MECSYCO: a Multi-agent DEVS Wrapping Platform for the Co-simulation of Complex Systems

Benjamin Camus, Thomas Paris, Julien Vaubourg, Yannick Presse, Christine Bourjot, Laurent Ciarletta, Vincent Chevrier

MECSYCO: a Multi-agent DEVS Wrapping Platform for the Co-simulation of Complex Systems

Benjamin Camus[1], Thomas Paris[1], Julien Vaubourg[2], Yannick Presse[2], Christine Bourjot[1], Laurent Ciarletta[1], and Vincent Chevrier[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54506, France. firstname.lastname@loria.fr
[2]Inria, 54600 Villers-lès-Nancy, France. firstname.lastname@inria.fr

Abstract

Most modeling and simulation (M&S) questions about complex systems require to take simultaneously account of several points of view. Phenomena evolving at different scales and at different levels of resolution have to be considered. Moreover, expert skills belonging to different scientific fields are needed. The challenges are then to reconcile these heterogeneous points of view, and to integrate each domain tools (formalisms and simulation software) within the rigorous framework of the M&S process. To answer to this issue, we propose here the specifications of the MECSYCO co-simulation middleware. MECSYCO relies on the universality of the DEVS formalism in order to integrate models written in different formalism. This integration is based on a wrapping strategy in order to make models implemented in different simulation software interoperable. The middleware performs the co-simulation in a parallel, decentralized and distributable fashion thanks to its modular multi-agent architecture. We detail how MECSYCO perform hybrid co-simulations by integrating in a generic way already implemented continuous models thanks to the FMI standard, the DEV&DESS formalism and the QSS method. The DEVS wrapping of FMI that we propose is not restricted to MECSYCO but can be performed in any DEVS-based platform. We show the modularity and the genericity of our approach through an iterative smart heating system M&S. Compared to other works in the literature, our proposition is generic thanks to the strong foundation of DEVS and the unifying features of the FMI standard, while being fully specified from the concepts to their implementations.

Keywords : DEVS, FMI/FMU, QSS, DEV&DESS, hybrid modeling, parallel simulation, multi-agent

1 Introduction

In this article, we are interested in the modeling and simulation (M&S) of complex systems. These systems formed a particular type of dynamic systems defined as being *"comprised of a great number of heterogeneous entities, among which local interactions create multiple levels of collective structure and organization"* [1]. Complex systems can correspond to natural or artificial systems: the former range from insect colony (e.g. ants, bees, termites) to collective motion (e.g. crowds, birds flocking)[2], while the latter include cities [3, 4], traffic networks and smart grids.

By experimenting in a rigorous way on a simplification of a complex system (i.e. a model) instead of a real one, the M&S process avoids cost, time and ethic constraints, and thus position itself as a choice tool for the complex systems science. However, when applied in this latter context, the M&S process faces many specific challenges. Indeed, most questions on complex systems require taking simultaneously account of several points of view. Then, we need to consider phenomena evolving at different scales (temporal and spatial), and different levels of resolutions (from micro to macro). Moreover, the expert skills required for describing a system may come from different domains (e.g. for a smart grid: the telecommunication, the information system, the electrical grid), each of them having their own tried and tested models and M&S tools (i.e. formalisms and simulation software). **The challenges are then to reconcile these heterogeneous points of view, and to integrate each domain models and tools within the rigorous framework of the M&S process.**

A very promising strategy to tackles these challenges lies in co-simulation. Co-simulation consists in performing a simulation by reusing models implemented in different simulation software, and managing exchanges of data between these software in order to make their models interact. It allows each specialist involved in a complex system

to keep using the tools which are popular in his/her community while providing to each of them a realistic context. In addition, each simulator can (in some cases) execute on a different machine, which makes possible the co-simulation of very large systems. However, co-simulation faces many issues directly related to the heterogeneity of the models and tools that need to interact.

Our contribution to tackle these issues is twofold in this paper:

- We give the whole operational specification of the MECSYCO (Multi-Agent Environment for Complex-SYstem CO-simulation) co-simulation middleware dedicated to the DEVS (Discrete EVent System specification) wrapping of pre-existing simulation tools.

- We propose DEVS wrappers for the FMI (Functional Mockup Interface) standard in order to make continuous equation-based models interact with discrete event models in a generic way.

The paper is organized as follows. Section 2 details the different challenges related to the M&S of complex systems. Section 3 details how the DEVS formalism -and more precisely the DEVS wrapping strategy- offers an essential solution to these challenges. The Section 4 details the MECSYCO platform which enables the parallel simulation of complex systems models in a rigorous and decentralized way. Section 5 details the promising benefits of the FMI standard regarding the integration of equation-based tools, and how we achieve the DEVS wrapping of this standard. Finally, in Section 9 we validate our proposition with a smart heating use case.

2 Co-simulation Challenges

2.1 Multi-representation integration

When modeling a non-complex dynamic system, we usually describe the system at a specific level of resolution. However, in order to represent a complex system, several levels of resolution may be simultaneously considered (e.g. micro, meso, macro) [5]. Such multi-level representation could be needed, for instance, when there is a lack of expressiveness of one level and a second one is required [6]; when available data explicitly refer to different levels of representation [5]; or when the modeling question is explicitly to study the mutual influences between the coupled levels dynamics[7]. Finally, a multi-level representation can be used in order to find a trade-off between a micro representation which offers more accurate results, and a macro representation which enables speeding-up the simulation[8, 9, 10].

As a consequence, the models involved in a co-simulation may have different representations of the system. The issue is then to reconcile the heterogeneous representations -i.e. given output data of a model which describes the system at a level of resolution, what operations are needed to translate these data into the level of resolution of another model?

2.2 Multi-formalism integration

Because of its heterogeneity, a complex system may exhibit both discrete and continuous dynamics, and several formalisms maybe required to describe the system [11]. Such formalisms may be for instance differential or algebraic equations for the continuous parts, and event-based, finite-state automata or one of the many formalization of the multi-agent paradigm for the discrete part.

As a consequence, discrete and continuous models may interact and co-evolve inside a co-simulation. At the execution level, this formalism heterogeneity implies dealing with different scheduling policies: cyclic or variable time-steps, event-based, etc. A rigorous framework is then needed to integrate these different models in order to have an univocal behavior of the co-simulation [12].

Two solutions exist to integrate these different formalisms[11]:

Translate the models in a same formalism and perform the simulation using the abstract simulator of this formalism. This is the solution chosen by $AToM^3$ [13], which enables to automatically translate two models by a sequence of transformations in their closest common formalism. To do so, $AToM^3$ relies on a Formalism Transformation Graph where each node corresponds to a formalism and each arc represents an existing automatic translation. The shortcoming of this solution is that it forces to rewrite and re-implement the existing models, thus loosing the co-simulation advantages -i.e. effort of translation (if not automatic) and then possibilities of errors.

Use a hybrid M&S formalism which explicitly describe how continuous and discrete systems interact and co-evolve. This super-formalism can be notably DEV&DESS [14] or HFSS (Heterogeneous Flow System Specification)[15], which merge a whole set of traditional techniques used in the field of hybrid modeling. Such techniques include notably (1) the integration of discrete input events during the evolution of the continuous system, and (2) the generation during the simulation of two kinds of discrete-events from the continuous system state: time-events and state-events[12]. While the former consist of events scheduled at predefined simulation times, the latter correspond to events whose occurrence is related to some specific condition on the continuous state (usually when a continuous variable cross a given threshold). From a simu-

lation perspective, the challenge is to integrate in a generic way this discrete-event logic during the numerical resolution of the continuous system (this latter being concern with finding the best trade-off between the accuracy of the solution and the simulation performances[16]). Most notably, the detection and the accurate localization in time of state-events during the simulation is a well known issue in hybrid simulation [17].

2.3 Simulation software interoperability

From a software perspective, co-simulation implies dealing with a heterogeneous set of simulation software. Indeed, as shown in Table 1, the different domain of expertise may have different simulation software which may be implemented in different programming languages and be compliant with different operating systems. Moreover some of these simulation software must be available only on some specific hardware (e.g. if a private license is required). Interoperability processes are then required[18] to synchronize these heterogeneous software execution and manage exchanges of usable data between them [19].

This interoperability can be achieve in an ad-hoc way by directly modifying the simulation software to make them compliant with each others. A more generic solution consist of using a simulation middleware dedicated to the management of the interoperability within the co-simulation. The advantage of this solution is its flexibility which enables easily adding, removing and changing some simulation software without impacting the rest of co-simulation implementation. This is feasible because in this case, the simulation software do not have to be directly interoperable with each others, but have to be interoperable with the middleware instead. The co-simulation middleware can also serves as a communication middleware thus enabling the distribution of the co-simulation, and being therefore compliant with the required hardware and OS diversity.

2.4 Synthesis

To sum up, setting up a co-simulation requires solving a set of specific issues at the representations, the formalisms and the software levels. The solutions that have to be provided are directly related to the heterogeneity found at each of these levels.

Additionally in a M&S process, the need for modularity is required -i.e. to enable adding, removing or changing models and simulation software and their connections without having to redefine all the co-simulation from scratch [24].

In order to fulfill this latter requirements, ad-hoc solutions should be avoided as a more generic and rigorous framework is needed. In the following we details how the DEVS formalism brings solution to these requirements.

3 DEVS as a pivotal formalism for heterogeneity integration

3.1 The DEVS formalism

DEVS [25] is an event-based formalism for the M&S of system of systems. One important feature of DEVS is its universality which positions it as a pivot formalism for multi-paradigm modelling and simulation [26]. Indeed, not only DEVS appears to be universal for describing discrete-event systems [25], but it can also integrate continuous systems [27] expressed for instance with differential equations [28]. Of particular interest in the scope of this article is the fact that, as shown by Zeigler[29], DEVS can also embed the DEV&DESS formalism [14]. This formalism offers a sound framework for describing hybrid systems as it describes how continuous systems interact and co-evolve with the discrete world.

Moreover, DEVS can encapsulate differential and algebraic equation solvers by relying on a quantized integrator approach like the Quantized State Systems (QSS) method [30]. This approach which is based on state quantization instead of the time discretization used by traditional integration methods, shows in some case[31] better performance than these latter [32]. QSS is well-suited for hybrid modeling as it makes the continuous component equivalent to a DEVS model which naturally integrates input events, and makes state-events detection trivial and costless [33].

As summarized by Quesnel[28], the integration of a formalism in DEVS can be performed either by a mapping or a wrapping. While the former consists in establishing the equivalence between the formalisms, the latter implies bridging the gap between the two abstract simulators [34]. The advantage of the wrapping strategy is to enable reusing pre-existing models already implemented in some simulation software [35].

DEVS distinguishes between atomic and coupled models. A DEVS atomic model describes the behavior of the system and corresponds to the structure:

$$M_i = (X_i, Y_i, S, \delta_{ext}, \delta_{int}, \lambda, ta) \quad (1)$$

where:

$X_i = \{(p, v) | p \in InPorts_i, v \in X_i\}$ is the set of input ports and values. These ports can receive external input events,

$Y_i = \{(p, v) | p \in OutPorts_i, v \in Y_i\}$ is the set of output ports and values. These ports can send external output events,

S is the set of the model states,

$\delta_{ext} : Q \times X_i \to S$ is the external transition function (describing how the model reacts to input events) where

Table 1: Example of M&S application domains and their simulation software.

Domain	Simulation software	Languages	Operating system
Collective motion	NetLogo [20]	Java API Java & Scala	GNU/Linux, Windows, Mac OS
	GAMA [21]	Java	GNU/Linux, Windows, Mac OS
Telecom networks	NS-3 [22]	C++, Python API	GNU/Linux
	OMNeT++ [23]	C++	GNU/Linux, Windows, Mac OS
Robotic	VREP	C/C++, Lua, Python, Java	GNU/Linux, Windows, Mac OS
Physical system	Dymola	Proprietary code	Windows
	Matlab/Simulink	Proprietary code, C/C++ API, Fortran	GNU/Linux, Windows, Mac OS

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state of the model,

e is the elapsed time since the last transition,

$\delta_{int} : S \rightarrow S$ is the internal transition function describing the internal dynamic of the model -i.e. the function processes an internal event which changes the model state,

$\lambda : S \rightarrow Y_i$ is the output function describing the output events of the model according to its current state,

$ta : S \rightarrow \mathbb{R}^{+}_{0,\infty}$ is the time advance function describing the time during which the model will stay in the same current state (in the absence of input event). The function is used to get the date of the next internal event.

A coupled model describes the structure of the system. It corresponds to the following structure which describes a set of interconnected atomic models:

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC) \quad (2)$$

where:

$X = \{(p, v) | p \in InPorts, v \in X\}$ is the set of input ports and values

$Y = \{(p, v) | p \in OutPorts, v \in Y\}$ is the set of output ports and values,

D is the set of models id,

$EIC = \{((N, ip_N), (d, ip_d)) | ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$ is the set of external input couplings,

$EOC = \{((d, op_d), (N, op_N)) | op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$ is the set of external output couplings,

$IC = \{((a, op_a), (b, ip_b)) | a, b \in D, op_a \in OutPorts_a, ip_b \in InPorts_b\}$ is the set of internal couplings,

The closure under the coupling of DEVS is an important property which enables hierarchical modeling by proving that a coupled model is equivalent to an atomic one. Thus, a DEVS coupled model can be both composed of interconnected DEVS atomic and coupled models (these latter may be at their turn composed of coupled models etc.). DEVS proposes sequential and parallel abstract simulators and coordinators for respectively simulating the atomic and the coupled models. Thanks to the closure under the coupling of DEVS, these abstract simulators and coordinators can be controlled in an unified way using the DEVS simulation protocol.

3.2 Positioning

DEVS have striking advantages for managing the integration of the formal and the software heterogeneity required by the co-simulation of complex system. This approach is the product of several decades of research, and constitutes a crucial scientific work we must capitalize.

