Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO

Benjamin Camus
Université de Lorraine, CNRS,
Inria, LORIA, UMR 7503
Vandœuvre-lès-Nancy,
F-54506, France.
benjamin.camus@loria.fr

Virginie Galtier
CentraleSupélec
UMI GT-CNRS 2958
Université Paris-Saclay
F-57070, Metz, France
virginie.galtier@centralesupelec.fr

Mathieu Caujolle
EDF - R&D MIRE/R44
1 avenue du Général de Gaulle
BP 408 F-92141,
Clamart, France
mathieu.caujolle@edf.fr

ABSTRACT
Co-simulation is a key tool in the design and operation of a growing number of complex cyber-systems. But efficiently yet accurately combining continuous time components (such as FMUs) with event-based ones can be challenging, both from a modeling perspective and an operational, tools-oriented one. We propose a platform to tackle this problem building up on MECSYCO, a MAS-based DEVS wrapping platform dedicated to co-simulation. Relying on the ability of DEVS to integrate the DEV&DESS formalism -which offers a sound framework for describing hybrid models- we propose a DEV&DESS wrapper for FMU. This wrapper encapsulates a version of the DEV&DESS simulation algorithm for FMU components which is notably composed of: (1) a forecast strategy which searches for the next state-event; (2) a bisectional algorithm to approach the location of the state-change in an FMU. Our solution is implemented using Java and JavaFMI to control the FMU. Our sample case is the co-simulation of a barrel-filler factory implemented in different FMUs and event-based models. Compared to related works, our proposal is functional, generic, yet evolutionary, and benefits from the strong foundations of DEV&DESS.

Author Keywords
co-simulation; hybrid model; DEVS; DEV&DESS; FMI

ACM Classification Keywords
I.6.1 SIMULATION AND MODELING: (e.g. Model Development).

1. INTRODUCTION

Within the next two decades the electrical distribution grids will undergo large transformations with the massive connection of intermittent energy resources (wind and solar generation) at Low and Medium Voltage levels, as well as the development of new usages such as electric vehicles charging. These evolutions burst the need for new functions, tools and communication media to be integrated in the Distribution Management System (DMS) of the Distribution System Operators (DSO), making the power grids ”smarter”.

These evolutions introduce a new level of complexity when studying the resulting system: the proactive management of the Smart Grids requires that domains such as telecommunication and information systems should be considered in addition to the ”classical” electrical world. And, because of the strong interactions existing between these domains, considering each domain separately becomes no longer sufficient to picture an accurate state of the whole system. Real testbeds are a necessary stage before experimenting on the field, but they are time and cost consuming and provide limited insight regarding large scale deployment.

A solution to this problematic thus lies in co-simulation. Co-simulation consists in performing a simulation by reusing models executed in different pre-existing simulation software. It allows each specialist involved in a complex system to keep using the tools which are popular in his/her community (for instance NS-3 for telecommunication networks specialists, Dymola for physics and control specialists, EMTP-RV for electromagnetic specialists...) 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. The challenge is to ensure interoperation of rather stand-alone tools: to synchronize the execution of the simulators, and to perform actual exchange of simulation data among them.

FMI (Functional Mock-Up Interface) [3] establishes itself as a standard for model exchange and co-simulation of equational models. It enforces some generic rules and a software interface to manipulate equational models and their numerical solver. Building on that interface, any equational component can be embedded into an FMU (Functional Mock-up Unit) helping to solve the interoperability problem for the co-simulation of equational models. The numerical resolution of a system can then be performed by defining a set of communication points between the FMUs according to a trade-off between the accuracy of the simulation results and the performances of the co-simulation process.

In the context of the modeling and simulation (M&S) of complex cyber-systems such as Smart Grids, the FMU components must interact with models written in others formalisms –such as event-based ones– within a multi-paradigm [27, 23] co-simulation. The issues faced are then related to hybrid modelling [8] where discrete and continuous formalisms are combined: in order to manage the FMU execution, we need to deal with event occurrences in addition to the communica-

SpringSim-TMS/DEVS 2016, April 3-6, 2016, Pasadena, CA, USA

tion points of the equations numerical resolution. However, as it focuses on the software API level, FMI does not offer a solution to this integration need. Thus, in order to obtain a generic solution, a sound framework for describing hybrid co-simulation with FMUs is needed.

