when do the data transfers end.

The SimGrid models are based on the discrete event paradigm, where an internal event corresponds to the completion of an action. Each model can be controlled using two operations:

- `updateModel()` updates the model to the current simulated time.
- `nextEvent()` get the date of the next internal event of the model.

SimGrid compartmentalizes the execution of the distributed processes to strictly manage their co-evolution and interactions with the models. This design, where each interaction of the processes with their environment is strictly mediated by a kernel is highly inspired from the classical design of an Operating System. To that extend, SimGrid gives a dedicated execution context to each process of the distributed application. These execution contexts can be implemented either with classical threads or with lighter *continuations* [2] mechanisms.

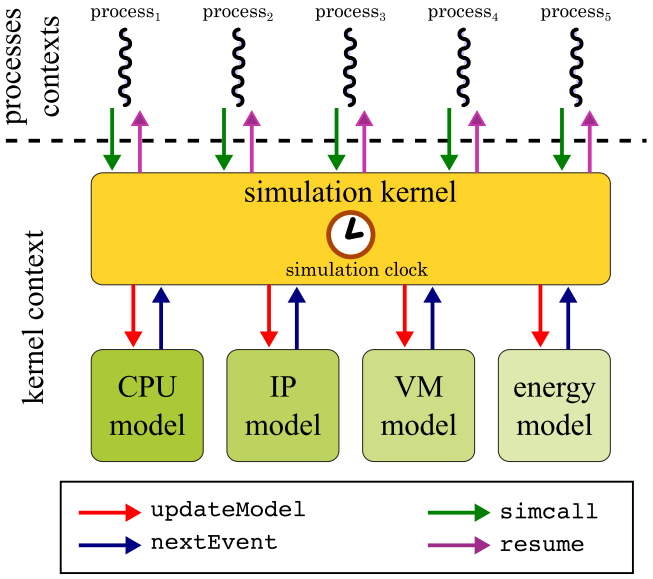


Figure 9: Simulation architecture of SimGrid.

This architecture, depicted in Figures 8 and 9, was first introduced to enable the parallel execution of the user processes [26] without relying of fine-grain locking of the simulator internals. It also enables the formal verification of legacy distributed applications, since their interactions with their environment are strictly controlled by the framework when they run within the simulator [25].

During a typical SimGrid simulation, all user processes are conceptually executed in parallel (in practice, parallelism can be disabled on user request). A **simulation kernel**, which has its own execution context, is in charge of:

(1) managing the simulation state (e.g. the simulation clock),
(2) coordinating the processes and models' executions, and
(3) mediating interactions between user code and models.

Some process actions can modify the state of a model. In our use case, examples of such actions include: starting a VM, sending data through the network, using CPU, migrating a VM and turning off a PM. When a process wants to perform an action that modifies the state of a model, it needs to use the *simcall* mechanism of the kernel. This mechanism is the equivalent of a system call in a classical OS. It causes the process to be blocked until the requested action ends in the simulation. As a consequence (1) the simulation clock may change between the time when a process is blocked and resumed by a simcall, and (2) every process computation that occurs between two simcalls are considered to be instantaneous in the simulated time.

When all processes are blocked at given simulation time, a context switch is performed to the kernel in order to move forward simulation time. The kernel first sequentially (and in a deterministic order) changes the models' states according to the requests made by the processes. Then, following a

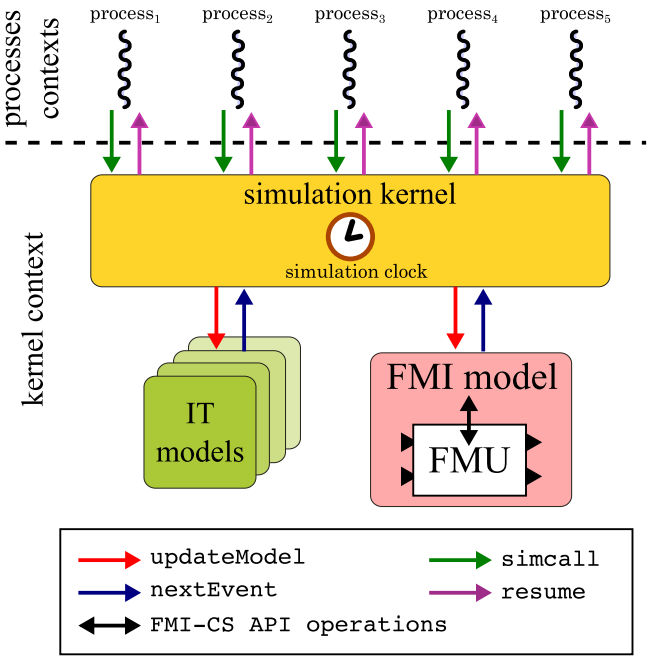


Figure 10: Integration of FMU into the simulation architecture of SimGrid.