In order to fulfill the requirements of complex system co-simulation, we propose to define a modular co-simulation middleware called MECSYCO (Multi-agent Environment for Complex SYstem CO-simulation) for managing in a generic way software interoperability. In this middleware, we propose to integrate heterogeneous formal models by using DEVS as a pivotal formalism in the following way:

1. integrate the different models in DEVS without implementing them again by using a wrapping strategy.
2. make these wrapped models interact within a DEVS coupled model.
3. simulate the DEVS coupled model using the DEVS simulation protocol in order to perform the co-simulation in an unified way.

Thus, we respond both to the formal integration and to the software interoperability requirements of the complex system co-simulation (detailed in Section 2). In the following section, we details the MECSYCO DEVS wrapping platform which is based on this proposition.

4 The MECSYCO platform

4.1 A Multi-agent Environment for M&S

MECSYCO [36] is a DEVS wrapping platform that takes advantage of the DEVS universality for enabling multi-paradigm co-simulation of complex systems. As shown in previous work [37], the platform also supports multi-level modeling. It is currently used for the M&S of smart electrical grids in the context of a partnership between LORIA/Inria[1] and EDF R&D (leading French electric utility company) [38].

MECSYCO is based on the AA4MM (Agents & Artifacts for Multi-Modeling) paradigm [39] (from an original idea of Bonneaud[40]) that sees an heterogeneous co-simulation as a multi-agent system. Within this scope, each couple model/simulator corresponds to an agent, and the data exchanges between the simulators correspond to the interactions between the agents. Thus, the co-simulation of the system corresponds to the dynamics of interaction between agents. Agents autonomy enables encapsulating legacy software by the use of wrappers[41]. Originality with regard to other multi-agent multi-model approaches is to consider the interactions in an indirect way thanks to the concept of passive computational entities called artifacts [42]. By following this multi-agent paradigm from the concepts to their implementation, MECSYCO ensures a modular, extensible (i.e. features can be easily added such as an observation system) decentralized and distributable parallel co-simulation. MECSYCO implements the AA4MM concepts according to DEVS simulation protocol for coordinating the executions of the simulators and managing interactions between models. In the following, we describe these concepts and their specifications.

4.2 MECSYCO Concepts

MECSYCO relies on four concepts to describe a co-simulation.

A **model** m_i is a partial representation of the target system implemented in a simulation software s_i (symbol in Figure 1a). A model possesses a set of input ports $x_i^{1..n}$ and output ports $y_i^{1..m}$.

An **m-agent** $\mathcal{A}_i$ (symbol in Figure 1b) manages a model m_i and is in charge of the interactions of this model with the other ones. Thus the m-agent is equivalent to a parallel abstract simulator for the models.

Each m-agent $\mathcal{A}_i$ sees its model m_i as a DEVS atomic model thanks to its **model artifact** $\mathcal{I}_i$ (symbol in Figure 1d). Therefore, $\mathcal{I}_i$ acts as a DEVS wrapper for m_i - i.e. it implements the DEVS simulation protocol functions for controlling m_i evolution through s_i.

[1] French IT research institute

(a) model (b) m-agent (c) coupling artifact (d) model artifact

Figure 1: Symbols of the MECSYCO components.

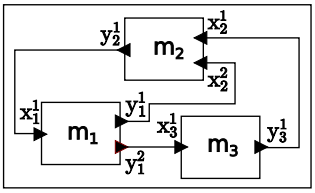


Figure 2: Bloc diagram view of a DEVS coupled model.

Each interaction from an m-agent $\mathcal{A}_i$ to an m-agent $\mathcal{A}_j$ is reified by a **coupling artifact** $\mathcal{C}_j^i$ (symbol in Figure 1c). A coupling artifact $\mathcal{C}_j^i$ works like a mailbox: the artifact has a buffer of events where the m-agents can post their external output events and get their external input events. Thus, a coupling artifact has two roles: for $\mathcal{A}_i$, it is an **output coupling artifact**, whereas for $\mathcal{A}_j$ it is an **input coupling artifact**. The coupling artifacts can transform the data exchanged between the models using operations that can be for instance, spatial and time scaling operations (converting kilometers to meters or hours to minutes), or aggregation/disaggregation operations [37].

According to the multi-agent paradigm, m-agents only have a local knowledge of the coupled model's interconnections. The coupled model's internal coupling set IC is split such as an m-agent $\mathcal{A}_i$ only knows which input coupling artifacts correspond to its model's input ports, and which output coupling artifacts correspond to its model's output ports. We define the set of input links IN_i of $\mathcal{A}_i$ as being composed of the couples (j,k) mapping the input coupling artifact $\mathcal{C}_i^j$ with the input port x_i^k. We define the set of output links OUT_i of $\mathcal{A}_i$ as being composed of the couples (n,j) mapping the output port y_i^n with the output coupling artifact $\mathcal{C}_j^i$.

The connection of the output ports of a model m_i with the input ports of a model m_j is done by the coupling artifact $\mathcal{C}_i^j$. The link from a model m_i to a model m_j (noted as L_j^i) corresponds to the tuple $(n,k,o_{j,k}^{i,n})$. It maps the output port y_i^n with the input port x_j^k and applies the o_k^n operation to transform the event between these two models representation. By default, an operation corresponds to the identity operation id. The Table 2 and the Figure 3 illustrate how a DEVS coupled model (showed in Figure 2) is described in a decentralized and distributable way thanks to MECSYCO.

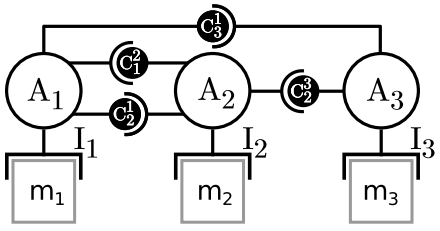


Figure 3: Graphical representation of the MECSYCO co-simulation of Table 2.

Table 2: Decentralized MECSYCO co-simulation of the DEVS coupled model of Figure 2

Descriptions	**Notations**
Output links of m_1	$\mathsf{OUT}_1 = \{(1,2),(2,3)\}$
Input links of m_1	$\mathsf{IN}_1 = \{(2,1)\}$
Output links of m_2	$\mathsf{OUT}_2 = \{(1,1)\}$
Input links of m_2	$\mathsf{IN}_2 = \{(1,2),(3,1)\}$
Output links of m_3	$\mathsf{OUT}_3 = \{(1,2)\}$
Input links of m_3	$\mathsf{IN}_3 = \{(1,1)\}$
Links from m_1 to m_2	$\mathsf{L}_2^1 = \{(1,2,o_{2,2}^{1,1})\}$
Links from m_1 to m_3	$\mathsf{L}_3^1 = \{(2,1,o_{3,1}^{1,2})\}$
Links from m_2 to m_1	$\mathsf{L}_1^2 = \{(1,1,o_{1,1}^{2,1})\}$
Links from m_3 to m_2	$\mathsf{L}_2^3 = \{(1,1,o_{2,1}^{3,1})\}$

4.3 Operational Specifications

The behavior of each m-agent corresponds to the DEVS conservative parallel abstract simulator which is based on the Chandy-Misra-Bryant (CMB) algorithm [43, 44]. This algorithm is proven to be deadlock free and to respect the causality constraint [25] -i.e. to ensure that the *"execution of the simulation program on a parallel computer will produce exactly the same results as an execution on a sequential computer"* [45].

Within this behavior, each m-agent $\mathcal{A}_i$ shares in its environment its Earliest Output Time estimate noted EOT_i. EOT_i corresponds to the date (in simulation time), below which $\mathcal{A}_i$ guarantees it will not send new external output event. $\mathcal{A}_i$ shares EOT_i in the **link time** of each of its output coupling artifact. The link time of a coupling artifact $\mathcal{C}_j^i$ is noted LT_j^i and correspond to the simulated time (initially equals to 0) up to which $\mathcal{A}_i$ has simulated the links from m_i to m_j [43].

Each m-agent $\mathcal{A}_i$ uses the link times of all of its input coupling artifacts to compute its Earliest Input Time estimate noted EIT_i. This EIT_i corresponds to the date (in simulated time) below which $\mathcal{A}_i$ will not receive any new external input event. EIT_i corresponds to the minimum link time of all of $\mathcal{A}_i$'s input coupling artifacts.

For each m-agent $\mathcal{A}_i$, all the events (internal or external) with a timestamp inferior or equal to EIT_i are said to be safe to process. In order to fulfill the causality constraint, each m-agent must process only safe events and in an increasing timestamped order.

Each EOT_i is given by the $\texttt{Lookahead}_i$ function:

$$Lookahead_i() = min\{nt_i, \mathsf{EIT}_i + D_i, t_{in_i+D_i}\} \quad (3)$$

with nt_i the next internal event time of m_i, t_{in_i} the time of the earliest event waiting to be processed in $\mathcal{A}_i$'s input coupling artifact, and $D_i (D_i > 0)$ the minimum propagation delay of m_i. This minimum propagation delay corresponds to the minimum delay (in simulated time) below which the processing of an external event can not schedule a new internal event in a model m_i. D_i has to be determined for each model m_i in the co-simulation.

This behavior which enables simulating a model until a time Z is formalized within the MECSYCO paradigm by the Algorithm 1 basing on the artifacts specifications detailed below.

A coupling artifact $\mathcal{C}_j^i$ proposes six functions to $\mathcal{A}_i$ and $\mathcal{A}_j$:

- `post` (e_{out}^n), n stores and transforms (according to $\mathcal{C}_j^i$'s operation) the external output event e_{out}^k of output port y_i^n, in the artifact's buffer.
- `getEarliestEvent` (k) returns the earliest external input event for the k^{th} input port of m_j, x_j^k.

- `getEarliestEventTime(`k`)` returns the time of the earliest external event for x_j^k.
- `removeEarliestEvent(`k`)` removes from the artifact's buffer the earliest external event for x_j^k.
- `setLinkTime(`t_i`)` set LT_j^i to t_i.
- `getLinkTime()` returns LT_j^i.

In order to manipulate m_i, each model artifact $\mathcal{I}_i$ proposes the following DEVS simulation protocol functions to $\mathcal{A}_i$. These functions, which are listed below, have to be defined for each simulation software:

- `init()` initializes the model m_i. It sets the parameters and the initial state of the model,
- `processExternalEvent(`e_{in_i}, t_i, x_i^k`)` processes the external input event e_{in_i} at simulation time t_i in the k^{th} input port of m_i, x_i^k,
- `processInternalEvent(`t_i`)` processes the internal event of the model m_i scheduled at time t_i,
- `getOutputEvent(`y_i^n`)` returns $e_{out_i}^n$, the external output event at the n^{th} output port of m_i, y_i^n,
- `getNextInternalEventTime()` returns the time of the earliest scheduled internal event of the model m_i.

4.4 Implementation

MECSYCO is currently implemented in Java (available at http://mecsyco.com) and C++. In order to make these two versions interoperable and to perform distributed co-simulations, MECSYCO relies on the JSON format and the OpenSplice implementation of the OMG standard DDS (Data Distribution Service). Using Opensplice, the coupling artifacts are divided in two parts, reader and writer, in order to split the co-simulation. DDS being based on the publish-subscribe communication pattern, writers coupling artifacts play the role of publishers while reader coupling artifacts act as subscribers. Each writer coupling artifact send data to its reader coupling artifact using a specific DDS topic (see Figure 4).

The UML diagram of Figure 5 shows how we implement the MECSYCO concepts following an object oriented programming. This implementation is in keeping with our multi-agent paradigm as each MECSYCO concept corresponds to a class of object, and as each autonomous magent corresponds to a thread. We retain then the advantages of our paradigm: the software architecture is composed of a set of modular software bricks which enables a decentralized and parallel simulation.

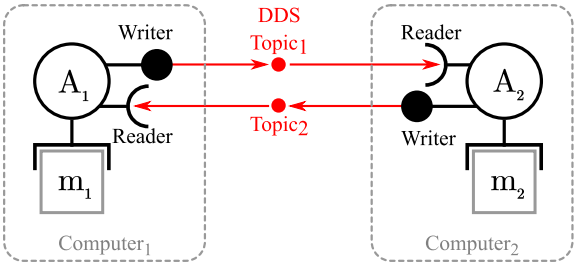


Figure 4: Distribution of a MECSYCO co-simulation.

4.5 DEVS wrapping of simulation software

So far, we successfully define DEVS wrappers for discrete modeling tools like the MAS simulator NetLogo [36], and the telecommunication network simulators NS-3 and OMNeT++ [46]. Aside from several difficulties met when wrapping NS-3 and OMNeT++ (mainly due to the high level of modeling details offered by these platforms, as well as to the complexity of the opening and the distribution of their telecommunication models), making these discrete modeling tools compliant with the DEVS simulation protocol was a straightforward process. This is due to the fact that these platforms have a discrete modeling paradigm which is very close to DEVS.

However, to our experience with MECSYCO several difficulties may arise when wrapping a simulation tools in DEVS. These problems depend mainly on two criteria:

- **the M&S formalism used by the tools** may not be explicitly defined by the software specification, and/or may be very different from DEVS. The challenge is then to answer the questions: what is the formalism used by the tools? How to bridge the formal gap between this formalism and DEVS?
- **the software interface** with the middleware may be difficult to produce as the tools API and the software architecture are not always documented and fully compliant with the DEVS simulation protocol. Moreover, the software may not be conceived to be externally manipulated.

Things getting especially complex with equation-based tools as their continuous modeling paradigm is very different from the discrete DEVS one. Thus, we need to bridge the gap between the discrete and the continuous paradigms, and a more complex wrapping strategy based on the hybrid capacity of DEVS is required. Regarding this issue, wrapping each of these equation-based tools (e.g. OpenModelica, Dymola, Matlab/Simulink) separately would be very inefficient.