We show in this article that, thanks to its integrative power [23] and its capacity to embed the DEV&DESS hybrid system formalism, the Discrete EVent System specification (DEVS) fullfils this requirement. **Our contribution consists in specifying a DEVS wrapper for hybrid co-simulation of FMU components by relying on DEV&DESS.** As an FMU offers a limited control whereas DEV&DESS does not make hypothesis on the equational models manipulation, **we propose to adapt the DEV&DESS simulation protocol in this wrapper.** We implement this proposal in MECSYCO, our DEVS wrapping platform.

The article is articulated as follows. Section 2 describes a simple yet representative example of an hybrid system and details the issues raised by co-simulation. Section 3 presents DEVS and how it enables hybrid modelling, and describes MECSYCO. Section 4 constitutes the center of our contribution as it describes our DEVS wrapper for FMU components. Section 5 presents the implementation of our solution and the simulation results for our use-case. Section 6 positions our proposition with regards to related works. Section 7 summarizes our contribution and lists perspectives.

2. USE-CASE

We present here the case of a barrel-filler factory (inspired from [18]) composed of four parts (Figure 1) detailed below. This example was chosen for an explanatory purpose, but also to illustrate that our work is generic and can be applied outside from the complex context of the Smart Grids.

- 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** storing the water to fill the barrels. The flow rate of water filling the barrel decreases with the level of water in the tank.
- A **controller** $c1$ manages the opening of the valve 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).
- A **controller** $c2$ regulates the whole filling process. It can change the target level of the barrels, and abort the current barrel filling. In the former case, when the command is received, the factory takes into account the new target level only after the current barrel has reached the previous target level. In the latter case, after a short reaction delay the current barrel is carried away and another barrel is filled.

To illustrate hybrid simulation, we choose to model the factory as a combination of continuous (the water flow from the tank to the barrels) and discrete (changes of barrel and

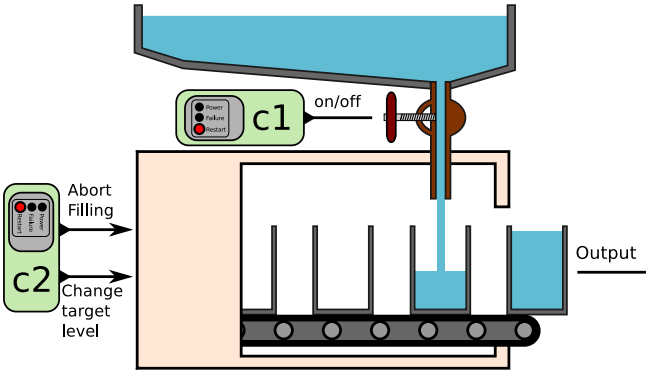


Figure 1. The barrel-filler factory.

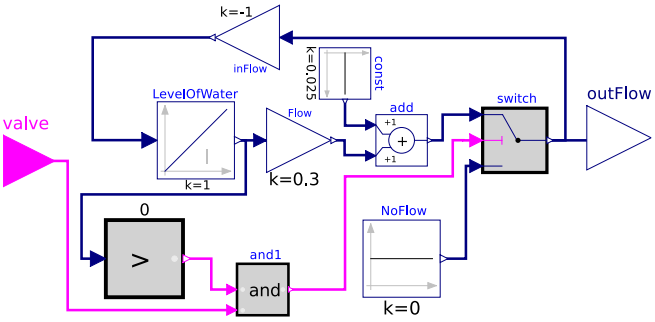


Figure 2. The *tank* model viewed in Dymola.

controllers commands) dynamics. We use four models to describe the factory functioning.

Two event-based models describe the discrete dynamics. These models send commands at predefined dates. The ***c1* model** sends "open"/"close" events through its *valve* output port in order to control the barrels water supply. The ***c2* model** has two output ports, *abort* and *size*, used respectively to abort the current barrel filling and to change the target level of the barrels.