discrete event logic, it synchronizes the models and processes executions by:

(1) calling the `nextEvent` method of each model in order to determine the time of the earliest internal event;
(2) updating the simulation clock and sequentially calling the `updateModel` method of all the models in order to move forward the simulation to the earliest internal event time;
(3) eventually resuming the processes whose simcalls ends. This step cause a context switch to the processes.

The Figure 8 summarizes this simulation behavior. Thanks to this unique model of execution, SimGrid can mix simulation models with distributed application code in order to perform scalable and deterministic simulation. We propose to take advantage of this feature in our context of DCPS simulation, by embedding each FMU-CS into a dedicated SimGrid model. In the following, we detail this proposition.

4 CONTRIBUTION

As shown in Figure 10, our approach is to import FMU-CS into a dedicated model which is added in the SimGrid simulation kernel. The kernel can then control the FMU like any other model. Note that with this mechanism, several FMUs can be imported in SimGrid, each of them being associated with a dedicated model. All the FMUs can then interact separately with the distributed application processes.

In traditional discrete event and multi-physical equation-based tools, such as an FMU co-simulation master, or OpenModelica, the input/output connections between an FMU and its environment are determined a priori at the model design time. Moreover, these links are in general hardwired and cannot be changed at run-time. In order to integrate FMU-CS in the SimGrid environment, we use a more flexible interaction mechanism which can handle the complexity of concurrent program behavior. Indeed, any of the computing processes is likely to interact with an FMU. Moreover, some of these processes may be created or killed during the simulation. As a consequence, with some complex concurrent code, it is very difficult –if not impossible– to determine a priori the interactions between each process and an FMU.

In the following, we detail how the FMU execution is coordinated with the distributed application (Section 4.1), how FMUs and distributed programs interact (Section 4.2), and how we overcome the FMI-CS limitation in terms of hybrid simulation (Section 4.3).

We rely on FMI++ [33] to implement our solution. This library provides high-level functionalities to load and manipulate FMUs in order to ease their integration in discrete simulation tools.

4.1 Coordination of FMU and distributed application

When the `updateModel` method of this SimGrid model is called by the kernel, it performs a `doStep` in order to move forward the FMU to the desired simulation time. This mechanism enables then to coordinate the simulations of the FMU and the distributed application.

In order to simulate our use-case, we can then import the FMU of our thermal system model in SimGrid. The thermal system will then co-evolve with the distributed application running in the data-centers.

Note that, as the SimGrid kernel performs discrete event simulations, our synchronization mechanism requires the FMU to support variable communication step-sizes. To handle an FMU that does not comply with this constraint, we can use the dedicated `FixedStepSizeFMU` wrapper of FMI++. This wrapper, implements a fixed step-size simulation algorithm that positions the FMU as close as possible to the required communication point. This may induce inaccurate results as the model synchronization is not perfectly done. However, this inaccuracy is inherent to fixed-step size simulation models.

4.2 Continuous input and output interactions

To enable flexible interactions between the FMUs and the distributed applications, we make all the input and output ports of all the FMUs imported in SimGrid accessible to all the running processes. We extend the SimGrid API with two methods that can be used by the processes in order to interact with the FMUs:

- `getReal/Integer/Boolean/String(string name)` returns the current value of the variable `name`.
- `setReal/Integer/Boolean/String(string name, double/int/bool/string value, bool doStep)`

sets the value of the variable `name` to `value`. If needed, the `doStep` parameter can be set to true in order to perform a `doStep` of duration 0 to update the FMU outputs. As it modifies the state of the FMU –which is now considered as a SimGrid model– this function triggers a SimGrid simcall. The inputs of the FMU is then set within the kernel context of SimGrid in order to maintain a deterministic simulation execution. This simcall ends at the same simulation time, making the set operations instantaneous from the simulation point of view.