Hopefully, a more efficient solution exists: most of these tools are compatible with the FMI standard which brings a generic API to manipulate equation-based models and

Algorithm 1 $\mathcal{A}_i$ m-agent behavior.

INPUT: IN_i, OUT_i, Dt_i
OUTPUT:

$nt_i \leftarrow \mathcal{I}_i$.getNextEventTime()
$t_{in_i} \leftarrow +\infty$
$\mathsf{EOT}_i \leftarrow 0$
$\mathsf{EIT}_i \leftarrow 0$
▷ while the end of simulation.
while $(\neg endOfSimulation)$ **do**
  $\mathsf{EIT}_i \leftarrow +\infty$
  $t_{in_i} \leftarrow +\infty$
  for all $(j,k) \in \mathsf{IN}_i$ **do**
    if $\mathcal{C}_i^j$.getLinkTime() $< \mathsf{EIT}_i$ **then** ▷ Compute EIT_i
      $\mathsf{EIT}_i \leftarrow \mathcal{C}_i^j$.getLinkTime()
    end if
    if $\mathcal{C}_i^j$.getEarliestEventTime(k) $< t_{in_i}$ **then** ▷ Take the next external event
      $t_{in_i} \leftarrow \mathcal{C}_i^j$.getEarliestEventTime($k$)
      $ein_i \leftarrow \mathcal{C}_i^j$.getEarliestEvent($k$)
      $p \leftarrow k$ ▷ Save the corresponding input port
      $c \leftarrow j$ ▷ Save the corresponding coupling artifact.
    end if
  end for
  ▷ Compute EOT_i and update output coupling artifact
  if $\mathsf{EOT}_i \neq \mathtt{Lookahead}_i(nt_i, EIT_i, t_{in_i})$ **then**
    $\mathsf{EOT}_i \leftarrow \mathtt{Lookahead}_i(nt_i, EIT_i, t_{in_i})$
    $\forall (k,j) \in \mathsf{OUT}_i : \mathcal{C}_j^i$.setLinkTime($\mathsf{EOT}_i$)
  end if
  ▷ Find the next secured (internalor exteral) event
  if $(nt_i \leq t_{in_i})$ **and** $(nt_i \leq \mathsf{EIT}_i)$ **and** $(nt_i \leq \mathsf{Z})$ **then** ▷ if the event is internal
    $\mathcal{I}_i$.processInternalEvent(nt_i) ▷ process the event
    for all $(k,j) \in \mathsf{OUT}_i$ **do** ▷ Send the resulting external output event
      $eout_i^k \leftarrow \mathcal{I}_i$.getOutputEvent($y_i^k$)
      if $eout_i^k \neq \varnothing$ **then**
        $\mathcal{C}_j^i$.post($eout_i^k$, nt_i)
      end if
    end for
    $nt_i \leftarrow \mathcal{I}_i$.getNextInternalEventTime()
  else if $(t_{in_i} < nt_i)$ **and** $(t_{in_i} \leq \mathsf{EIT}_i)$ **and** $(t_{in_i} \leq \mathsf{Z})$ **then** ▷ if the event is external
    $\mathcal{I}_i.processExternalEvent(ein_i, t_{in_i}, x_i^p)$ ▷ process the event
    $\mathcal{C}_i^c$.removeEarliestEvent(p)
    $nt_i \leftarrow \mathcal{I}_i$.getNextInternalEventTime()
  end if
end while

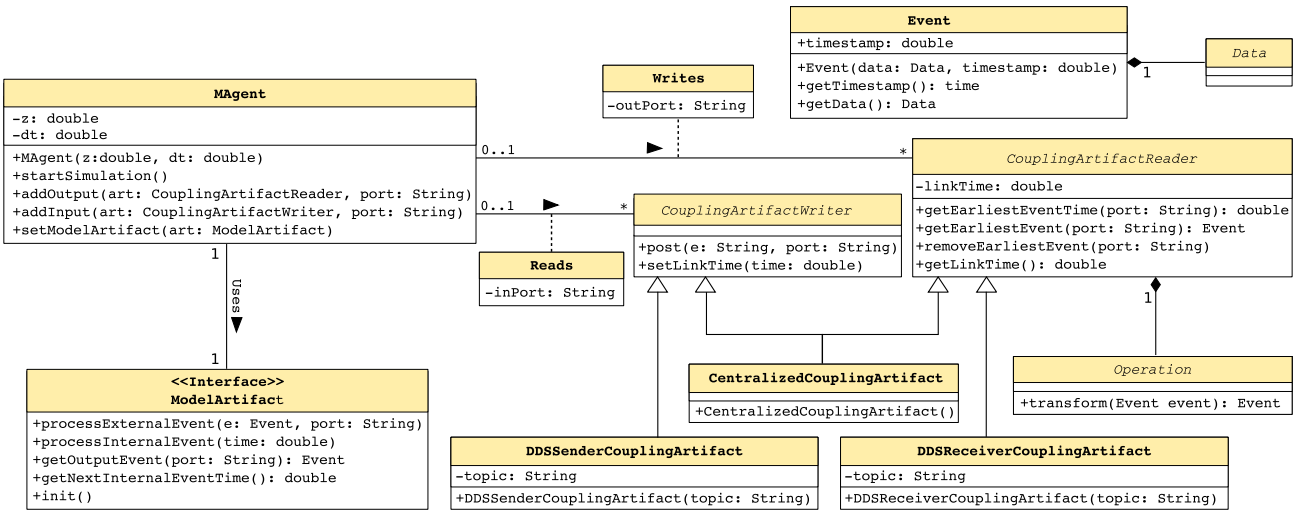


Figure 5: UML description of the MECSYCO software architecture.

their solvers. **Our proposition is then to apply our DEVS wrapping strategy to FMI in order to define a generic way of making continuous equations-based tools interact with discrete-event one.** In the following section, we detail the FMI standard and how we wrap it in DEVS using the hybrid M&S capacity of DEV&DESS and QSS.

5 DEVS wrapping of the FMI standard

5.1 The FMI standard

FMI [47] is a standard of the MODELISAR Consortium and the Modelica Association which proposes a generic software interface for manipulating equation-based models and their solvers. These models may be composed of a mixture of differential, algebraic and discrete-time equations, for instance described with the Modelica language. FMI aims at (1) defining a generic way of exchanging and using models designed with different equation-based simulation tools, and (2) protecting the intellectual property of these models by ensuring that they are seen as black-boxes.

A model implementing the FMI standard is called a Functional Mock-up Unit (FMU). The FMU interface differentiates the output variables whose values are accessible from the outside (thus equivalent to the output ports of the model), from the input variables whose value can be set from the outside (thus equivalent to the input ports of the model). From a software perspective, this interface is composed of a set of C functions, and an XML file. The C functions enable controlling the FMU, whereas the XML file describes the FMU and its interface. More precisely, the XML file describes the variables names, types (i.e. Real/Integer/Boolean/String), variability (constant/discrete/continuous) and causality (input/output/parameter), as well as the continuous states vector.

So far around 42 simulation tools (e.g. Dymola, MATLAB/Simulink, OpenModelica) claim to be compliant with FMI v2.0 (80 with FMI v1.0), including 23 tools officially certified (29 with FMI v1.0)[2]. In order to support the standard, these tools need either (1) to export their models as an FMU or (2) to import existing FMUs and use them as a component in a model. FMI allows two ways of exporting and importing a model: FMI for co-simulation and FMI for model-exchange.

With **FMI for model-exchange**, the model is exported without its solver. The FMU must be then associated with an external solver in order to be simulated. For that purpose, the solver can especially use the following C functions of the FMU API:

- `fmi2GetReal/Integer/Boolean/String` returns the current value of a given output variable.
- `fmi2SetReal/Integer/Boolean/String` sets a specific input variable to a given value.
- `fmi2SetTime` sets the clock of the model to a given simulated time.
- `fmi2GetContinuousStates` returns the continuous state vector.
- `fmi2SetContinuousStates` sets a the continuous state vector.
- `fmi2GetDerivatives` returns the derivative vector of the continuous state.

[2] according to https://www.fmi-standard.org/tools consulted on 07/04/16

- `fmi2CompletedIntegratorStep` indicates that the integration step is finished, and evaluates if internal event has to be processed.

- `fmi2GetEventIndicators` returns indicators of state-events occurrence.

- `fmi2EnterEventMode` enters into the discrete event mode, i.e. makes the discrete-time equations active. While the FMU is in this mode, the integration of the continuous state is stopped but discrete-events (time, state or external) can be processed.

- `fmi2EnterContinuousTimeMode` enters into the continuous-time mode, i.e. disable the discrete-time equations. In this mode, the continuous state of the FMU can be solved, but the discrete state has to remain constant (i.e. events can not be processed).

- `fmi2NewDiscreteStates` evaluates the discrete-time equations (should therefore only be called in event mode) -i.e. processes the potential time and state events. Information returned by this function includes (1) the date of the next time-event (if scheduled), (2) indication if the processed event(s) has changed the continuous state (thus creating a discontinuities in the state trajectory), and (3) indication if the discrete state has to be immediately re-evaluate (e.g. to solve an internal algebraic loop).

With **FMI for co-simulation**, a model is exported with its solver. As this solver has a passive behavior, an FMU for co-simulation is considered as a slave, and proposes in particular the following C functions in order to be controlled by a master algorithm[48]:

- `fmi2DoStep` performs an integration for a given duration.

- `fmi2SetReal/Integer/Boolean/String` set a specific input variable to a given value.

- `fmi2GetReal/Integer/Boolean/String` get the current value of a given output variable.

- `fmi2GetFMUState` and `fmi2SetFMUState` are optional (but essential[49]) functions used to to export/import the model state. They enable therefore to perform a rollback during the simulation of the model.

FMI gives generic guidelines on how a master must manage a set of interconnected FMUs in order to jointly solve their equations: the FMUs executions are synchronized thanks to communication points. These communication points which are shared by all the FMUs, correspond to the points in the simulated time where (1) the FMU simulation must be stopped, and (2) exchanges of data has to be performed between FMUs according to their output-to-input links.

Aside from these guidelines, FMI does not give the specification of the master algorithm. As a consequence, different master algorithms are currently developed like FIDE[50] (FMI Integrated Design Environment) and DACCOSIM[51] (Distributed Architecture for Controlled CO-SIMulation), and numerous issues related to the distributed numerical resolution of the system[49] are still under investigation by the community (e.g. how to determine the best communication points interval during the simulation? how to manage algebraic loop between FMUs?).

Rather than focusing on these distributed numerical resolutions aspects which arise when several FMUs are directly interconnected, we focus in this paper on the hybrid simulation issues which arise when an FMU interact with a discrete-event component (e.g. a NS-3 model). Indeed, in a hybrid context, the communication points simulation strategy of FMI faces the following issues:

- state events occurring between 2 points of communication are localized at the upper communication point, pending improvements of the hybrid co-simulation in the FMI standard.

- new inputs are only taken into account at the next communication point no matter when they are received (the abort orders are only applied at the communication points).

An effort is thus required to integrate the operational software in such a way as to respond to events. For that purpose, we present in the following our DEVS wrapping strategy for FMU. As FMUs for co-simulation and FMUs for model exchange do not have the same API and do not convey the same constraints, we specify a different wrapper for each of them in order to be fully compliant with the standard.

5.2 Wrapping strategies

As any model in MECSYCO, an FMU to integrate will be connected to the co-simulation by a model artifact. This artifact exposes a DEVS view of the FMU, and must allow it to deal with events. To define such a model artifact we can rely on the DEV&DESS formalism as it can be embedded into DEVS, and as it offers a sound framework for describing hybrid systems.

As defined by Zeigler [29], the DEVS version of a DEV&DESS model is composed of three components, each of them being formalized as a DEVS atomic model. With this structure, a DEV&DESS model can be incorporated into a larger DEVS schema as a coupled model. Thus the DEV&DESS model can be simulated using the DEVS

simulation protocol. The three components composing the model are:

- **A continuous component** describing the evolution of the continuous part of the system according to continuous inputs, and producing continuous outputs.
- **An event-detection function** determining when state-events occur based on the continuous states of the model (i.e. the FMU state in our case).
- **A discrete-event component** describing the evolution of the discrete part of the system. This component describes the behavior of the model in the discrete-world, that is to say how it schedules internal events, how it produces and reacts to discrete inputs (i.e. external events), and what are the impacts of state-events. Potentially, for each of these events, the event-based component can change the whole DEV&DESS states, that is to say (1) its own state, (2) the continuous component state (thus creating a discontinuity in the state trajectory) and (3) the event detection function.

The two strategies we propose to wrap FMUs in DEVS using DEV&DESS are the following:

- **FMI for model-exchange** proposes primitives that can handle an hybrid model. However, as stated in Section 5.1, in order to be simulated an FMU for model-exchange needs to be associated with a solver. We need then to implement such an hybrid solver in our DEV&DESS wrapper. In order to manage the continuous state simulation, the original DEVS version of DEV&DESS relies on a quantized integrator approach. The rationale behind this choice is that, quantized integrators have a discrete-event behavior as they quantize the states space instead of discretizing the time dimension. Thus, a quantized integrator naturally bridge the gap between the continuous and the discrete-event worlds [33]: its working principle is already based on the integration of inputs events and on the detection of state-events[30] (i.e. localizing when the state trajectory cross a given threshold). It makes then perfectly sense to keep this choice and to implement a quantized integrator in our wrapper. More precisely, we choose the QSS approach[30] (developed mainly by Kofman) as it offers some of the most advanced mathematical solutions for solving equation-based system, while exhibiting striking simulation performances. We have currently implemented QSS1 [52] (i.e. first order numerical method) and QSS2 [32] (i.e. second order numerical method) solvers for FMU.
- **FMI for co-simulation** embeds a solver but does not include yet the primitives required for managing a discrete-event behavior[53, 49, 50] (e.g. the date of the next scheduled time-event can not be obtained from the FMU). Therefore, we consider that FMI for co-simulation only specifies the continuous behavior of the system. We need then to specifies its discrete behavior (i.e. the equivalent of the DEV&DESS event-detection function and the discrete-event component) within our wrapper. Additionally, specifying the discrete-event behavior outside the FMU enables a more flexible wrapping: different discrete-event behaviors can be associated with a single FMU depending on the co-simulation context (e.g. the discrete-event component can produce a discrete output signal by regularly sampling the continuous output of an FMU, or send events when the continuous output signal of the same FMU reaches a given threshold). Moreover, when wrapping an FMU for co-simulation in DEVS, we have to take account of an additional constraint: the FMU is exported with its **time-stepped** solver as a black box onto which we have a limited control. By its time-stepped nature, an FMU can not be considered as a QSS model, and therefore we need to adapt the original DEVS version of DEV&DESS in our model artifact.