Two equational models are used to describe the continuous dynamics. These models are separately defined in Modelica using block diagrams, and exported from Dymola as FMUs for co-simulation according to FMI 2.0 standard. The ***tank* model** (Figure 2) describes, according to the valve status, the evolution of the level of water in the tank, and the resulting output flow rate. The ***barrel* model** (Figure 3) describes the evolution of the level of water in the current barrel, according to the input flow rate.

This example illustrates that the continuous component must define within all its continuous states, some states which are meaningful for the event-oriented world. For instance a barrel can be *empty*, *filling* or *target reached*. The change from *filling* to *target reached* constitutes a "state event"[8]. In a classic FMU-based co-simulation, each FMU executes independently and FMUs exchange simulation data at pre-determined instants of communication. The frequency of these communications is chosen according to a performance-accuracy trade-off. From a model point of view, it is obvious that when a state

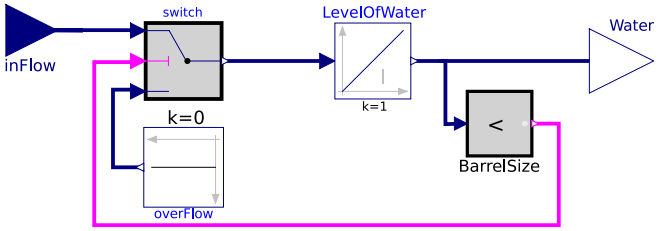


Figure 3. The *barrel* model viewed in Dymola.

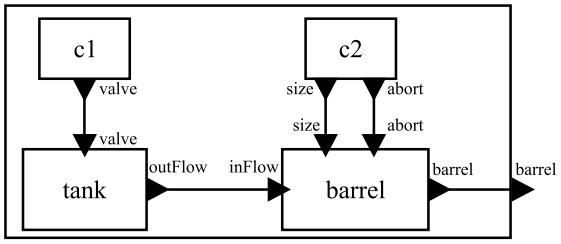


Figure 4. DEVS view of the barrel-filler factory.

event occurs, it is important that the continuous component immediately informs the event-based world; otherwise, simulation would skew the factory behaviour: the factory will keep filling the barrel for a while after the target level is reached. As a consequence, the result of the simulation - the number of barrels produced by the factory - will be inaccurate: it will be less than what the tank volume would allow in the real factory. Reciprocally, when the continuous model (the barrel) receives a command (for instance "abort") from the control system, it must take it into account immediately in conformance with the model: if after the reaction delay, the barrel keeps filling up for a while, the result of simulation will again be inaccurate.

But with an FMU-based co-simulation, these two actions are not done so easily since :

- state events occuring between 2 points of communication are localized at the upper communication point, pending improvements of the hybrid-cosimulation 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. In addition, this integration should be as generic as possible since this example is for illustration purposes only and the solution we propose applies in a broader context.

3. MULTI-PARADIGM M&S WITH A DEVS WRAPPING PLATFORM

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 [22]. Indeed, not only DEVS appears to be universal for describing discrete-event systems [25], but it can also integrate continuous systems [2] expressed for instance with differential equations [19]. Of particular interest in the scope of this article is the fact that, as shown by [26], DEVS can also embed the DEV&DESS formalism [18]. This formalism offers a sound framework for describing hybrid systems as it describes how continuous systems interact and co-evolve with the discrete world.

As summarized by [19], 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 [12]. The advantage of the wrapping strategy is to enable reusing pre-existing models already implemented in some simulation software [14]. Once integrated in DEVS, different heterogeneous models can be co-simulated using a common simulation protocol: the sequential or the parallel DEVS one. Thanks to its integrative power, DEVS constitutes a strong base for the M&S of complex cyber-systems.

The Figure 4 shows a DEVS view of our barrel-filler factory example. Each model corresponds to a DEVS atomic model. Together, these atomic models describe the behaviour of the system, and the whole barrel-filler factory is represented by a DEVS coupled model which describes the structure of the system (i.e. how the atomic models are interconnected).