In our use-case, we use these methods to update the PM power consumption in the FMU with the value computed by SimGrid. The Algorithms 3 and 4 show how we easily extends the Algorithms 1 and 2 for this purpose. We also design a simple sampling process behavior (shown in Algorithm 5) to observe the continuous state trajectory of the FMU (i.e. the evolution of the temperature in DC1) during the simulation.

Algorithm 3: Extension (in red) of the scheduling algorithm in order to interact with the FMU.

```
while DC1 is not full do
  Let pmList be the list of PM of DC1.
  for pm in pmList do
    if pm as enough cores available to run a VM then
      deploy VM on pm
      Let p be the current power consumption of DC1.
      setReal("P_load_DC",p)
      break
    end if
  end for
end while
```

Algorithm 4: Extension (in red) of the emergency algorithm to interact with the FMU.

```
wait for emergency message
shutdown unused PM in DC1
Let p be the current power consumption of DC1.
setReal("P_load_DC",p)
Let deployedVM be the list of all VM deployed on DC1.
for vm in deployedVM do
  Let pm0 be the PM of vm.
  Let pmList be the list of PM of DC2.
  for pm1 in pmList2 do
    if pm1 as enough cores available to run vm then
      migrate vm on pm1
      if pm0 is empty then
        shutdown pm0
      end if
      Update p.
      setReal("P_load_DC",p)
      break
    end if
  end for
end for
```

Algorithm 5: sampling algorithm which monitors the temperature evolution.

```
while true do
  double temp = getReal("T_Rout")
  save the value of temp in output log
  wait x seconds
end while
```

4.3 discrete event interaction

In order to comply with the requirements of hybrid modeling, we have to overcome the limitations of FMI-CS in terms of discrete event behavior (introduced in Section 3.1). We design an event triggering mechanism which emulates discrete event output signals from the continuous output of the FMU. Thanks to this mechanism, discrete changes in the FMU (e.g. the chiller failure and power supply shutdown in our use case) can then be notified to the distributed application.

We extend the SimGrid API with the following method:

```
registerEvent(bool (*condition)(vector<string>),
              void (*callback)(vector<string>),
              vector<std::string> parameters)
```

This method registers an event notification request to the SimGrid model that embeds the FMU. This mechanism enables flexible interactions between the FMU and the distributed application since any process of the distributed application can use it at any simulated time. Moreover, each process is free to specify in which conditions the event occurs by specifying the `condition` function. Each process also sets what is its impact on the distributed application by specifying the `callback` method. For instance, this effect can range from simple logging activity to process kill, creation, resuming or suspending. When executed, both `condition` and `callback` receive `parameters` as input.

Each time `registerEvent` is called by a process, an instantaneous simcall, in terms of simulated time, is triggered to ensure a deterministic simulation, and the event notification request is stored in the SimGrid model which contains the FMU. The event conditions are evaluated immediately and after each modification of the FMU state –i.e. after each call to setReal/Integer/Boolean/String and `updateModel`. When the condition is satisfied, the associated callback function is executed to propagate the event effect in the distributed application. The event notification request is then deleted (but note that the callback function is free to register another similar request).

When at least one event notification request is pending, we use a classical lookahead exploration strategy in order to accurately determine the occurrence time of events. This strategy, shown in Figure 11, consists in scheduling internal events at regular intervals (using the `nextEvent` method of the SimGrid model) in order to frequently check if an

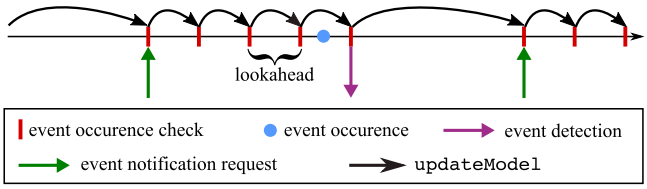


Figure 11: Example of event detection with the lookahead strategy.