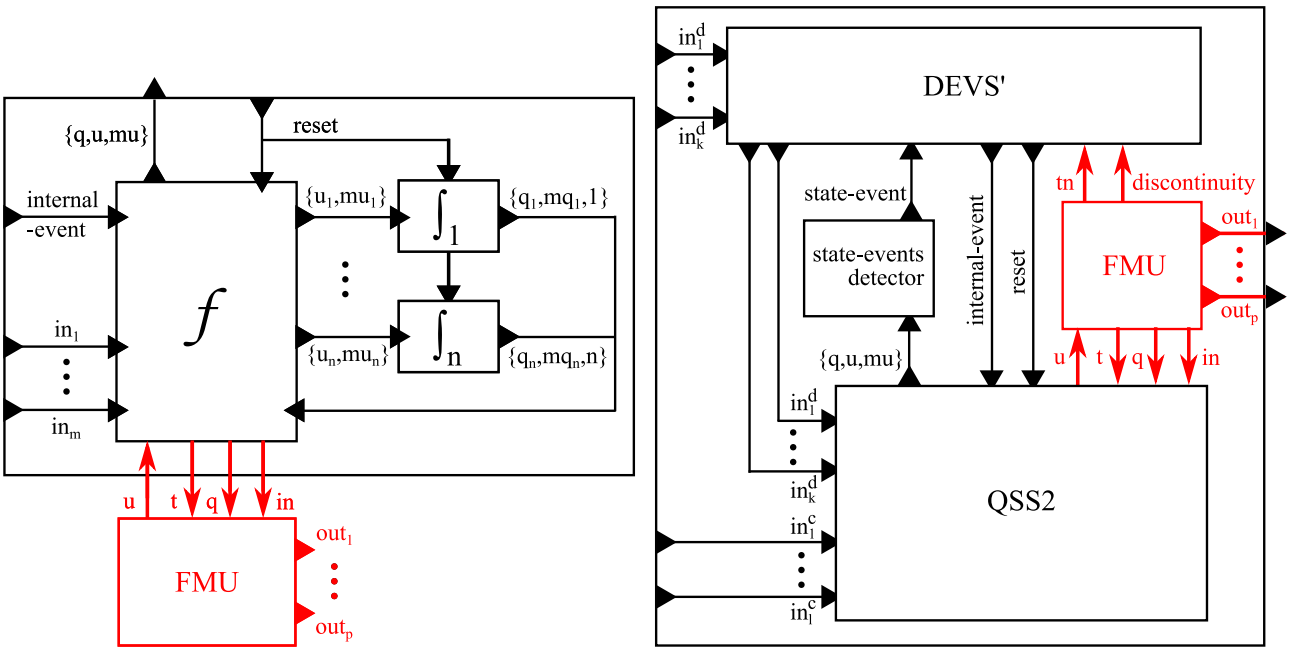
The two next sections detail our wrappers and their validations.

6 Wrapping of FMU for model-exchange

Figure 6a shows the architecture of our QSS2 solver for FMU. This architecture follows mainly the one defined by Kofman, but also has slight differences because two criteria where not handled by the original QSS specifications: (1) due to the FMU nature, the model is clearly separated from its solver, and (2) the discrete-events may cause discontinuities in the continuous state trajectory. In order to highlight these differences, we now describe how our QSS solver works (Section 6.1) and how it interacts with the other components of DEV&DESS: the state-event detector (Section 6.2) and the discrete-event behavior component (Section 6.3). This whole structure of the wrapper is detailed in Figure 6b and corresponds to a DEVS coupled model which is managed by a classic DEVS coordinator not detailed here for sake of concision. This coordinator is directly controlled by the API of the MECSYCO wrapper. Section 6.4 details the validation of the wrapper.

6.1 Continuous behavior simulation with QSS

In the original QSS specifications, the solver interacts with two clearly separated function blocks which respectively



(a) QSS2 solver for FMU model-exchange

(b) global view

Figure 6: bloc diagram view of the DEVS wrapper for FMU model-exchange

define the output and the input behaviors of the model. In our wrapper, these blocks are directly embedded inside the FMU. Therefore the outputs (both discrete and continuous) of the solver correspond to the FMU ones. The output ports of our wrapper coupled model are directly linked to the FMU ones. However, according to the standard, the FMU discrete output ports produce piecewise-continuous signals -i.e. these signals are always present no matter the time instant[50]. In order to generate discrete-event output signals (i.e. signals that are present only at some instants in time) for these discrete ports, we propose an optional mode in our wrapper which filters the output of the FMU in order to generate signals (i.e. external events) only at the moments of the time and/or state-events.

In accordance with the QSS approach, each variable x_i of the FMU continuous state vector is associated with a DEVS quantized integrator $\int_i$. Each integrator $\int_i$ takes in input the first and second derivatives of x_i respectively noted u_i and mu_i, and produces in output the new values and slopes of x_i noted respectively q_i and mq_i. These integrators numerically solve the equation in an asynchronous way. A DEVS atomic model f is in charge of computing the derivatives slopes, handling the inputs of the equation-based system - therefore the model has a set $\{in_1..in_m\}$ of input ports corresponding to the FMU ones- and interacting with the integrators. In the original QSS specifications, the equation-based system is directly embedded into f. This is not feasible in our case cause the system is already embedded in an FMU. As a consequence, in our solver f also manages the interaction with the FMU in the following way:

- when it has to update the FMU continuous state (e.g. when it receives from an integrator a new value and slope for a continuous state variable), f first switches the FMU into the continuous mode (using `fmi2EnterContinuousTimeMode`) if it was not already, and call the `fmi2SetContinuousStates` function.

- when it has to update the value of an input variable of the FMU (i.e. when it receives an input event through a in_i ports), f first checks the variability of the variable into the XML description file. Depending if this variability is continuous or discrete, f calls the `fmi2EnterContinuousTimeMode` or `fmi2EnterEventMode` function in order to set the FMU in the appropriate mode (if it was not already). f checks then the input variable type in the XML file, and updates its value in the FMU using `fmi2SetReal/Integer/Boolean/String` function. If the updated variable is discrete, f asks (several times if needed by the FMU) the FMU to re-evaluate its discrete state using `fmi2NewDiscreteStates`.

- when it receives any events at its input ports (e.g. from the integrators or at a in_i port), f updates the clock

of the FMU to the timestamp of the events using the `fmi2SetTime` function.

- when it has to get the derivative u_i (e.g. in order to compute its slope mu_i and to forward these two values to $\int_i$), f uses the `fmi2GetDerivatives` function of the FMU.

As shown in Figure 6b the solver interacts with two atomic models in order to simulate the discrete behavior of the FMU. These models correspond to the ones defined by Zeigler in the DEVS version of DEV&DESS.

6.2 State-event detector

The **state-event detector** atomic model is in charge of the accurate localization of state-events during the simulation of the continuous equations. In order to take advantages of the QSS approach for detecting state-events, we make the hypothesis that the state-event thresholds of the FMU are *a priori* known (either because this information can be obtained from the model designer or from the XML description file). In the original hybrid QSS specifications[33], Kofman suggests two ways of feeding the state-event detector from the QSS solver:

1. It can receive the variables values q and derivatives u and mu. This way, as stated by Kofman, the detector only *"have to find the roots of a second degree polynomial"*[33] in order to find the time of the next state-event (in the absence of new state and derivatives update received from the solver). The detector schedules then an internal event at the time of this state-event in order to produce an output notifying the occurrence of the event.
2. Or it can only receive the derivatives u and their slopes mu directly from the output ports of f. In this case, in addition to find the time of the next state-event and schedule the resulting internal event, the detector has to integrate (in parallel of the system resolution) the variables concerned with the thresholds.

Kofman opts for the second option as it does not implies any modification of the QSS solver. However, the drawback of this option is that the detector can not be aware of the discontinuities in the continuous state trajectory caused by discrete-events processing (time, state or external). That is why we choose the first option in our wrapper: the model f forward immediately to the detector all the updates of the continuous states vector q and its derivatives u and mu through a dedicated output port.

6.3 Discrete-event behavior simulator

The **DEVS'** atomic model is in charge of managing the occurrences of discrete events (state, time and external). After each modification of the discrete state of the FMU -i.e. after each external/time/state-event processing in the FMU-, this component (1) retrieves the time tn of the next time-event scheduled in the FMU, and (2) checks if the event processing has created a discontinuity in the continuous state trajectory (by checking the information returned by the last called of the FMU `fmi2NewDiscreteStates` function). The DEVS' component schedules an internal event at each tn. It also receives notifications of state-events occurrences from the detector. Moreover, all the discrete inputs of the FMU are first sent to the DEVS' component before being immediately forwarded to the QSS solver. This enables the DEVS' component to be aware of discrete-input occurrences, and thus to interact with the FMU (i.e. update tn and check discontinuities) after the discrete input was processed by the solver. Therefore as shown in Figure 6b, we distinguish in the QSS solver interface between:

- the set $\{in_1^c, ..., in_k^c\}$ of input ports which correspond to the continuous inputs of the FMU. These ports are directly connected to the input ports of the wrapper. This way, the solver can directly receive continuous inputs of the FMU from the other simulation tools of the co-simulation.
- the set $\{in_1^d, ..., in_l^d\}$ of input ports which correspond to the discrete inputs of the FMU. These ports are duplicated in the DEVS' component interface.

As soon as it computes a time event or it receives a state-event notification, the DEVS' component sends an internal event notification to the QSS solver through a dedicated port. The solver processes this notification in the same way it does with discrete inputs: it sets the FMU to the discrete mode and asks the FMU to re-evaluate its discrete state, thus causing the time/state-event to be processed. The only difference is that, as no discrete input of the FMU corresponds to this internal event notification port, the solver does not change any input variable of the FMU. Finally, as soon as the DEVS' component detects a discontinuity in the continuous state trajectory, it sends immediately a reset event to the QSS solver through a dedicated port. In accordance with Zeigler's DEV&DESS specifications, this event resets both the quantized integrators and the f model state, thus enabling the QSS solver to handle the discontinuity.

6.4 Implementation and validation

We have implemented this wrapper in the Java version of MECSYCO. In order to interact with the FMU, we rely on JavaFMI[54]. As this library only covers FMU for co-simulation, we proposed an extension to interact with FMU for model-exchange. We have verified the behavior of our wrapper by reproducing two QSS2 use-cases proposed by

Kofman[33]. The first one corresponds to an DC-AC inverter circuit equipped with switches controlled by discrete-inputs which are sent according to a Pulse Width Modulation (PWM) strategy. The second one corresponds to a ball bouncing downstairs with state-events occurring twice each bounce (one when the ball hits the ground and one when it leaves it). We translated the Kofman's models into the Modelica language (Figures 7a and 8a) and export them in FMUs for model-exchange using OpenModelica. We found similar simulation results (Figures 7b and 8b) and performances with our solver and with the Kofman one.

As these two models do not include discontinuities in the continuous state trajectory, we also propose another use case to test this aspect with our solver (Figure 9a). This use case correspond to the simulation of a a barrel-filler factory inspired by the one proposed by Praehofer[14]. In this factory, we consider a queue of barrels waiting on a conveyor to be filled. The factory fills only one barrel at a time. As soon as the water reaches a given level in the barrel, the barrel is carried away by the conveyor, and the filling process starts again for the next empty barrel. A tank stores the water to fill the barrels. The flow rate of water filling the barrel decreases with the level of water in the tank. A valve controls the flow of water between the tank and the barrel. The valve can only be in two states "open" (water goes from the tank to the barrel) or "close" (the filling process is stopped). The continuous dynamics of the model correspond to the levels of water in the current barrel and in the tank. The model receives discrete inputs controlling the valve. State-events correspond to the moment where the current barrel is full. At this point, the level of water in the current barrel is reset to represent the change of barrel. The model produces a discrete output signal corresponding to a regular sampling of the level of water in the barrel. This signal can be for instance sent to a controller for monitoring the filling process. We found similar results when simulating this model with our QSS2 solver (Figure 9b) and with the OpenModelica solvers.

7 Wrapping of FMU for co-simulation

As stated in Section 5.2, we need the three components of DEV&DESS to integrate a FMU into DEVS. An FMU for co-simulation provides the continuous behavior and we need to define the two remaining components (i.e. the state-events detector and the discrete-behavior component) in the wrapper. These components are dependent of the wrapping context:

- the discrete-behavior component has to specify the behavior of the FMU in the discrete world. This component corresponds to a DEVS atomic model which can interact with the FMU component. For example, this component can sample a continuous output of the FMU by regularly scheduling internal event, and producing external output event according to the current value of the FMU variable using `fmi2GetReal`.
- the state-events detector has to specify the condition of occurrence of state-events according to the FMU state. This detector corresponds to a boolean function $S \rightarrow \{true, false\}$ with S the set of the FMU states. For example, this function can return true (i.e. a state-event occurs) only if a variable of the FMU is superior or equals to a given value.

