Generation of Co-simulation Algorithms Subject to Simulator Contracts*

Cláudio Gomes[1], Casper Thule[2], Levi Lúcio[3],
Hans Vangheluwe[1], and Peter Gorm Larsen[2]

[1] University of Antwerp, Flanders Make, Belgium,
{claudio.gomes, hans.vangheluwe}@uantwerp.be
[2] Aarhus University, Denmark, casper.thule@eng.au.dk
[3] fortiss, Munich, Germany lucio@fortiss.org

Abstract. Correct co-simulation results require a careful consideration of how the interacting simulators are implemented. In version 2.0 of the FMI Standard, input handling implementation is left implicit, which leads to the situation where a simulator can be interacted with in a manner that its implementation does not expect, yielding incorrect results.

In this paper, we build on prior work to make information about each simulator implementation explicit, in order to derive correct interactions with it. The formalization we use is specific to two kinds of contracts, but could serve as a basis to a general approach to black box co-simulation. The algorithm we propose generates a co-simulation execution plan in linear time. It has been successfully applied to an industrial case study, and the results are available online.

Keywords: co-simulation, prolog, contract-based code generation, constraint solving.

1 Introduction

Correct co-simulation results require a careful consideration of how the interacting simulators are implemented (e.g., see [12, 13, 19], and references thereof). Co-simulation is a technique to combine multiple black-box simulators, each responsible for a model, in order to compute the behavior of the combined models over time [16]. The simulators, often developed independently from each other, are coupled using a master algorithm, also often developed independently, that communicates with each simulator via its interface. This interface comprises functions for setting/getting inputs/outputs, and computing the associated model behavior over a given interval of time. An example of such interface, the terminology of which we adopt here, is the Functional Mockup Interface

* C. G. is a FWO Research Fellow, at the University of Antwerp, supported by the Research Foundation - Flanders (File Number 1S06316N). We thank the organizers of the CAMPaM workshop for providing a platform where the ideas described here flourished.

(FMI) Standard [3, 4]. In the FMI Standard, the simulators are called Functional Mockup Units (FMUs).

The widespread adoption of co-simulation is hindered by the lack of guarantees on the correctness of the results [13]. Indeed, a recent empirical survey has shown that practitioners still experience difficulties in the configuration of co-simulations [17,18]. Version 2.0 of the FMI Standard does not impose a single way of interacting with an FMU (see Section 2.1). However, different interaction protocols with an FMU lead to assumptions on the implementation of that FMU. Recent work [12] shows that one of the reasons for these difficulties is the lack of information about the implementation of each FMU.

Contribution. In this paper, we propose a way to model simulator capabilities, which we denote as contracts, and automatically generate fixed-step master algorithms that satisfy those contracts. While our long term research goal is to consider arbitrary contracts, in this paper, we restrict our attention to input approximation and output calculation contracts. These contracts correspond to a partial view of how the FMUs implement their input approximation schemes and the algebraic dependencies used to calculate the outputs. Hence, they do not expose intellectual property. As we argue next, respecting these contracts is a necessary condition to obtaining correct results. In the future, more advanced master algorithms can be generated if simulators expose more capabilities (Section 5 discusses some of these).

Prior Work. The need for these contracts has been identified in prior work [11] and an incomplete solution is advanced in [12]. The solution proposed in [11] works under the assumptions that FMUs have the same contract for every input (because it assumes FMUs have a single input/output vector), and the solution described in [12] neglects how the outputs are computed [12, Assumption 2]. The submitted manuscript [10] addresses these omissions and describes the semantics of a master algorithm that satisfies such contracts. Since that formalization is defined in pure Prolog, it can be used to generate master algorithms as well. However, this process takes exponential time in the size of the co-simulation scenario (which, for long running co-simulation, becomes negligible). With the current manuscript, we propose a linear time algorithm to perform such generations.

Structure. The next section recalls the formalization proposed in [10], shows why different contracts require different FMU implementations, and formalizes our research problem. Then, Section 3 describes our contribution and application results. Section 4 describes related work and Section 5 concludes.

2 Background

In this section, we provide a formalization of FMI co-simulation with a restricted set of contracts. We show, through a simple but representative example, that minimizing the error in the co-simulation involves a careful consideration of both master and FMU implementations of such contracts. Such formalization has been implemented in Prolog, presented in [10], and available online[4].