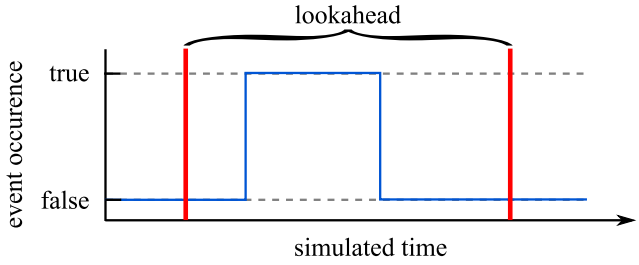


Figure 12: Example of undetected event occurrence with the lookahead strategy.

event has occur in the FMU. The fixed delay between these intervals corresponds to the lookahead of the model. It is determined by the user at the FMU import time. Setting the lookahead is about finding a trade-off between accuracy and performance, as both determine the event detection precision, and the frequency of event checks. A known limitation of this strategy is that it may miss some event occurrences if the lookahead is too large (as shown in Figure 12). We could have used other event detection algorithms that may be more accurate and efficient (like bisectional search, regular falsi or the Illinois algorithm [23]). However, these strategies are less generic as they are only compliant with FMUs which implement the optional rollback features [8]. As soon as there are no pending event notification requests, the SimGrid model containing the FMU stops scheduling internal events.

Getting back to our use-case, we can use this mechanism to handle the events related to the chiller failure and the power supply shutdown. The behavior of the probe that waits for the chiller failure and sends a message to the emergency manager is formalized by the Algorithm 6. This algorithm uses a `zeroValue` condition function that returns true when the *chiller* output variable of the FMU is equal to 0, and a `wakeMeUp` callback function that simply resumes the probe process execution. In order to manage the power supply shutdown, we register an event notification at the beginning of the simulation with a condition function that detects when $power = 0$, and a callback function that creates the PM shutdown process.

5 EVALUATION

To evaluate our solution, we perform the co-simulation of our use-case and study the simulation results. In the following, we detail how we validate our solution. We first describes our experimental scenario in Table 1. Then, we detail and interpret the simulation results in Section 5.1. Finally, we detail our validation process in Section 5.2.

Algorithm 6: algorithm of the probe detecting chiller failure.

```
registerEvent(zeroValue,wakeMeUp,nullptr)
suspend the execution of this process
send the message to the emergency manager
```

5.1 Results interpretation

Figure 13 shows the simulation results of our co-simulation with SimGrid: the power consumption of the system, the temperature of DC1, the status of the chiller, and the power supply over time. We can see that from simulation time 0 to time 260, the power consumption of DC1 (computed by SimGrid) and the chiller load (computed by the FMU-CS) increase progressively. This is due to the scheduling algorithm (Algorithm 1) that is deploying VM every 10 seconds.

At time 260, the chiller load exceeds the critical load of 23,000 W and the chiller stops functioning. As a consequence, the chiller load falls immediately down to 0, the temperature starts increasing, and the probe sends a message to notify this issue. After a transmission delay simulated by the TCP model of SimGrid, this message is received at time 260.263 by the emergency manager (Algorithm 2) which immediately shutdowns the unused PM in DC1. As a consequence, the power consumption of DC1 decreases suddenly from 17,310 W to 9,668.8 W.

Then, from time 260.263 to time 846.614, the emergency manager migrates VMs from DC1 to DC2. This causes an increase of the power consumption of DC2 and a decrease of the power consumption of DC1. Note that the power consumption of DC1 changes at a faster rate than the power consumption of DC2. This is due to the PM shutdowns performed by the emergency manager in DC1. We can see that, by transferring the computation load from DC1 to DC2, the emergency manager progressively limits the temperature increase in DC1. However this is not sufficient and, at time 846.614, the temperature reaches the critical threshold of 40°C and the power supply of DC1 is shut down. This event is successfully propagated to SimGrid as all the PMs are immediately turned off and the total power consumption rapidly reaches 0. These results are in complete accordance with the expected behavior.

5.2 Validation

As shown in Figure 14, to validate this co-simulation, we compare these results with the trajectory generated by a monolithic simulation performed with OpenModelica. As it is not possible to model and simulate our IT system in Modelica, we use the power consumption traces generated by SimGrid as input of OpenModelica. We found similar simulation results. Because it immediately depends on the PM power consumption, the chiller failure occurs at the exact

Domain	Parameters	Value
Physical system model	inlet air temperature $T_{R_{in}}$	24 °C
	mass of the air in the room m_{air}	294 kg
	specific heat	1.006 kJ/kg
	cooling inefficiency	0.9
	chiller critical load *criticalLoad*	23 kW
	temperature threshold *tempThreshold*	40 °C
	ratio of the other devices in the total DC heat dissipation	0.2
IT model	number of PM in each DC	129
	PM core size	12
	PM maximum power consumption	220 W
	PM power consumption when using 1 core	128 W
	PM idle power consumption	97 W
	PM off power consumption	0 W
	VM RAM size	3 Go
	VM core size	6
	VM deployment time (i.e. delay between two deployments)	10 sec
Co-simulation	lookahead value (i.e. event detection precision)	0.01 sec

Table 1: Experimental conditions.