In the following we detail how we implement the main DEVS primitives into the wrapper.

7.1 Time of the next internal event

In our DEVS wrapper for FMU for co-simulation[55], we rely on the FMI specifications to simulate the continuous output of the component: we consider that the FMU produces outputs at a sequence of pre-defined communication points. From our DEVS point of view, these communication points are seen as internal events producing external output events. In the same way, from our DEVS point of view we see updates of the continuous input values received by the FMU as input events.

According to the DEVS semantic, the `getNextInternalEventTime()` function must return the date of the earliest scheduled internal event in the model. In the DEV&DESS context, this date corresponds to the minimum between:

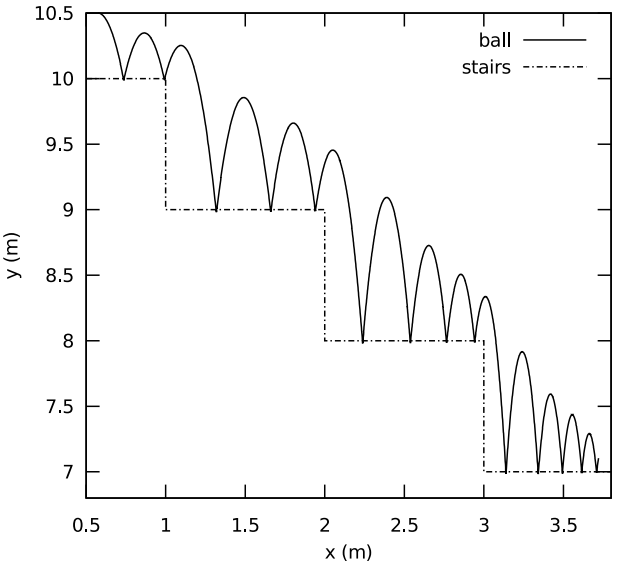
- the date of the next internal event scheduled in the discrete-event component,
- the date of the next communication point of the FMU,
- the date of the next state-event.

Getting the first two dates is trivial as they are *a priori* known. Things get more complex for the state-events: because of the numerical resolution of the equational model, state-events can only be detected *after* each integration step of the FMU, and their localization in time can only be approximated.

In order to get the date of the next state-event, we need to perform an exploration with the FMU to see if a state-event will occur before its next communication point. Thus, the component will always be "in the future" compared to the current simulation time. As according to the DEVS semantic the `getNextInternalEventTime()` function must not change the state of the model, it is imperative to be able to come back to the previous state of the FMU which is the only legitimate state from the simulation point of view.

```
model BouncingBall
  output Real x(start = 0.575); "horizontal position (m)"
  output Real y(start = 10.5); "vertical position (m)"
  output Real vx (start = 0.5); "horizontal speed (m/s)"
  output Real vy(start = 0); "vertical speed (m/s)"
  discrete Integer sw(start=0); "discrete position"
  parameter Real k = 100000;
  parameter Real m = 1;
  parameter Real b = 30;
  parameter Real ba = 0.1;
  parameter Real g = 9.80665; "gravity (m/s^2)"
  parameter Real h = 10; "first step height (m)"
  equation
    der(x) = vx;
    der(y) = vy;
    der(vy)=-g-ba*vy/m-sw*(b*vy/m+k/m*(y-floor(h+1-x)));
    der(vx)=-ba/m*vx;
    when y<=floor(h+1-x) and pre(sw)==0 then
      sw= 1;
    elsewhen y>=floor(h+1-x) then
      sw= 0;
    end when;
end BouncingBall;
```

(a) Modelica code of the model

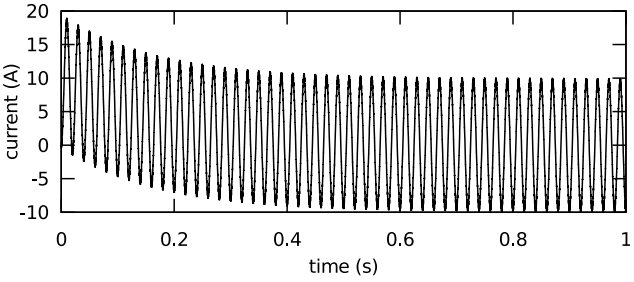


(b) MECSYCO simulation results

Figure 7: Simulation of the bouncing ball system

```
model inverter-circuit
  parameter Real R = 0.6 "resistance (ohm)";
  parameter Real L = 0.1 "inductance (H)";
  parameter Real Vin = 300; "input voltage (V)"
  Real iL(start = 0) "current (A)";
  discrete input Integer sw(start = -1) "switch";
  equation
    der(iL) = (-R / L) * iL + sw * Vin;
end inverter-circuit;
```

(a) Modelica code of the model

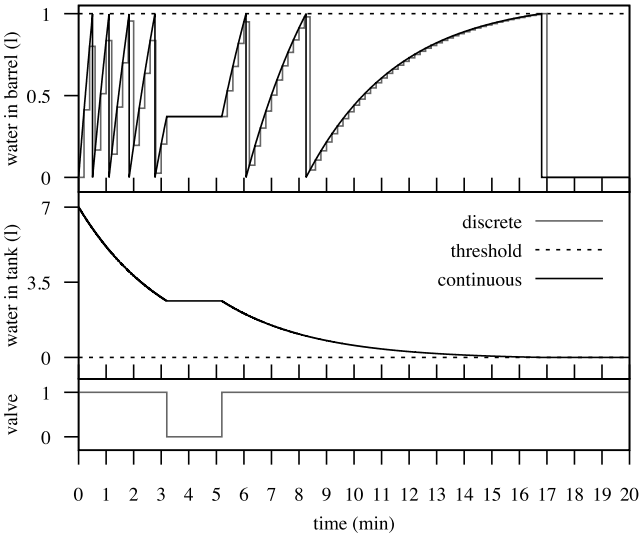


(b) MECSYCO simulation results

Figure 8: Simulation of the DC-AC inverter circuit

```
model barrel
  discrete input Boolean valve(start = true);
  parameter Real qmax = 7 "initial tank water (L)";
  Real q(start = qmax) "water in the tank (L)";
  Real flow(start = 0) "flow of water";
  parameter Real gain = 0.3;
  parameter Real value = 0.025;
  output Real x(start = 0) "water in the barrel";
  parameter Real xmax = 1 "wanted barrel water(L)";
  parameter Real f = 10 "sampling frequency";
  discrete output Real y(start = 0) "output sampling";
  equation
    flow = if valve and q > 0 then q*gain+value else 0;
    der(q) = -flow;
    der(x) = flow;
    when x >= xmax then
      reinit(x, 0);
    end when;
    when sample(0, 1 / f) then
      y = x;
    end when;
end barrel;
```

(a) Modelica code of the model



(b) MECSYCO simulation results

Figure 9: Simulation of the barrel-filler factory

The rollback capability of the FMU assures this feature as long as no new integration step is performed.

When a state event is detected during an exploration, we perform a bisectional search [17, 56] in order to localize the state-event as precisely as possible in the time. This search is formalized by the Algorithm 2 which, given the initial integration step ΔT and a number of iterations m (formalizing the search precision), positions the FMU as close as possible to the state-event occurrence. The algorithm basically progresses by a succession of integration steps whose duration δt is adapted according to state-event occurrences, and following a dichotomous strategy. As, again, the original state must always be accessible, and as only one integration step can be canceled at a time, the algorithm always goes back to the legitimate state before performing a new integration step.

Algorithm 2 Bisectional search for state-event localization.

INPUT: $\Delta T \in \mathbb{R}_0^+, m \in \mathbb{N}_0^+$
  $\delta t \leftarrow 0$
  $\Delta t \leftarrow \Delta T$
  for 1 **to** m **do**
    `fmu.rollBack()`
    $\Delta t \leftarrow \Delta t/2$
    `fmu.doStep(`$\delta t + \Delta t$`)`
    if $\neg$`stateEventOccurence()` **then**
      $\delta t \leftarrow \delta t + \Delta t$
    end if
  end for

7.2 Events processing

According to the DEV&DESS semantics, when an event (internal, external or state-event) occurs at simulated time t, the equational component describes the continuous evolution of the system until t, and the event is processed by the discrete-event component. This behavior is translated in our model artifact as follows.

When the `processExternalEvent(`e_{in_i}, t, x_i^k`)` function is called to report the occurrence of an external input event e_{in_i} into the x_i^k input port, the first step consists in rolling back the FMU to its previous state, which is, as stated in the previous section, the only legitimate state from the simulation point of view. Then the FMU performs an integration step until t in order to reach the point where the event occurs. Finally, if x_i^k is a continuous port, the FMU is parametrized accordingly. If x_i^k is a discrete port, the external transition function of the discrete-event component is triggered in order to process e_{in_i}.

In a similar way, when the `processInternalEvent(`t`)` function is called to process the next internal event, the FMU is rolled back to its previous state and an integration step is performed until t. If the next internal event corresponds to a communication point of the FMU, then the model artifact retrieves the continuous output ports values, and produces the external output events accordingly. On the other hand, if the next internal event corresponds to a state-event or the next internal event of the discrete-event component, then the internal transition function of this latter is called, which could produce external output events.

8 Discussion and related works

We have presented in Section 4 the whole specification of the MECSYCO middleware which enables the rigorous co-simulation of complex systems. On the contrary to the mosaik co-simulation platforms [57] which is based on discrete time-step framework, MECSYCO enables the rigorous integration of models written in heterogeneous formalisms. Indeed, on this point MECSYCO relies on the formal guarantees offered by DEVS, and on the practical guidelines offered by the numerous integrative works around DEVS in the literature.

The High Level Architecture (HLA) standard [58] gives generic guidelines and rules for building a co-simulation middleware. It as been shown that an HLA-compliant middleware, can also used a DEVS framework [59, 60]. However, on the contrary to HLA, we gives here the whole specification of our platform. These specifications range from the decentralized synchronization algorithm to the distributed agent-based architecture and its object oriented implementation. Thus, we clearly define our middleware working principle, making our co-simulations more reproducible and our middleware more flexible: different implementations of MECSYCO are interoperable and therefore can be simultaneously used in a co-simulation, as shown with our Java and C++ versions.

Finally on the contrary of the DACCOSIM [51] platform dedicated to the co-simulation of FMU components, MECSYCO is not limited to a specific simulation software or norm.

Using our MECSYCO platform, we have shown how FMU components can be wrapped into a DEVS model using DEV&DESS and QSS. We propose then a generic way of making continuous models exported in the FMI standard interacting with discrete models. It is important to note that, as we base this wrapping directly on the DEVS protocol, this work is not limited to the MECSYCO platform, but can be implemented in any DEVS-based platform. With our DEVS wrapping of FMU for model-exchange, we define a hybrid QSS solver tailored to the FMI standard. Like other existing QSS versions of the litterature [61, 62], our solver can simulate Modelica models. However, the originality is that our QSS solver can also solves models written in any

of the numerous software compliant with FMI for model-exchange (e.g. MATLAB/Simulink).

We also want to underline the fact that, whereas our wrapping of FMU for model exchange is adapted both for FMI 1.0 and 2.0 versions, our wrapping of FMU for co-simulation is only adapted for FMI 2.0. This is due to the fact that we rely on the roll-back capacity of the FMU which is only available in the latest version of the standard.

In the following section, we show the features of our solutions through a proof of concept of a smart heating M&S.

9 Use Case

Our use case is inspired by different works around smart-heating [62][63]. We want to simulate the evolution of the temperature inside a system composed of two buildings equipped with electric heaters, and the power consumption of these latter. Using this simulation, we are interesting in the design of a controller for limiting the consumption peaks duration in the building. To do so, this controller temporarily disables some heaters according to the information it receives on the buildings temperatures and power consumption. This controller interacts with the buildings system through an IP telecommunication network. Such a goal could lead to an typical iterative M&S process driven by the following series of questions:

1. What are the power consumption and the temperatures evolution in the buildings without the controller?
2. Does the controller actually achieve its goal without the delays and the interference induced by the telecommunication network?
3. What are the influences of the telecommunication network on the controller performances?

This leads to three major steps in the M&S process.

In order to answer to the first question, we need to simulate the thermic system. We use three models. One describes the outside temperature evolution. One model describes the power consumption and temperature evolution of one building according to the outside temperature evolution. We perform the co-simulation of these three models by feeding the building models with the outside temperature trajectory.

In order to answer the second question, we build the model of the controller. We use this model twice (one for each building) in the co-simulation. Each controller model is fed with the outputs of its building model (i.e. rooms temperatures and power consumption). When needed, it produces as output the heaters switch off/on orders which are sent to the building models as inputs.

In order to answer the last question, we add a model of the telecommunication network between the buildings and their controller. The outputs of the buildings models now first pass through the network model before arriving to the controller models. Reciprocally, the controller orders transit through the network model before delivery. The network model adds then delays and perturbation (i.e. packet loss and noise) to the system.

This "toy" use case does not claim to be realistic. We keep the individual models of the use case simple since we are here more focused on demonstrating the following MECSYCO properties rather than on presenting a credible use case:

- **Modularity:** the use case development follows an iterative M&S process. We first begin the co-simulation with the thermic model of the building. Then, we add step by step the models of the controller and the telecommunication network. We show that passing from one of these steps to another does not require to rebuild the co-simulation from scratch.
- **Software interoperability management:** each model of the co-simulation is implemented in a different simulation software. The thermic model is defined in Modelica and exported into FMUs for model-exchange and co-simulation, the telecommunication model is defined using the NS-3 simulator, and the controller model is implemented in an ad-hoc way using the Java language. We show that MECSYCO properly manages the exchange of data between these heterogeneous software.
- **Multi-formalism integration:** the models of the co-simulation are defined in different formalisms. The thermic model is an hybrid model composed of differential and discrete equations. The telecommunication model is a discrete event model whereas the controller model is a discrete time-stepped model. We show that MECSYCO enables the rigorous integration of these heterogeneous models.
- **Multi-representation integration:** the models evolve at different temporal scales, namely the seconds for the controller and the thermic models and the nanoseconds for the telecommunication network. We show that MECSYCO rigorously synchronizes these models executions during the co-simulation.
- **Distributed multi-platform execution:** we execute the co-simulation on two computers connected on a LAN. These two computers uses different operating systems, and different implementations of MECSYCO. The telecommunication network model is executed on GNU/Linux Debian with the C++ version of MECSYCO, whereas the other models are executed on Microsoft Windows 10 with the Java version of MECSYCO.