3.2. Presentation of MECSYCO

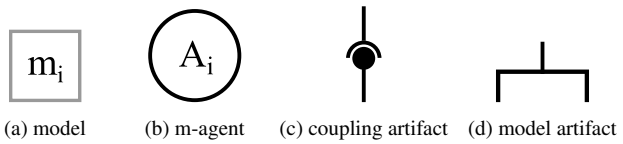
A Multi-agent Environment for M&S

MECSYCO (Multi-agent Environment for Complex SYstem CO-simulation) [5] 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 [6], 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 Inria (French IT research institute) and EDF R&D (leading French electric utility company) [24].

MECSYCO is based on the AA4MM (Agents & Artifacts for Multi-Modeling) paradigm [21] 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. Originality with regard to other multi-agent multi-model approaches is to consider the interactions in an indirect way thanks to the concept of artifacts [20]. By following this multi-agent paradigm from the concepts to their implementation, MECSYCO ensures a modular, 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.

Operational Specifications

MECSYCO relies on four concepts to describe a multi-model.



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

Figure 5. Symbols of the MECSYCO components.

A **model** m_i is a partial representation of the target system implemented in a simulation software s_i (symbol in Figure 5a). 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 5b) manages a model m_i and is in charge of interactions of this model with the other ones. The behavior of each m-agent corresponds to the DEVS conservative parallel abstract simulator which is based on the Chandy-Misra-Bryant (CMB) algorithm [9, 4]. This algorithm is proven to be deadlock free and to respect the causality constraint [25].

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 5c). 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. 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 [6].

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 5d). 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. 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,
- $getOutputEvent(y_i^k)$ returns $e_{out_i}^k$, the external output event at the k^{th} output port of m_i, y_i^k,
- $getNextInternalEventTime()$ returns the time of the earliest scheduled internal event of the model m_i.

The Figure 6 shows how these concepts are combined in order to perform the co-simulation of the barrel-filler factory. For each simulation software, we define a model artifact performing the wrapping with DEVS. Thanks to these artifacts the factory model can be simulated as the DEVS coupled model of Figure 4.

Defining the model artifacts for the two controllers is trivial as these models are already event-based. In previous works

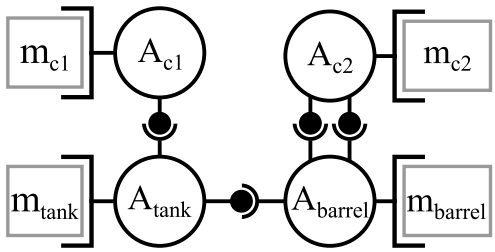


Figure 6. MECSYCO architecture for the barrel-filler factory.

on MECSYCO like [24], the authors have created model artifacts for FMUs solely based on the FMI for co-simulation specifications. However as discussed in the following section, wrapping the two FMUs in DEVS in our more complex hybrid context requires defining a sound DEV&DESS model artifact.

4. MODEL ARTIFACT FOR HYBRID CO-SIMULATION OF FMUS

4.1. Foundation

As any model in MECSYCO, the 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. As stated in 3.1., 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 described by [26], in its DEVS version 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. In our model artifact, we propose this component to correspond to the FMU we want to integrate into an hybrid co-simulation.
- **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. In our model artifact, this component describes the behaviour of the FMU 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 FMU evolution) and (3) the event detection function.

4.2. Application to FMU Components

For managing the continuous component evolution, the original DEVS version of DEV&DESS relies on a quantized integrator approach like the Quantized State Systems (QSS) method [13]. This approach is based on state quantization instead of the time discretization used by traditional integration methods. 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. However, as stated before, we add in our model artifact an operational constraint: the continuous component corresponds to an FMU i.e. an equation-based model 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't be considered as a QSS model, and therefore we need to adapt the original DEVS version of DEV&DESS in our model artifact.

More precisely, an FMU component provides the following functionalities: (1) perform an integration for a given time-step (*fmi2DoStep*), and (2) set inputs and get outputs (*fmi2SetReal/Integer/Boolean/String* and *fmi2GetReal/Integer/Boolean/String*). We also assume that it is possible to implement a roll-back functionality to go one single integration step back. This last feature is feasible if the FMU component implements the *fmi2GetFMUState* and *fmi2SetFMUState* optional functions enabling to export/import the model state.