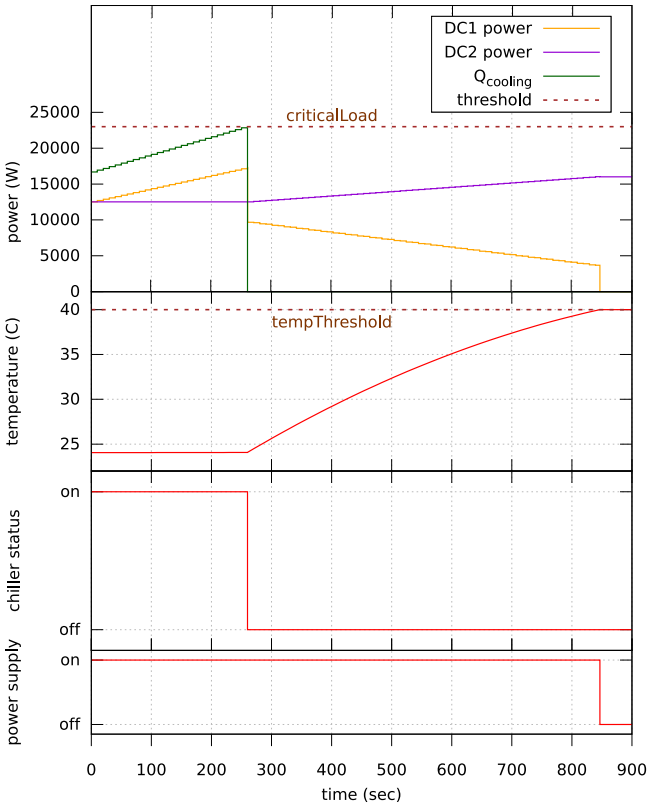


Figure 13: Simulation results of the use-case.

same simulation time. The power shutdown occurs less than 0.001 second earlier in the OpenModelica simulation. This is accordance with our expectations because it corresponds to the event detection precision (i.e. the lookahead) that we set in our co-simulation.

In order to test our capacity to handle several FMUs, we split our physical model into two FMUs. The first FMU embeds the equation 7 and corresponds to the chiller failure model. The second FMU contains the other equations of the physical model. We modify the Algorithm 3 and 4 to update the two FMU inputs. The event notification request related to the chiller failure (resp. power supply shutdown) is now registered in the first (resp. second) FMU. The co-simulation results obtained are again similar to the monolithic simulation. Therefore, these experiments demonstrates the validity of our solution.

6 CONCLUSION

In this article, we detailed how FMU-CS are integrated into the SimGrid platform in order to manage the co-simulation of multi-physical models and concurrent programs. Our contribution consists in embedding FMUs into a dedicated SimGrid model. We proposed an extension of the SimGrid API that enables flexible interactions between FMU and concurrent program. In particular, we defined a dynamic generation mechanism of discrete event signals that overcomes the limitations of FMU in terms of hybrid simulation. We experimentally validated our solution by demonstrating that our co-simulation of a chiller failure in a data-center gives similar results when compared to a monolithic simulation.

From a software perspective, it is important to note that our solution does not require any extension of the SimGrid simulation kernel. It corresponds then to a SimGrid plug-in. All our code is open-source and it will be soon integrated in the SimGrid distribution.

Our solution has several advantages:

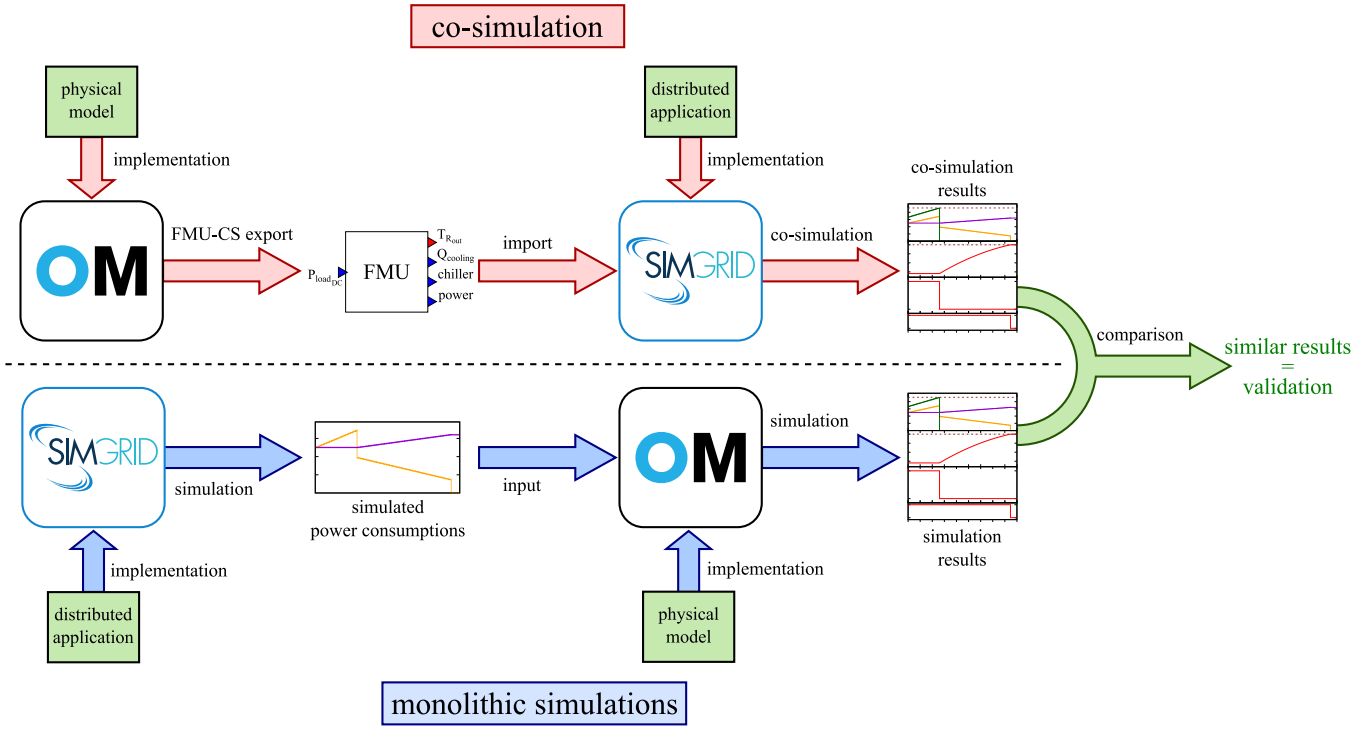


Figure 14: Validation process.

- **versatility:** we can import models from the numerous multi-physical M&S tools that support the FMI-CS standard. Also, we benefit from the validated IT models of SimGrid and its diversity of application contexts. Thus, FMU-CS can now interact with Grids, Clouds, High Performance Computing infrastructures or Peer-to-Peer systems.
- **reproducibility:** we benefit from the OS-based simulation architecture of SimGrid that ensures deterministic co-simulation between FMUs and concurrent programs.
- **scalability:** thanks to the optimized model of execution of SimGrid, FMU-CS can simultaneously interact with a very large amount a concurrent processes.

A limit of our approach is that we rely on a lookahead algorithm to detect event occurrences in the FMU. This strategy may be less efficient and accurate than other algorithms of the literature but it is more generic in term of FMU integration. Also, we are able to import simultaneously several FMUs into SimGrid and make them interact with concurrent processes. However, the potential interactions between these FMU have to be handled in an ad-hoc way. Considering the complexity of FMU interactions which often require numerical methods (e.g. algebraic loop resolution, numerical error estimation) a more generic and automatic solution is required.

In future work, we plan to extend our proposition in order to integrate tools that support the FMU-ME standard. We also plan to implement other event-detection algorithms to increase the accuracy and performance of our co-simulations. As soon as they will be available, we want to take advantage of the new FMI 2.1 extensions for hybrid simulation. Finally, we plan to use this work in order to simulate green computing systems where Smart-Grid models interact with distributed cloud infrastructures.