2.1 FMUs and Contracts

Definition 1. *An FMU with identifier c is represented by the tuple*

$$\langle S_c, U_c, Y_c, \texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c \rangle ,$$

where: – S_c represents the state space; – U_c and Y_c the set of input and output variables, respectively; – $\texttt{set}_c : S_c \times U_c \times \mathcal{V} \to S_c$ and $\texttt{get}_c : S_c \times Y_c \to \mathcal{V}$ are functions to set the inputs and get the outputs, respectively (we abstract the set of values that each input/output variable can take as $\mathcal{V}$); and – $\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c$ is a function that instructs the FMU to compute its state after a given time step.

If an FMU is in state $s_c^{(n)}$ at time t, $\texttt{doStep}_c(s_c^{(n)}, H)$ approximates the state of the corresponding model at time $t+H$. The result of this approximation is encoded in state $s_c^{(n+1)}$. If the model is continuous, the FMU will internally approximate the evolution in the interval $[t, t+H]$, using an approximation function to estimate the values of the inputs in that interval. In this formalization, we leave this function implicit in the $\texttt{doStep}_c$, as reflected in the version 2.0 of the FMI Standard.

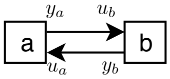

Fig. 1. Running Example.

Definition 2 (Scenario). *A scenario is a structure $\langle C, L \rangle$ where each identifier $c \in C$ is associated with an FMU, as defined in Definition 1, and $L(u) = y$ means that the output y is connected to input u. Let $U = \bigcup_{c \in C} U_c$ and $Y = \bigcup_{c \in C} Y_c$, then $L : U \to Y$. It is common to represent a co-simulation scenario as a diagram. For example, Figure 1 shows an example scenario with two FMUs, connected in a feedback loop.*

The following definitions correspond to the operations that are permitted in a co-simulation, and are correlated later in Definitions 8 to 10.

Definition 3 (Step). *Given a scenario $\langle C, L \rangle$, a co-simulation step, or just step, is a finite ordered sequence of FMU function calls $(f_i)_{i \in \mathbb{N}} = f_0, f_1, \ldots$ with $f_i \in F = \bigcup_{c \in C} \{\texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c\}$, and i denoting the order of the function call.*

Definition 4 (Initialization). *Given a scenario $\langle C, L \rangle$, we define the initialization procedure $(I_i)_{i \in \mathbb{N}}$ in the same way as a step, with $I_i \in F$.*

Definition 5 (Master). *Given a scenario $\langle C, L \rangle$, a step size H, a step $(f_i)_{i \in \mathbb{N}}$, and an initialization procedure $(I_i)_{i \in \mathbb{N}}$, a master algorithm is a structure defined as $\mathcal{A} = \langle C, L, H, (I_i)_{i \in \mathbb{N}}, (f_i)_{i \in \mathbb{N}} \rangle$.*

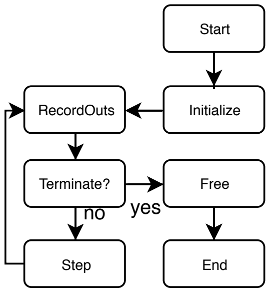

Fig. 2. Generic master.

Figure 2 shows the main steps of any master algorithm, and sheds light on the relationship between the initialization and step procedures.

Algorithms 1 to 3 are possible step procedures for the scenario introduced in Figure 1. We use the notation $s_c^{(0)}, s_c^{(1)}, \ldots$ to stress the transformations on the internal state of the FMU. The index is independent of the co-simulation time, so the state can undergo multiple transformations at the same co-simulation time.

Version 2.0 of the FMI standard [4] is not sufficiently rigorous to conclude whether any of the three algorithms is not a valid step. In fact, page 104 contains "There is the additional restriction in 'slaveInitialized' state that it is not allowed to call fmi2GetXXX functions after fmi2SetXXX functions without an fmi2DoStep call in between", invalidating Algorithms 2 and 3. However, this is contradicted by the fact that the standard supports *feed-through* dependencies, which induce algebraic dependencies between inputs and outputs. To quote the standard:

- "'output': The variable value can be used by another model or slave. The algebraic relationship to the inputs is defined via the dependencies attribute of `<fmiModelDescription><ModelStructure><Outputs><Unknown>`.", page 45.
- "Attribute dependencies defines the dependencies of the outputs from the knowns [...] at the current Communication Point (CoSimulation).", page 58.

The need for feed-through is also described in the scientific literature by the founders of the standard (e.g., [1, Fig. 3]). Since even the simplest mechanical systems, such as mass-spring-dampers, when coupled in a co-simulation, exhibit such feed-through effect [14], we are convinced that the statement on page 104 is incorrect. Therefore, Algorithms 1 to 3 satisfy the standard.