According to the FMI co-simulation specification, we consider that this FMU component produces outputs at a sequence of pre-define 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.

The next section presents how we propose to implement the DEVS simulation protocol within our model artifact in order to respect the DEV&DESS semantic with these FMI constraints.

4.3. Functioning of the DEV&DESS Model Artifact for FMU Components

Getting the Time of the Next Internal Event

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. 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 [16] in order to localize the state-event as precisely as possible in the time. This search is formalized by the Algorithm 1 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 cancelled at a time, the algorithm always goes back to the legitimate state before performing a new integration step.

Algorithm 1 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**
    $solver.rollBack()$
    $\Delta t \leftarrow \Delta t/2$
    $solver.doStep(\delta t + \Delta t)$
    if $\neg detFunction.stateEventOccurence()$ **then**
      $\delta t \leftarrow \delta t + \Delta t$
    end if
  end for

Processing Events

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 behaviour 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.

4.4. Known Limitation

A similar approach is used by DACCOSIM [10, 7] to approach the date of a state-change, except DACCOSIM proceeds sequentially instead of bisectionally. A limitation of both solutions is that some state-changes might go undetected; for instance, if a boolean value changes twice during the exploration, the detection function will not see its value as modified. For this reason, it is important that the co-simulation designer carefully sets the date of the first data exchange between continuous and event-based domains so that only one state-change could take place within an exploration. Otherwise we could observe the occurrence of late propagation of internal events: let's assume the next communication point is set to t_{10} and a boolean value changes at t_2 and t_9. The first exploration, simulating from t_0 to t_{10} will not detect the t_2 and t_9 events (because of the even number of state-changes). Let's imagine another agent propagates an event for t_8. The equation-based component will roll back to t_0 and resume a simulation from t_0 to t_8. The boolean value which changed at t_2 will be propagated with a time-stamped of t_8 instead of t_2, which is the correct value.

5. IMPLEMENTATION AND USE-CASE EXECUTION

5.1. Model Artifacts Definition

In our use-case example, we define two DEV&DESS model artifacts $\mathcal{I}_{barrel}$ and $\mathcal{I}_{tank}$ to manage the integration of the *barrel* and *tank* FMUs into the discrete world. We define the discrete-event components and the event-detection functions of these model artifacts for wrapping these two models, .

In $\mathcal{I}_{barrel}$, the event detection function considers that a state event occurs when the level of water in the *barrel* FMU reaches the desired threshold (which is initialized at $1\ liter$). Thus, state-events correspond to switches of barrels.

The discrete-event component of $\mathcal{I}_{barrel}$ has two discrete input ports named *size* and *abort*. Through the *abort* port, the component can receive requests for canceling the current barrel filling. When such an event is received at a date t, the component external transition function schedules an internal event at a date $t + d$ –the internal event representing the switch of barrel, and d representing the reaction delay of the model. Through the *size* port, the component can receive update notifications of the desired level of water in the barrels. When such an event is received, the component external transition function stores the command in order to apply it at the next state event (i.e. after the current barrel is full) or next internal event (i.e. after receiving a cancel order).

When its internal transition function is triggered either by a state or an internal event, the discrete-event component represents the switch of barrel by: (1) producing through its *barrel* discrete output port an external event in order to signal the barrel departure, (2) resetting the FMU state (thus setting the water level back to 0) in order to represent the arrival of a new barrel, and (3) changing the threshold of the event detection function if a command awaits application.

In $\mathcal{I}_{tank}$, there is no state-event, so the event-detection function is disabled. The discrete-event component does not have any output port, but has one input port in order to receive command to open or close the water inlet. When the command is received, the event-driven component changes the value of the valve variable in the FMU accordingly.

5.2. Implementation

We used the Java version of MECSYCO (available at http://mecsyco.com) in order to implement our DEV&DESS model artifacts. This version of MECSYCO relies on the OMG standard Data Distribution Service (DSS). The event-based models were directly implemented in Java. The equational models were written in Modelica language and exported in FMI 2.0 using the Dymola software. Our DEV&DESS model artifact relies on JavaFMI [11] in order to interact with the FMUs. Yet let's emphasize that FMUs were only used as illustrative examples. The same results can also be achieved (and indeed were) with continuous models written directly in Java, provided that a wrapper code is developed to offer an FMU-like interface for the Java model.