REFERENCES

[1] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. 2010. Predicting the Performance of Virtual Machine Migration. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, Miami Beach, FL, USA, 37–46. https://doi.org/10.1109/MASCOTS.2010.13

[2] A. W. Appel and T. Jim. 1989. Continuation-passing, Closure-passing Style. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '89)*. ACM, New York, NY, USA, 293–302. https://doi.org/10.1145/75277.75303

[3] Jens Bastian, Christop Clauß, Susann Wolf, and Peter Schneider. 2011. Master for Co-Simulation Using FMI. In *Proceedings of the 8th International Modelica Conference*. Linköping University Electronic Press; Linköpings universitet, Dresden, Germany, 115–120.

[4] Torsten Blochwitz, Martin Otter, Johan Åkesson, et al. 2012. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proc. 9th International Modelica Conference*. The Modelica Association, Munich, Germany, 173–184.

[5] David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. 2013. Determinate Composition of FMUs for Co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*. IEEE Press, Piscataway, NJ, USA.

[6] Benjamin Camus, Fanny Dufossé, and Anne-Cécile Orgerie. 2017. A Stochastic Approach for Optimizing Green Energy Consumption

in Distributed Clouds. In *SMARTGREENS 2017 - Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems*. SciTePress, Porto, Portugal, 47–59. https://doi.org/10.5220/0006306500470059

[7] Benjamin Camus, Virginie Galtier, Mathieu Caujolle, Vincent Chevrier, Julien Vaubourg, Laurent Ciarletta, and Christine Bourjot. 2016. Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*. Society for Modeling & Simulation International (SCS), Pasadena, CA, USA.

[8] Benjamin Camus, Thomas Paris, Julien Vaubourg, Yannick Presse, Christine Bourjot, Laurent Ciarletta, and Vincent Chevrier. 2018. Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware. *SIMULATION* 0, 0 (2018), 0037549717749014. https://doi.org/10.1177/0037549717749014

[9] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *J. Parallel and Distrib. Comput.* 74, 10 (June 2014), 2899–2917. http://hal.inria.fr/hal-01017319

[10] Francois E Cellier. 1979. Combined continuous/discrete system simulation languages–usefulness, experiences and future development. *Methodology in systems modelling and simulation* 9, 1 (1979), 201–220.

[11] Fabio Cremona, Marten Lohstroh, Stavros Tipakis, Christopher Brooks, and Edward A. Lee. 2016. FIDE – An FMI Integrated Development Environment. In *SAC'16*, ACM (Ed.). ACM, Pisa, Italy.

[12] Leandro Fontoura Cupertino, Georges Da Costa, Ariel Oleksiak, Wojciech Piatek, Jean-Marc Pierson, Jaume Salom, Laura Siso, Patricia Stolf, Hongyang Sun, and Thomas Zilio. 2015. Energy-Efficient, Thermal-Aware Modeling and Simulation of Datacenters: The CoolEmAll Approach and Evaluation Results. *Ad Hoc Networks* vol. 25 (February 2015), pp. 535–553. http://oatao.univ-toulouse.fr/15206/ Thanks to Elsevier editor. The definitive version is available at http://www.sciencedirect.com The original PDF of the article can be found at Ad Hoc Networks website : www.sciencedirect.com/science/journal/15708705.

[13] Joachim Denil, Bart Meyers, Paul De Meulenaere, and Hans Vangheluwe. 2015. Explicit Semantic Adaptation of Hybrid Formalisms for FMI Co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (DEVS '15)*. Society for Computer Simulation International, San Diego, CA, USA, 99–106. http://dl.acm.org/citation.cfm?id=2872965.2872979

[14] Saikou Y. Diallo, Heber Herencia-Zapana, Jose J. Padilla, and Andreas Tolk. 2011. Understanding interoperability. In *Proceedings of the 2011 Emerging M&S Applications in Industry and Academia Symposium*. SCS, San Diego, CA, USA, 84–91.

[15] Virginie Galtier, Stephane Vialle, Cherifa Dad, et al. 2015. FMI-Based Distributed Multi-Simulation with DACCOSIM. In *DEVS '15 Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, Munich, Germany, 39–46.

[16] Leilani Gilpin, Laurent Ciarletta, Yannick Presse, Vincent Chevrier, and Virginie Galtier. 2014. Co-simulation Solution using AA4MM-FMI applied to Smart Space Heating Models. In *7th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Lisbon, Portugal, 153–159.