We now show that the results produced by each of these algorithms depend on the implementation of the FMUs. Regarding the feed-through of each FMU, assume that the output y_a depends instantaneously on, or has a feed-through from, the input u_a, and that the output y_b does not depend instantaneously on u_b. The instantaneously dependency condition can be expressed formally as: $\exists v, v' \in \mathcal{V}$,

[4] http://msdl.cs.mcgill.ca/people/claudio/projs/PrologCosimGeneration.zip

Algorithm 1

```
$s_a^{(1)} \leftarrow \texttt{doStep}_a(s_a^{(0)}, H)$
$s_b^{(1)} \leftarrow \texttt{doStep}_b(s_b^{(0)}, H)$
$v_a \leftarrow \texttt{get}_a(s_a^{(1)}, y_a)$
$v_b \leftarrow \texttt{get}_b(s_b^{(1)}, y_b)$
$s_b^{(2)} \leftarrow \texttt{set}_b(s_b^{(1)}, u_b, v_a)$
$s_a^{(2)} \leftarrow \texttt{set}_a(s_a^{(1)}, u_a, v_b)$
$s_a^{(0)} \leftarrow s_a^{(2)}$
$s_b^{(0)} \leftarrow s_b^{(2)}$
```

Algorithm 2

```
$s_b^{(1)} \leftarrow \texttt{doStep}_b(s_b^{(0)}, H)$
$s_a^{(1)} \leftarrow \texttt{doStep}_a(s_a^{(0)}, H)$
$v \leftarrow \texttt{get}_b(s_b^{(1)}, y_b)$
$s_a^{(2)} \leftarrow \texttt{set}_a(s_a^{(1)}, u_a, v)$
$v \leftarrow \texttt{get}_a(s_a^{(2)}, y_a)$
$s_b^{(2)} \leftarrow \texttt{set}_b(s_b^{(1)}, u_b, v)$
$s_a^{(0)} \leftarrow s_a^{(2)}$
$s_b^{(0)} \leftarrow s_b^{(2)}$
```

Algorithm 3

```
$s_b^{(1)} \leftarrow \texttt{doStep}_b(s_b^{(0)}, H)$
$v \leftarrow \texttt{get}_b(s_b^{(1)}, y_b)$
$s_a^{(1)} \leftarrow \texttt{set}_a(s_a^{(0)}, u_a, v)$
$v \leftarrow \texttt{get}_a(s_a^{(1)}, y_a)$
$s_b^{(2)} \leftarrow \texttt{set}_b(s_b^{(1)}, u_b, v)$
$s_a^{(2)} \leftarrow \texttt{doStep}_a(s_a^{(1)}, H)$
$s_a^{(0)} \leftarrow s_a^{(2)}$
$s_b^{(0)} \leftarrow s_b^{(2)}$
```

Fig. 3. Three algorithms conforming to the FMI Standard (version 2.0). The last two lines represent the assignment of the new state to the state to be used in the next co-simulation step.

such that $s_a^{(1)} = \texttt{set}_a(s_a^{(0)}, u_a, v), s_a^{(2)} = \texttt{set}_a(s_a^{(0)}, u_a, v')$, and $\texttt{get}_a(s_a^{(1)}, y_a) \neq \texttt{get}_a(s_a^{(2)}, y_a)$. With these suppositions, Algorithm 1 is inadequate, because the value of y_a can only be computed after the value of u_a is known.

The feed-through information is a piece of information about the implementation of the FMU that allows us to code master algorithms that produce better results. It is natural to wonder whether there are other aspects that we can use to distinguish Algorithms 2 and 3.

Comparing Algorithms 2 and 3, one notices that, in Algorithm 3, the input u_a is set after $\texttt{doStep}_b$ is invoked. This means that, FMU b advances in time, produces an input to FMU a, and only after does FMU a catch up to the time that b is in. Indeed, this is the main difference between a Gauss-Seidel master algorithm, and a Jacobi one. These are well known algorithms in the literature, and accepted as being compatible with the FMI Standard [2].