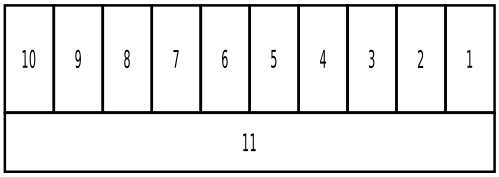


Figure 10: Architecture of the building

In order to make this use case reproducible and describe in a transparent way all its heterogeneity, we detail in the following sections, each model and its implementation. Finally, in Section 9.4 we describe the different co-simulations made with these models, we discuss the simulation results, and highlight the benefits offered by MECSYCO.

9.1 The thermic system models

We create two kinds of models for the thermic system. One corresponds to the outside temperature trajectory. For sake of simplicity, this model generates a simple sinusoidal signal representing day/night temperature cycles.

The second model corresponds to the temperature and power consumption evolution of one building. Recall that, as the two buildings are identical, we use this model twice. Each building of the thermic system is composed of ten offices linked by a corridor following the Figure 10. Each room is influenced by the outdoor temperature, by the adjacent rooms, and contains an electric heater with an internal thermostat which turns on when the temperature inside the room falls under a minimal value and turns off when this temperature reaches a maximum value. Then we need to build models for rooms to get the temperature, for walls to get the heat flow between two rooms (or between a room and the outside temperature) and for electric heaters to get the instantaneous power consumed to heat. For sake of simplicity, we assume here that all the heaters have the same features (i.e. setpoint temperature, power and tolerance).

For these models, we choose to use Modelica [64] which is an object oriented language adapted to the modeling and simulation of hybrid systems. We use the standard library of Modelica to build our models. The thermal part of the building is built using the `Modelica.Thermal.HeatTransfer` library and the electric heater model is built with the `Modelica.Electrical.Analog` library.

The following list presents the interface of the building model used to interact with the other models of our use case.

Inputs:

- $blackout_i$ is a discrete boolean input. When set to true, the electric heater of the room i is shut down.
- T_{out} is the continuous outside temperature in K.

Outputs:

- R_iTemp is a discrete signal sampling the temperature of the room i. This signal is updated every $period$ of time, and represents the information sent regularly by a thermometer to the controller.
- R_iPow is the instantaneous power consumption inside the room i. It is a discrete variable updated each time the heater starts and stops.

Rooms are modeled as heat capacitors. Each room is seen as a volume of air with a temperature. The different influences (from the walls and from its heater) are modeled as heat flow exchanges. The behavior of the model of a room i is characterized by the equation:

$$C_i * \frac{dT_i}{dt} = Q_{in_i} + Q_{heater_i}$$

Where:

- C_i is the constant thermal capacity of the room in J/K.
- T_i is the temperature of the room in K.
- Q_{in_i} is the sum of the heat flows received from the walls connected to the room.
- Q_{heater_i} is the heat flow received from the electric heater. We consider here that it is equal to the instantaneous power consumption of the room in W -i.e. $R_iPow = Q_{heater_i}$.

The heat flows are computed in the following way. The model of the wall determines the heat flows between the two air volumes k and l it is connected with. Note that in our case, an air volume can be a room or the outside environment. The heat flows depends on the temperatures of k and l as well as on the thermal conductance of the wall. This is represented by the following equations:

$$Q_{kl} = G_i * (T_k - T_l)$$
$$Q_{lk} = -Q_{kl}$$

Where:

- G_{kl} is the constant thermal conductance of the wall in J/K.
- Q_{kl} (resp. Q_{lk}) is the heat flow from the volume k (resp. l) to the volume l (resp. k) in J.

Q_{heater_i} is determined by the behavior of the electric heater which is modeled as a basic electrical circuit with a

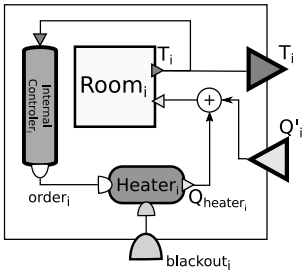


Figure 11: Heated room model "R_i"

constant voltage, an electrical resistance and a switch. This is represented by the following equation:

$$\begin{aligned}&\textbf{if } order_i \textbf{ and not } blackout_i\\ &\textbf{then } Q_{heater_i} = \frac{U^2}{R}\\ &\textbf{else } Q_{heater_i} = 0\end{aligned}$$

Where:

- U is the constant voltage in V.
- R in Ω is the constant electrical resistance of each heater in the building.
- $order_i$ is a boolean representing the command of the internal controller of the heater. When it is equals to true, the heater is on.

$order_i$ is set to true when the temperature inside the room is below a minimal value, and to false when this temperature reaches a maximal value. This behavior corresponds to the conditional statement:

$$\begin{aligned}&\textbf{when } T_i \leq T_{wanted} - \frac{bandwidth}{2}\\ &\quad\textbf{then } order_i = \textbf{true}\\ &\textbf{else when } T_i \geq T_{wanted} + \frac{bandwidth}{2}\\ &\quad\textbf{then } order_i = \textbf{false}\end{aligned}$$

Where:

- T_{wanted} is the desired temperature in every room of the building.
- $bandwidth_i$ is the temperature tolerance of every heater in the building.

Each discrete port R_iTemp samples the continuous temperature evolution of the room i according to the following Modelica code:

$$\begin{aligned}&\textbf{when } sample(0, period) \textbf{ then}\\ &\quad R_i\,Temp\ = T_i\\ &\textbf{end when};\end{aligned}$$

Where $period$ is a constant interval of time in s. The Modelica function $sample(0, period)$ is used to update $R_i\,Temp$ each $period$ of time in order to represent the discrete signal regularly sent by the thermometers to the controller.

The model of a room with its heater and controller can be described in bloc diagram by Figure 11. Using this model, the whole building can be described by the bloc diagram of Figure 12. According to OpenModelica, this model is composed of 1622 equations including 11 differential equations.

The building model is an hybrid system which combines continuous and discrete behavior. The simulation of this model requires to solve the differential equations system describing the temperatures evolution while taking account of discrete time and state-events. These discrete events correspond to the Modelica "when" statements. Each update of the discrete output ports R_iTemp corresponds to a time-event scheduled in advance by the model for regularly sampling the continuous temperature evolution. At the opposite, a state-event occurs each time the temperature of a room reaches one of the two heaters thresholds -i.e. each time $T_i = T_{wanted} \pm \frac{bandwidth}{2}$. Considering the 11 rooms of the building, 22 state-event thresholds have then to be simultaneously monitored. Moreover, the continuous inputs of the outside temperature and the discrete inputs corresponding to the blackout orders of the controller have to be integrated during the simulation.

9.2 The controller model

As the two buildings do not interact together (i.e. no temperature exchange occurs between the buildings), the controller can manage each building separately. Then, we define here the model of the controller for managing only one building. This model can be duplicated in order to control both buildings.

Recall that the goal of the controller is to limit power consumption peaks duration in the building. To do so, the controller temporary disables some heaters when the total power consumption of the buildings is equal or higher than a given threshold Pow_{max}. Hence we accept to lower the temperatures of some rooms beneath the setpoint for a specific period of time. Nevertheless, in order to maintain a minimum of comfort in every room, the controller makes sure that the temperature is above a given threshold $Temp_{min}$ in K (assumed to be lower than the temperature setpoint of the heaters).

The controller maintains a set of variables $Temp_i$ and Pow_i for saving respectively the last temperature and the last instantaneous power consumption values received from the sensors of each room i of the building. Basing on these variables, the controller regularly evaluates at a given frequency if some heaters need to be disabled or enabled. If so, it sends the corresponding orders to the heaters.

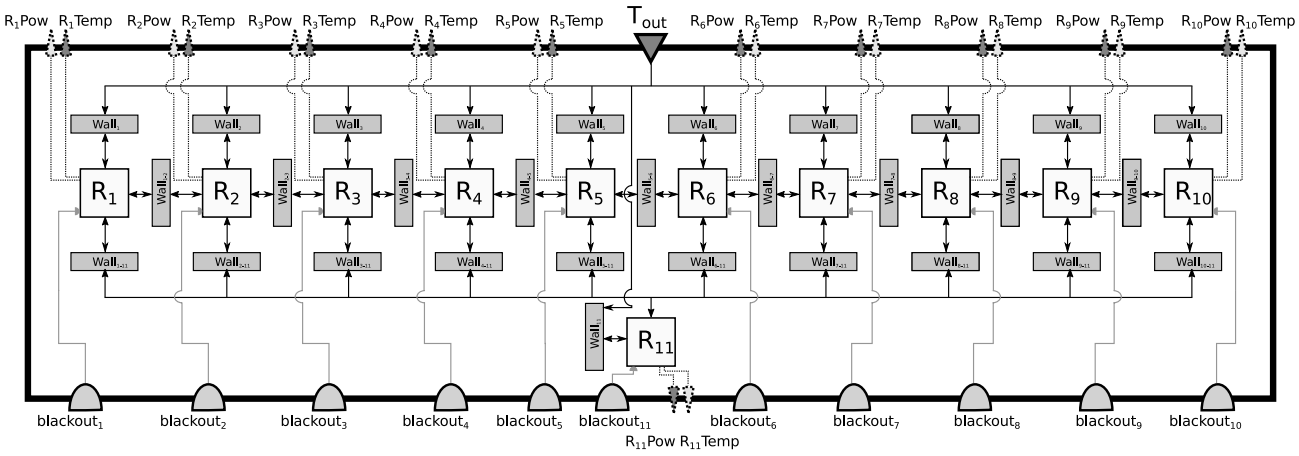


Figure 12: Building model

The policy used to determine these orders at each evaluation point is the following:

1. The controller checks for each room i if $Temp_i \leq Temp_{min}$. If so, the controller immediately enables the corresponding heaters.

2. In order to check if some heaters have to be shut down, the controller computes the building total instantaneous power consumption Pow_{tot} according to the following equation:

$$Pow_{tot} = \left(\sum_{i=1}^{11} Pow_i \right) + \frac{U^2}{R} * N_{on}$$

With:

- N_{on} the number of heaters that have just been enabled by the controller in step 1.
- U the constant voltage of the heaters in V.
- R the constant electrical resistance of the heaters in Ω.

If $Pow_{tot} \geq Pow_{max}$, then the controller computes the number $N_{off} \in \mathbb{N}$ of heaters which have to be shut down in order to lower Pow_{tot} below Pow_{max}. The controller disables then the heaters of the N_{off} rooms having the highest temperatures. N_{off} is computed according to the following equation:

$$N_{off} = \mathsf{int} \left(\frac{Pow_{tot} - Pow_{max}}{U^2/R} \right) + 1$$

With $int : \mathbb{R} \rightarrow \mathbb{N}$ the integer typecasting function which truncates a decimal number to zero digits.

This controller is described by a discrete time-stepped model where each time-step corresponds to an evaluation point. Each Pow_i and $Temp_i$ variable can be updated by specific input ports of the model. The controller order to the heater of each room i corresponds to a boolean sent through a specific output port $blackout_i$. From our DEVS wrapping perspective, we consider each time-step as an internal event and each input/output as an external event.

9.3 The telecommunication network model

The IP network is modeled with NS-3 [22], a popular discrete-event IP network simulator. NS-3 models can be wrapped into DEVS as a coupled model composed of network components [46].

From the perspective of the IP network, each room corresponds to two network devices. A heater sends information about its power consumption to the controller, and receive commands from this latter, asking them to stop heating for a moment. A thermometer regularly sends the current temperature of the room, to the controller too.

The IP network topology is shown on Figure 13 (with only three rooms a building instead of eleven). We consider that there is one switch (S) a building, connecting all heaters and thermometers in a same local area network. Then, each building is connected to the Internet with its own router (R). The Internet is just modeled with one big central router, and the controller is itself connected to it. Network devices are connected to external models through input and output ports (marked on the sides of the figure), for receiving and transmitting data. In this case, external models correspond to the application layer of the devices.

Heaters and thermometers can exchange measures and commands with the controller over the fake Internet, thanks to TCP or UDP connections, depending on the choice of the experimenter. Choosing TCP (reliable protocol) or UDP (unreliable protocol) is important due to the error model installed on the links between the building routers and the

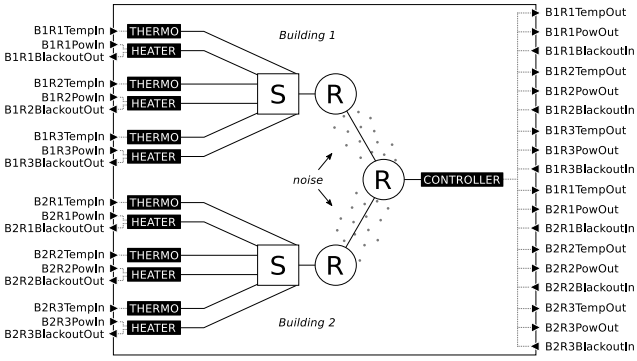


Figure 13: IP network topology, with three rooms a building, and with DEVS ports on the sides.

Internet, used for modeling some noise on the network. Experimenters can configure this error model, choosing a bit error rate (e.g. one incorrect bit for every thousand bits sent). The network model is build using the standard NS-3 component library.

9.4 Co-simulations

This section details the co-simulations and their results. The parameters used in the different co-simulations are provided in Table 3.