[17] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2017. Co-simulation: State of the art. *CoRR* abs/1702.00686 (2017), 157. arXiv:1702.00686 http://arxiv.org/abs/1702.00686

[18] Franz C. Heinrich, Tom Cornebize, Augustin Degomme, Arnaud Legrand, Alexandra Carpen-Amarie, Sascha Hunold, Anne-Cécile Orgerie, and Martin Quinson. 2017. Predicting the Energy Consumption of MPI Applications at Scale Using a Single Node. In *IEEE Cluster*. IEEE, Honolulu, HI, USA, 92–102.

[19] T. Hirofuchi, A. Lebre, and L. Pouilloux. 2015. SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems. *IEEE Transactions on Cloud Computing* PP, 99 (Sept. 2015), 1–14.

[20] Yunbo Li, Anne-Cécile Orgerie, and Jean-Marc Menaud. 2015. Opportunistic Scheduling in Clouds Partially Powered by Green Energy. In *IEEE International Conference on Green Computing and Communications (GreenCom)*. IEEE, Sydney, Australia, 448–455.

[21] Saurabh Mittal, Mark Ruth, Annabelle Pratt, et al. 2015. A System-of-systems Approach for Integrated Energy Systems Modeling and Simulation. In *Proceedings of the Conference on Summer Computer Simulation*. SCS/ACM, Chicago, Illinois, USA, 1–10.

[22] MODELISAR Consortium and Modelica Association. 2014. Functional Mock-up Interface for Model Exchange and Co-Simulation – Version 2.0, July 25, 2014. Retrieved from https://www.fmi-standard.org. (2014).

[23] Cleve Moler. 1997. Are We There Yet? Zero Crossing and Event Handling for Differential Equations. *Matlab News & Notes* Simulink 2, special edition (1997), 16–17.

[24] W. Muller and E. Widl. 2013. Linking FMI-based components with discrete event systems. In *Proc. SysCon*. IEEE, Orlando, FL, USA, 676–680. https://doi.org/10.1109/SysCon.2013.6549955

[25] Anh Pham, Thierry Jéron, and Martin Quinson. 2017. Verifying MPI Applications with SimGridMC. In *Proceedings of the First International Workshop on Software Correctness for HPC Applications (Correctness'17)*. ACM, New York, NY, USA, 28–33. https://doi.org/10.1145/3145344.3145345

[26] Martin Quinson, Cristian Rosa, and Christophe Thiery. 2012. Parallel Simulation of Peer-to-Peer Systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012) (CCGRID '12)*. IEEE Computer Society, Washington, DC, USA, 668–675. https://doi.org/10.1109/CCGrid.2012.115

[27] Ragunathan (Raj) Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical Systems: The Next Computing Revolution. In *Proceedings of the 47th Design Automation Conference*. ACM, New York, NY, USA, 731–736.

[28] Jean-Philippe Tavella, Mathieu Caujolle, Charles Tan, Gilles Plessis, Mathieu Schumann, Stéphane Vialle, Cherifa Dad, Arnaud Cuccuru, and Sébastien Revol. 2016. Toward an Hybrid Co-simulation with the FMI-CS Standard. (April 2016). Research Report.

[29] Hans Vangheluwe, Juan De Lara, and Pieter J Mosterman. 2002. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems)*. SCS, Lisboa, Portugal, 9–20.

[30] Pedro Velho. 2011. *Accurate and Fast Simulations of Large-Scale Distributed Computing Systems*. Theses. Université Grenoble Alpes.

[31] Pedro Velho, Lucas Schnorr, Henri Casanova, and Arnaud Legrand. 2013. On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. *ACM Transactions on Modeling and Computer Simulation* 23, 4 (Oct. 2013), 23:1–23:26.

[32] Kunpeng Wang, Peer-Olaf Siebers, and Darren Robinson. 2017. Towards Generalized Co-simulation of Urban Energy Systems. *Procedia Engineering* 198 (2017), 366 – 374. https://doi.org/10.1016/j.proeng.2017.07.092 Urban Transitions Conference, Shanghai, September 2016.

[33] E. Widl, W. Müller, A. Elsheikh, M. Hörtenhuber, and P. Palensky. 2013. The FMI++ library: A high-level utility package for FMI for model exchange. In *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, Berkeley, CA, USA, 1–6. https://doi.org/10.1109/MSCPES.2013.6623316