5.3. Results

The Figure 7 shows the results of the MECSYCO barrel-filler factory co-simulation (with the reaction delay of $\mathcal{I}_{barrel}$ equals to 0.01). At $t = 0$, the tank contains 7 l and the barrel is empty. At $t = 0.5$, the valve opens and the liquid starts flowing from the tank into the barrel (volume of liquid in the tank continuously decreases while volume in the barrel increases). At $t = 1.016$, the volume of liquid in the barrel reaches 1 l. This triggers the emission of the barrel and a new barrel is set up (volume in barrel is 0 again). At $t = 1.613$ the 2^{nd} barrel is full (1 l). Again, that triggers the emission of the barrel and a new barrel is set up under the tank. But at $t = 1.9$ the factory is ordered to abort the current barrel. After a very small delay, at $t = 1.91$, the 3^{rd} barrel, containing only 0.43 l, is pushed out and a new barrel is set up and starts filling up. At $t = 2.715$ an event notifying that this 4^{th} barrel is full is generated and a 5^{th} barrel is set up. But at $t = 3.2$ the valve is closed. The tank stops emptying and the volume in the barrel stays the same. At $t = 5.2$ the valve opens again and the 5^{th} barrel resumes its filling. At $t = 5.773$ this 5^{th} barrel is pushed out and a 6^{th} barrel is installed. But at $t = 6.5$, before this 6^{th} barrel reaches 1 l, the desired volume in the barrel is changed from 1 l to 0.5 l. The factory keeps filling up to 1 l the 6^{th} barrel but when that old threshold is reached at $t = 7.342$, the new target volume is taken into account for the filling of the 7^{th} barrel: at $t = 8.535$, when the volume in the 7^{th} barrel reaches 0.5 l, the barrel is pushed out and replaced by a new one. As the volume in the tank is getting low, the output rate of the tank decreases and it takes a little

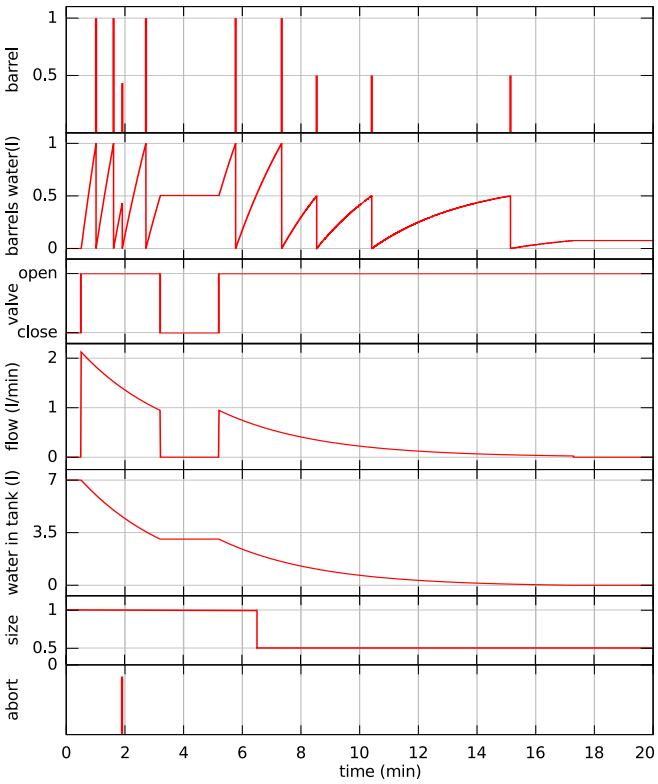


Figure 7. Simulation results.

longer for this 8^{th} barrel to fill up and be replaced by a 9^{th} barrel which reaches 0.5 l at $t = 15.139$. The 10^{th} barrel, installed at $t = 15.139$, fills up to 0.07 l but afterwards the tank is empty.