In order to answer to the first question, we export the thermic building model into an FMU for model-exchange to handle discrete-events. As said previously, we use two instances of this model, one for each building we want to simulate. We export the outside temperature model as an FMU for co-simulation called Out.

According to our wrapping strategy, each building FMU is associated with an instance of our QSS solver. Each of these QSS solves the 11 differential equations of its models and monitors its 22 state-event thresholds. We set the quantization of all the integrators of the solvers to 0.0001. As shown in Figure 14b, we interconnected the wrapped models in MECSYCO in order to form the DEVS coupled model of Figure 14a.

The co-simulation is executed on a single computer using Windows 10 and the Java implementation of MECSYCO. We simulate one day of the system evolution.

The co-simulation results are shown in Figure 17a. For the sake of concision, these results solely show the state trajectory of the first building. These results are similar to the ones obtained with OpenModelica, and match perfectly the expectation: state-events are handled at the right times (i.e. the heaters start and stop just when the temperatures evolution reach one of the two thresholds), and we can see the influences of the building symmetry with room 1 to 10 in the state trajectory (e.g. the rooms 1 and 10, or the rooms 5 and 6 which receive similar thermal influences have similar trajectories).

In order to answer the second question, we add the two controller models (one for each building) to the co-simulation. According to the co-simulation parameters, the controller considers that consumption peaks occurs when the total power consumption of the building is higher than the power consumption of one active heater (i.e. when at least two heaters are active at the same time). We configure the models in order the controller to evaluate the building states every minutes. 30 seconds after each evaluation point (and its potential orders sent to the heaters) the controllers will receive new information from the buildings sensors, and wait another 30 seconds until the next evaluation points. We connect the wrapped model in order to form the coupled model of Figure 15a.

The Figure 17b shows the simulation results for the first building. In this graph, grey areas represent the periods of time during which the heaters should be shut down according to the controller model outputs. Again, the simulation results are in accordance with the model. Indeed, we can see that the controller model outputs are well integrated into the building model: when the controller sends the shut down orders, the heaters immediately stop working and the corresponding rooms temperatures start decreasing according to the walls heat transfers. On the contrary, as soon as the controller sends starting orders to the heaters, the corresponding rooms temperatures immediately start increasing and oscillate as expected between the two state-event thresholds.

In order to answer the last question, we add the telecommunication model to the co-simulation as indicated by the Figure 16. As NS-3 works at a nanosecond timescale whereas the FMUs use a second time scale, we use transformation operations in the coupling artifacts between NS-3 and the FMUs in order to convert the timestamps of the exchanged events.

It is important to note that the models are compliant with different OS: the FMUs components we have generated are only compliant with Microsoft Windows, whereas the NS-3 model requires a GNU/Linux. Moreover, we used different implementation of MECSYCO to wrap our models: the FMU components and the controller model are wrapped in the Java version whereas the NS-3 model is wrapped using the C++ version. As a consequence, we have to distribute the co-simulation on two computers:

- The first computer runs on Windows 10 and uses the Java version of MECSYCO to simulate the FMUs and the controller model.
- The second computer runs on GNU/Linux Debian and

uses the C++ version of MECSYCO to simulate the NS-3 model.

When we configure NS-3 in order to simulate a TCP protocol on the network without any error model, the simulation results are similar to the previous ones (shown on Figure 17b) -i.e. the network does not impact the system behavior. This is due to the fact that, in this case, the network only introduces very small delays (on a second time scale) in the communications between the buildings and the controller. However, when we configure NS-3 for the network model to introduce perturbations in the communications (i.e. packets losses or corruptions), the simulation results are changed as shown by Figures 17c and 17d. To do this, we use in NS-3 an UDP protocol without checksum and an error model of 1 bit altered respectively every 10000 ones and every 1000 ones. We can see that, as one can expect, the more noise we add in the network, the more different the system trajectory becomes. It is interesting to note that in the results shown by Figure 17d the noise is so high that some controllers orders (for instance the shutdown order for the heater of room 1 at time 8370) do not even arrive to the buildings. Note that the Figures 17c and 17d only display an example of simulation results as the NS-3 error model introduces a stochastic process.

9.5 Synthesis

With this use case, we have shown that MECSYCO can rigorously integrate different kinds of heterogeneity. At each step of this use case we introduced a new heterogeneity at the software, formalism and representation levels.

- The first step shows that MECSYCO handles the FMI standard (both co-simulation and model exchange), and hybrid dynamics (i.e. continuous evolution with state and time events).
- The second step shows that MECSYCO enables the interaction of continuous and time-stepped models, and properly manages the data exchange between FMUs and ad-hoc simulators.
- The last step shows that the NS-3 discrete-event simulator can rigorously interact with FMUs and ad-hoc models in a distributed multi-platform architecture within MECSYCO.

Through this iterative proof of concept, we have shown that MECSYCO enables the modular M&S of a complex system. Indeed, it is important to note that, at each next co-simulation step, we only add and connect the new models to the previous co-simulation. Hence, we do not have to modify neither the models nor their MECSYCO wrappers: we only have to change the co-simulation structure (i.e. models interconnections and co-simulation distribution).

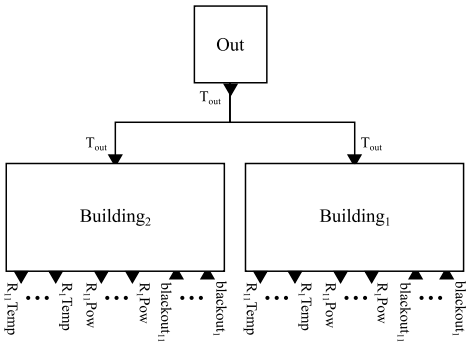
10 Conclusion

In this work, we presented the MECSYCO middleware specifications enabling the rigorous modeling and simulation of complex systems through a co-simulation approach. MECSYCO relies on the universal property of the DEVS formalism in order to integrate models written in different formalisms. This integration is made thanks to a wrapping strategy in order to make models implemented in different simulation software interoperable. The middleware performs then the co-simulation in a parallel, decentralized and distributable fashion thanks to its modular multi-agent architecture.

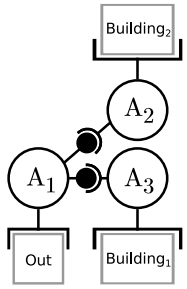
We also stated that although the wrapping of models relies on the formal guaranties offered by DEVS, it can be not trivial to perform. That is why we have detailed how continuous equation-based models -which are amongst the most difficult to integrate- can be integrated in a generic way thanks to the FMI standard in order to perform hybrid co-simulations. We based this wrapping on the DEV&DESS formalism and on the QSS solver strategy. Thus we have illustrated that the wrapping of models in MECSYCO has not necessarily to be done from scratch, but can benefit from the numerous rigorous works around DEVS integration existing in the literature. We also underlined the fact that our DEVS wrapping of the FMI standard is not restricted to MECSYCO but can be performed in any DEVS-based platform.

Compared to other works in the literature, our middleware is generic and modular thanks to the strong foundation of DEVS: there is no need to change the specifications when a model is changed/added/removed in the co-simulation. Yet our solution is evolutionary as one can still keep the framework and change some of the specification, for instance other DEVS co-simulation algorithms can be implemented. Moreover, our middleware is fully specified from the concepts, to their implementations making different implementations of MECSYCO interoperable, and the co-simulations reproducible.

In future works, we plan to extend the application domain of MECSYCO by interfacing more simulation software. We also want to integrate real-time simulation capacity into MECSYCO in order to perform interactive co-simulations. We also want to propose extension of our approach in order MECSYCO to support the whole M&S process from the definition of the experimental plan to the simulation results analysis. We plan to introduce hierarchical modeling like in DEVS, in order to reuse a model predefined with MECSYCO into a wider co-simulation. Finally, we want to develop a Domain Specific Language approach within MECSYCO in order to define co-simulations using directly the experts language. Such an approach could indeed make MECSYCO accessible outside the M&S experts

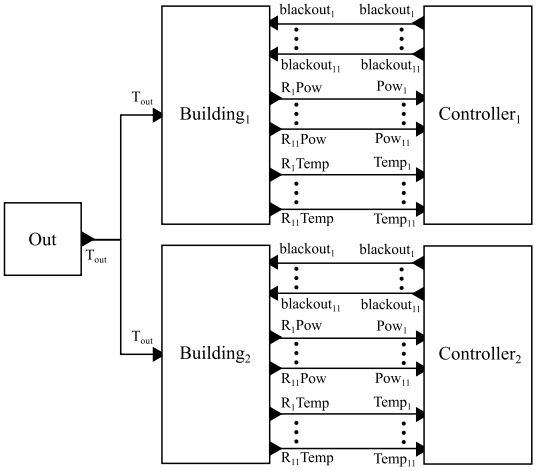


(a) Bloc diagram view of the DEVS model

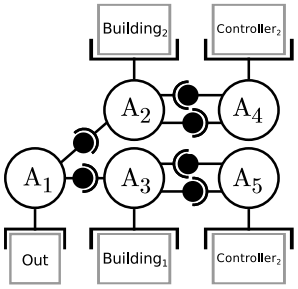


(b) MECSYCO view of the co-simulation

Figure 14: Co-simulation of the building system without controller.

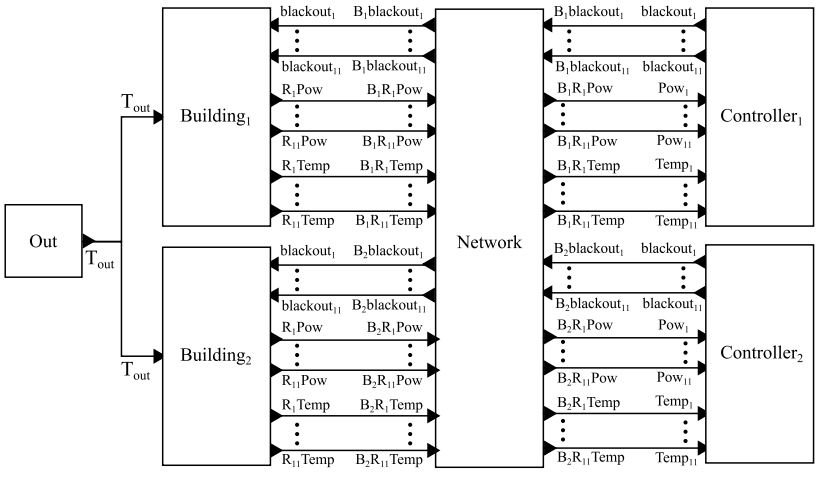


(a) Bloc diagram view of the DEVS model

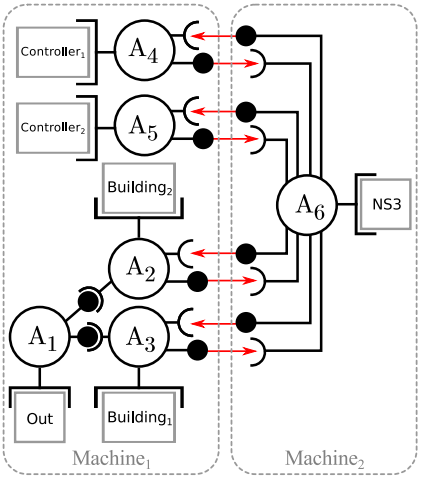


(b) MECSYCO view of the co-simulation

Figure 15: Co-simulation of the building system with a controller but no network

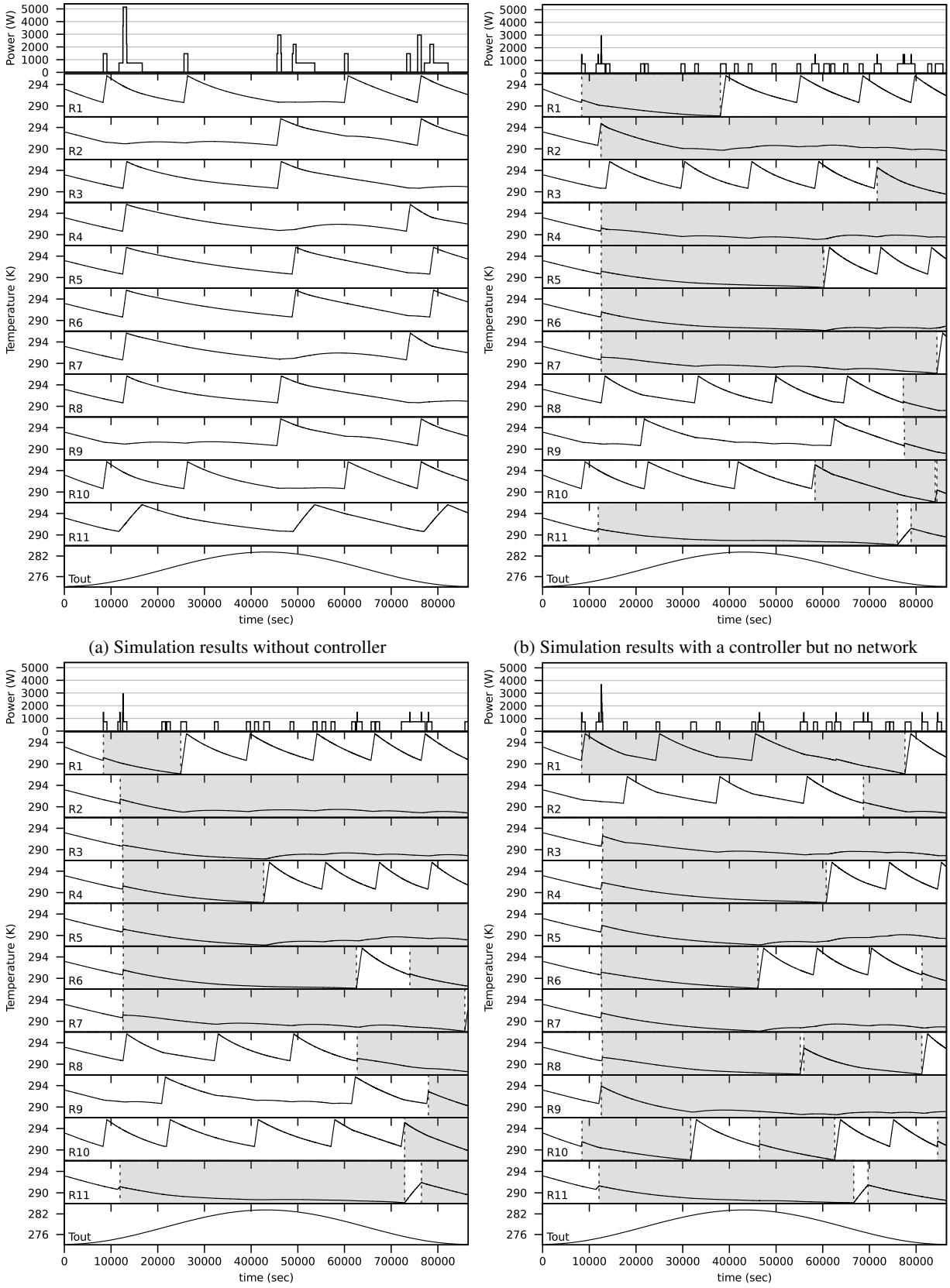


(a) Bloc diagram view of the DEVS model



(b) MECSYCO view of the co-simulation

Figure 16: Co-simulation of the building system with a controller and a network



(a) Simulation results without controller

(b) Simulation results with a controller but no network

(c) Example of simulation results with UDP and 1 bits altered every 10000 ones

(d) Example of simulation results with UDP and 1 bits altered every 1000 ones

Figure 17: MECSYCO co-simulation results of the building-controller system.