The Table 1 compares the times where the barrels are generated with our solution (with $m = 100$), and with a classic FMU co-simulation strategy (still performed with MECSYCO but without using our DEV&DESS model artifact) where the events are detected and integrated at each communication points only. For each solution, we test different communication step sizes for the numerical resolution of the FMU. We can see that with a classic FMU co-simulation strategy, events occurrences are directly dependent on the communication step size. On the opposite, with our solution events occur at the same simulation times (with a precision of 10^{-3} at least) no matter the configuration required by the FMU numerical resolution. Thus, our solution enables to reconcile both the event-based and the equational worlds needs.

6. RELATED WORK

Aside from the papers already cited in the previous sections, other works related to hybrid modeling should be mentioned here. In [17], the authors integrate FMUs into Ptolemy II event-based models by relying on a similar exploration mechanism for detecting state-events. However, this solution does not provide the same richness of expression offered by our DEV&DESS wrapper: events can't change the event-detection function, and discrete behavior like internal events scheduling can't be specified.

The Heterogeneous Flow System Specification (HFSS) [1] also offers an interesting hybrid M&S framework as it brings numerical multi-rate integration methods for modeling and simulating continuous parts of the systems. However, this formalism is not yet fully integrated in DEVS [26], and therefore doesn't benefit from the strong integrative power of this latter. Yet, defining an HFSS wrapper for FMU-like components should be investigated in future works.

7. PERSPECTIVES AND CONCLUSIONS

In this article, we proposed a fully implemented and functional wrapper (called model artifact in the MECSYCO paradigm) for integrating FMU 2.0 components into a hybrid co-simulation. This wrapper implements the DEVS simulation protocol based on the DEV&DESS semantics and the FMI 2.0 operational constraints. Thanks to our wrapper, FMU and event-based components can interact within the rigorous integrative framework of DEVS. The hybrid co-simulations of these components can be then managed by one of the many DEVS simulation algorithms, in our case the parallel conservative one. Thanks to the strong foundations of DEV&DESS, this wrapper is generic: there is no need to change the algorithms when the model is changed. Yet our solution is evolutionary: one can still keep the framework and change the algorithms, for instance the event localization one.

Our solution will be used in smart spaces and large smart-grid simulations which will provide testbeds richer than our use case (e.g. more models, larger models, distributed components) and drive future evolutions. In future works, we plan to test other state-event localization strategies such as the Illinois algorithm [15] or a combination of existing algorithms. Having different strategies available would enable adapting our solution to different operational constraints, such as the simulation performance (e.g. where we should limit the number of rollbacks of a large FMU), or the simulation accuracy. We also want to integrate in our wrapper the new event-detection features planned for the FMI 2.1 standard as soon as they will be available. Finally, in order to manage a set of FMUs instead of a single one, we also plan to integrate DACCOSIM [10] as a continuous component into our model artifact. This integration can be done directly through the DACCOSIM interface, or by embedding DACCOSIM into an FMU (using for instance the FMU builder of JavaFMI).

ADDITIONAL AUTHORS

1. Vincent Chevrier, Université de Lorraine, CNRS, Inria, LORIA, UMR 7503, Vandœuvre-lès-Nancy, F-54506, France. vincent.chevrier@loria.fr

2. Julien Vaubourg, Inria, Université de Lorraine, CNRS, LORIA, UMR 7503 Vandœuvre-lès-Nancy, F-54506, France. julien.vaubourg@inria.fr

3. Laurent Ciarletta, Université de Lorraine, CNRS, Inria, LORIA, UMR 7503 Vandœuvre-lès-Nancy, F-54506, France. laurent.ciarletta@loria.fr

4. Christine Bourjot, Université de Lorraine, CNRS, Inria, LORIA, UMR 7503 Vandœuvre-lès-Nancy, F-54506, France. christine.bourjot@loria.fr

	step size	$t_{barrel1}$	$t_{barrel2}$	$t_{barrel3}$	$t_{barrel4}$	$t_{barrel5}$	$t_{barrel6}$	$t_{barrel7}$	$t_{barrel8}$	$t_{barrel9}$
Series1	**0.001**	1.016	1.613	1.910	2.715	5.773	7.342	8.535	10.412	15.139
	0.01	1.016	1.613	1.910	2.715	5.773	7.342	8.535	10.412	15.139
	0.1	1.016	1.613	1.910	2.715	5.773	7.342	8.535	10.412	15.139
Series2	**0.001**	1.017	1.615	1.910	2.716	5.774	7.344	8.538	10.419	15.167
	0.01	1.020	1.620	1.900	2.710	5.770	7.340	8.540	10.430	15.230
	0.1	1.100	1.800	1.800	2.600	5.700	7.300	8.500	10.400	15.300

Table 1. Comparison of the barrels output times (rounded to 10^{-3}) between our DEV&DESS integration of the FMU (Series1) and a classic FMU co-simulation strategy (Series2).

REFERENCES

1. Barros, F. J. Dynamic structure multiparadigm modeling and simulation. *ACM Trans. Model. Comput. Simul. 13*, 3 (July 2003), 259–275.
2. Barros, F. J., and Zeigler, B. P. Model interoperability in the discrete event paradigm: Representation of continuous models. In *Modeling and Simulation: Theory and Practice*. Springer US, 2003, 103–126.
3. 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* (2012), 173–184.
4. Bryant, R. E. Simulation on a distributed system. In *Proc. of the 16th Design Automation Conf.* (1979).
5. 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 (2015), 85–90.
6. 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 18*, 3 (2015), 7.
7. Camus, B., Caujolle, M., and Galtier, V. MECSYCO-DACCOSIM coupling. Tech. rep., LORIA, Inria, EDF R&D, CentraleSupélec, 2015.
8. Cellier, F. E. Combined continuous/discrete system simulation languages–usefulness, experiences and future development. *Methodology in systems modelling and simulation* (1979), 201–220.
9. Chandy, K. M., and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Software Engineering* (1979).
10. Galtier, V., Vialle, S., Dad, C., et al. FMI-based distributed multi-simulation with DACCOSIM. In *Proc. of TMS/DEVS 15*, SCS (2015), 39–46.
11. Hernández-Cabrera, J. J., Évora Gómez, J., and Cortès-Montenegro, J. JavaFMI. SIANI. University of Las Palmas, Spain.
12. Kim, Y. J., and Kim, T. G. A heterogeneous simulation framework based on the DEVS BUS and the high level architecture. In *Proc. of WSC '98*, vol. 1, IEEE (1998).
13. Kofman, E. Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing 25*, 5 (2004).
14. 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 (2015), 1–10.
15. Moler, C. Are we there yet? *Zero crossing and event handling for differential equations, Matlab News & Notes* (1997), 16–17.
16. Mosterman, P. Hybrid dynamic systems: Modeling and execution. In *Handbook of dynamic system modeling*, P. A. Fishwick, Ed. CRC Press, 2007, ch. 15, 1–26.
17. Muller, W., and Widl, E. Linking FMI-based components with discrete event systems. In *Proc. SysCon* (2013), 676–680.
18. Praehofer, H. System theoretic formalisms for combined discrete-continuous system simulation. *International Journal of General System 19*, 3 (1991), 226–240.
19. Quesnel, G., Duboz, R., Versmisse, D., and Ramat, É. DEVS coupling of spatial and ordinary differential equations: VLE framework. In *Proc. OICMS '05* (2005).
20. 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 (2007).
21. 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 (2010).
22. Vangheluwe, H. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Proc. of CACSD '00* (2000), 129–134.
23. Vangheluwe, H., De Lara, J., and Mosterman, P. J. An introduction to multi-paradigm modelling and simulation. In *Proc. AIS2002.* (2002), 9–20.
24. 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, 240–251.
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. Zeigler, B. P. Embedding DEV&DESS in DEVS. In *Proc. DEVS Integrative M&S Symp*, vol. 7 (2006).
27. Zeigler, B. P., and Ören, T. I. Multifaceted, multiparadigm modeling perspectives: tools for the 90's. In *Proc. of WSC '86*, ACM (1986), 708–712.