Table 3: Parameters used in the smart heating co-simulation use case.

Models	Parameters Descriptions	Values
thermic building	temperature setpoints of the heaters, T_{wanted}	293.15 K
	tolerance of the heaters, $bandwidth$	5 K
	electrical resistance of the heaters, R	2 Ω
	power supply voltage, U	230 V
	thermal capacities of rooms 1 to 10	112.5 kJ/K
	thermal capacities of rooms 11	600 kJ/K
	thermal conductances of the outside walls of rooms 1 and 10	2 J/K
	thermal conductances of the outside walls of rooms 2 to 9	1.25 J/K
	thermal conductance of the outside wall of room 11	7.5 J/K
	thermal conductances of the inside walls between rooms 1 to 10	3.75 J/K
	thermal conductances of the inside walls between rooms 11 and room 1 to 10	2.25 J/K
	rooms initial temperature (NB: identical for all the rooms)	293.15 K
	temperature evolution sampling period	60 s
outside temperature evolution	amplitude	5K
	offset	278.15 K
	period	1 day
	phase	$-\pi/2$
controller	consumption peaks occurrence threshold, Pow_{max}	735 W
	minimum temperature threshold, $Temp_{min}$	288.15 K
	evaluation points period (i.e. model time step)	60 s
	initial evaluation point time (i.e. evaluation points offset)	30 s

circle.

Acknowledgement

This work is partially funded by EDF R&D through the strategic project MS4SG.

References

[1] Chavalarias D, Bourgine P and E Perrier, F Amblard, F Arlabosse, et al. French Roadmap for complex Systems 2008-2009, 2009.

[2] Camazine S, Deneubourg JL, Franks NR et al. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[3] Batty M. Urban modeling. *International Encyclopedia of Human Geography, Elsevier, Oxford* 2009; .

[4] Bretagnolle A, Daudet E and Pumain D. From theory to modelling : urban systems as complex systems. *CyberGeo: European Journal of Geography* 2006; (335): 1–17.

[5] Gil-Quijano J, Louail T and Hutzler G. From biological to urban cells: Lessons from three multilevel agent-based models. In Desai N, Liu A and Winikoff M (eds.) *Principles and Practice of Multi-Agent Systems*, *Lecture Notes in Computer Science*, volume 7057. Springer Berlin Heidelberg, 2012. pp. 620–635.

[6] El Hmam M, Abouaissa, Hassane; Jolly D et al. Macro-micro simulation of traffic flow. In *Proceeding of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM*, volume 12-1. pp. 351–356.

[7] Duboz R, Ramat É and Preux P. Scale transfer modelling: Using emergent computation for coupling an ordinary differential equation system with areactive agent model. *Systems Analysis Modelling Simulation* 2003; 43(6): 793–814.

[8] Gaud N, Galland S, Gechter F et al. Holonic multilevel simulation of complex systems: Application to real-time pedestrians simulation in virtual urban environment. *Simulation Modelling Practice and Theory* 2008; 16(10): 1659 – 1676. The Analysis of Complex Systems.

[9] Vo DA, Drogoul A and Zucker JD. An operational meta-model for handling multiple scales in agent-based simulations. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*. pp. 1–6.

[10] Xiong M, Cai W, Zhou S et al. A case study of multi-resolution modeling for crowd simulation. In Wainer GA, Shaffer CA, McGraw RM et al. (eds.) *SpringSim*. SCS/ACM.

[11] Vangheluwe H, De Lara J and Mosterman PJ. An introduction to multi-paradigm modelling and simulation. In *Proc. AIS2002*. pp. 9–20.

[12] Cellier FE. Combined continuous/discrete system simulation languages–usefulness, experiences and future development. *Methodology in systems modelling and simulation* 1979; : 201–220.

[13] Lara J and Vangheluwe H. AToM3: A tool for multi-formalism and meta-modelling. In Kutsche RD and Weber H (eds.) *Fundamental Approaches to Software Engineering*, *Lecture Notes in Computer Science*, volume 2306. Springer Berlin Heidelberg, 2002. pp. 174–188.

[14] Praehofer H. System theoretic formalisms for combined discrete-continuous system simulation. *International Journal of General System* 1991; 19(3): 226–240.

[15] Barros FJ. Dynamic structure multiparadigm modeling and simulation. *ACM Trans Model Comput Simul* 2003; 13(3).

[16] Esquembre F and Christian W. Ordinary differential equations. In Fishwick PA (ed.) *Handbook of dynamic system modeling*. CRC Press, 2007.

[17] Mosterman P. Hybrid dynamic systems: Modeling and execution. In Fishwick PA (ed.) *Handbook of dynamic system modeling*, chapter 15. CRC Press, 2007. pp. 1–26.

[18] Dahmann JS, Fujimoto RM and Weatherly RM. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*. IEEE Computer Society, pp. 142–149.

[19] Diallo SY, Herencia-Zapana H, Padilla JJ et al. Understanding interoperability. In *Proceedings of the 2011 Emerging M&S Applications in Industry and Academia Symposium*. EAIA '11, San Diego, CA, USA: Society for Computer Simulation International, pp. 84–91.

[20] Wilensky U. Netlogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1999. URL http://ccl.northwestern.edu/netlogo/.

[21] Taillandier P, Vo DA, Amouroux E et al. GAMA: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *Principles and Practice of Multi-Agent Systems*. Springer, 2012.

[22] Henderson TR, Roy S, Floyd S et al. NS-3 project goals. In *Proceeding of WNS2 '06*. ACM, p. 13.

[23] Varga A and Hornig R. An overview of the OMNeT++ simulation environment. In *Proceedings of ICST*. p. 60.

[24] Argent RM. An overview of model integration for environmental applications-components, frameworks and semantics. *Environmental Modelling and Software* 2004; .

[25] Zeigler B, Praehofer H and Kim T. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[26] Vangheluwe H. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Proc. of CACSD '00*.

[27] Barros FJ and Zeigler BP. Model interoperability in the discrete event paradigm: Representation of continuous models. In *Modeling and Simulation: Theory and Practice*. Springer US, 2003. pp. 103–126.

[28] Quesnel G, Duboz R, Versmisse D et al. DEVS coupling of spatial and ordinary differential equations: VLE framework. In *Proc. OICMS '05*.

[29] Zeigler BP. Embedding DEV&DESS in DEVS. In *Proc. DEVS Integrative M&S Symp*, volume 7.

[30] Cellier FE, Kofman E, Migoni G et al. Quantized state system simulation. *Proc GCMS'08, Grand Challenges in Modeling and Simulation* 2008; : 504–510.

[31] Bergero F, Fernandez J, Kofman E et al. Time discretization versus state quantization in the simulation of a one-dimensional advection-diffusion-reaction equation. *Simulation* 2016; 92(1): 47–61.

[32] Kofman E. A second-order approximation for devs simulation of continuous systems. *Simulation* 2002; 78(2): 76–89.

[33] Kofman E. Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing* 2004; 25(5).

[34] Kim YJ and Kim TG. A heterogeneous simulation framework based on the DEVS BUS and the high level architecture. In *Proc. of WSC '98*, volume 1. IEEE.

[35] Mittal S, Ruth M, Pratt A et al. A system-of-systems approach for integrated energy systems modeling and simulation. In *Proc. of SummerSim' 15*. SCS/ACM, pp. 1–10.

[36] Camus B, Bourjot C and Chevrier V. Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proc. of TMS/DEVS 15*. SCS.

[37] Camus B, Bourjot C and Chevrier V. Considering a multi-level model as a society of interacting models: Application to a collective motion example. *JASSS* 2015; 18(3): 7.

[38] Vaubourg J, Presse Y, Camus B et al. Multi-agent multi-model simulation of smart grids in the MS4SG project. In *Proc. PAAMS 15*. Springer, 2015. pp. 240–251.

[39] Siebert J, Ciarletta L and Chevrier V. Agents and artefacts for multiple models co-evolution: building complex system simulation as a set of interacting models. In *Proc. of AAMAS '10*. AAMAS/ACM.

[40] Bonneaud S. *Des agents-modèles pour la modélisation et la simulation de systèmes complexes - Application à l'écosystémique des pêches*. PhD Thesis, 2008.

[41] Jennings NR. An agent-based approach for building complex software systems. *Commun ACM* 2001; 44(4): 35–41.

[42] Ricci A, Viroli M and Omicini A. Give agents their artifacts: the A&A approach for engineering working environments in MAS. In *AAMAS '07*. ACM.

[43] Chandy KM and Misra J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans Software Engineering* 1979; .

[44] Bryant RE. Simulation on a distributed system. In *Proc. of the 16th Design Automation Conf.*

[45] Fujimoto RM. Parallel simulation: parallel and distributed simulation systems. In *Proceedings of the 33nd conference on Winter simulation*. WSC '01, IEEE Computer Society.

[46] Vaubourg J, Chevrier V, Ciarletta L et al. Co-simulation of ip network models in the cyber-physical systems context, using a devs-based platform. In SCS/ACM (ed.) *Communications and Networking Simulation Symposium (CNS'16)*.

[47] Blochwitz T, Otter M, Åkesson J et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proc. 9th International Modelica Conference*. pp. 173–184.

[48] MODELISAR Consortium and Modelica Association. Functional mock-up interface for model exchange and co-simulation – version 2.0, july 25, 2014. retrieved from https://www.fmi-standard.org.

[49] Broman D, Brooks C, Greenberg L et al. Determinate composition of FMUs for co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*. EMSOFT '13, Piscataway, NJ, USA: IEEE Press.

[50] Cremona F, Lohstroh M, Tipakis S et al. FIDE – an FMI integrated development environment. In ACM (ed.) *SAC'16*.

[51] Galtier V, Vialle S, Dad C et al. FMI-based distributed multi-simulation with DACCOSIM. In *Proc. of TM-S/DEVS 15*. SCS, pp. 39–46.

[52] Kofman E and Junco S. Quantized-state systems: a devs approach for continuous system simulation. *Transactions of The Society for Modeling and Simulation International* 2001; 18(3): 123–132.

[53] Tavella JP, Caujolle M, Tan C et al. Toward an Hybrid Co-simulation with the FMI-CS Standard, 2016. Research Report.

[54] Hernández-Cabrera JJ, Évora Gómez J and Cortès-Montenegro J. JavaFMI. SIANI. University of Las Palmas, Spain.

[55] Camus B, Galtier V, Caujolle M et al. 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*.

[56] Moler C. Are we there yet? *Zero crossing and event handling for differential equations, Matlab News & Notes* 1997; .

[57] Schütte S. *Simulation model composition for the large-scale analysis of smart grid control mechanisms*. PhD Thesis, BIS der Universität Oldenburg, 2013.

[58] Dahmann J and Morse K. High level architecture for simulation: an update. In *Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings. 2nd International Workshop on*. pp. 32–40.

[59] Kim TG and Kim JH. DEVS framework and toolkits for simulators interoperation using HLA/RTI. In *Proceedings of Asia Simulation Conference/the 6th International Conference on System Simulation and Scientific Computing*. pp. 16–21.

[60] Zeigler BP, Cho H, Lee J et al. The DEVS/HLA distributed simulation environment and its support for predictive filtering. *DARPA Contract N6133997K-0007: ECE Dept, UA, Tucson, AZ* 1998; .

[61] Bergero F, Floros X, Fernandez J et al. Simulating modelica models with a stand-alone quantized state systems solver. In *Proc. 9th International MODELICA Conference*. 076, Linköping University Electronic Press, pp. 237–246.

[62] Floros X, Bergero F, Ceriani N et al. Simulation of smart-grid models using quantization-based integration methods. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096, Linköping University Electronic Press, pp. 787–797.

[63] Gilpin L, Ciarletta L, Presse Y et al. Co-simulation Solution using AA4MM-FMI applied to Smart Space Heating Models. In *7th International ICST Conference on Simulation Tools and Techniques*. Lisbon, Portugal, pp. 153–159.

[64] Fritzson P and Engelson V. Modelica—a unified object-oriented language for system modeling and simulation. In *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.