However, the implementation of FMU a in Algorithm 3 must differ from the implementation of FMU a in Algorithm 2. To see why, suppose that a does a linear *interpolation* of its inputs. An interpolation formula between two given inputs v_t and v_{t+H}, expected at times t and $t + H$, respectively, is given by $\tilde{u_a}(t + \mathit{\Delta} t) = v_t + \frac{v_{t+H} - v_t}{H} \mathit{\Delta} t$. In contrast, an extrapolation between two given inputs v_{t-H} and v_t, expected at times $t - H$ and t, is given by $\tilde{u_a}(t + \mathit{\Delta} t) = v_t + \frac{v_t - v_{t-H}}{H} \mathit{\Delta} t$. Note the difference between the two formulas and the expected timestamps of the inputs. Since the timestamps of the inputs are implicit in the FMI Standard, the same FMU will either implement an interpolation, or an extrapolation, but cannot implement both.

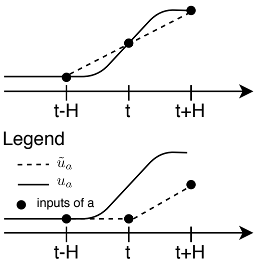

Fig. 4. Application of an interpolation formula to inputs with the wrong timestamps.

If FMU a implements an interpolation, and is used in Algorithm 2, then the result will be a delayed input approximation of u_a. This is because an interpolation will given the wrong inputs, for instance, with timestamps $t - H$ and t, instead of t and $t + H$. The result is illustrated Figure 4. For purely continuous systems, this delay may not introduce substantial errors. However, as has been shown in [20], in systems with discontinuities, the delay can propagate to trigger abrupt changes in the systems' behavior.

We now formalize the contracts over the outputs of the FMU (input/output feed-through), and the contracts over the inputs (interpolation or extrapolation). We use the more generic term *reactivity* to the contracts over the inputs because it can be used for purposes other than input approximation implementations. For example, a software FMU may not implement a linear input interpolation, but still be reactive to reflect the fact that it runs with a very short sampling interval (short relative to co-simulation step size).

Definition 6 (Feed-through). *The input $u_c \in U_c$ feeds through to output $y_c \in Y_c$, that is, $(u_c, y_c) \in D_c$, when there exists $v_1, v_2 \in \mathcal{V}$ and $s_c \in S_c$, such that $\mathtt{get}_c(\mathtt{set}_c(s_c, u_c, v_1), y_c) \neq \mathtt{get}_c(\mathtt{set}_c(s_c, u_c, v_2), y_c)$.*

Definition 7 (Reactivity). *For a given FMU c with input $u_c \in U_c$, $R_c(u_c) =$ true if the function $\mathtt{doStep}_c$ assumes that the input u_c comes from a FMU that has advanced forward relative to FMU c.*

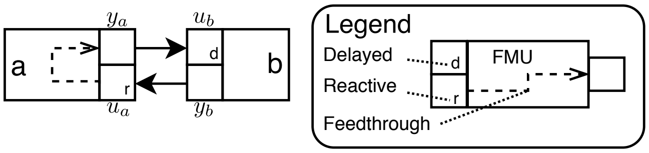

Fig. 5. Contracts notation.

The feed-through and reactivity information for the scenario introduced in Figure 1 can be represented visually as in Figure 5, i.e., FMU a is reactive and has feed-through, whereas FMU b is delayed.

2.2 Master Algorithms

In order to determine whether a given master algorithm satisfies a given scenario and contracts, we need to formalize the constraints that the contracts impose on the valid master algorithms. For that, we need to formalize the run-time state of each FMU, and how the invocation of each co-simulation operation evolves this state.

Definition 8 (Run-time State). *Given an FMU c as defined in Definition 1, the run-time state of c is a member of the set $S_c^R = \mathbb{R}_{\geq 0} \times S_{U_c}^R \times S_{Y_c}^R$, where $\mathbb{R}_{\geq 0}$ is the time base, $S_{U_c}^R = \prod_{u_c \in U_c} S_{u_c}^R$ represents the aggregated state set of* *the input ports,* $S^R_{u_c} = \mathbb{R}_{\geq 0} \times \{\mathit{defined}, \mathit{undefined}\}$ *represents the set of states of an input port* $u_c \in U_c$*,* $S^R_{Y_c} = \prod_{y_c \in Y_c} S^R_{y_c}$ *represents the aggregated state set of the output ports, and* $S^R_{y_c} = \mathbb{R}_{\geq 0} \times \{\mathit{defined}, \mathit{undefined}\}$*.*

Note that the run-time state of an FMU c differs from the state of the FMU. The later belongs to the state space S_c, while the former is defined next. Moreover, note that each port has its own timestamp, the reason of which will become clear when we define how each co-simulation operation changes the run-time state of the co-simulation.

Definition 9 (Co-simulation State). *Given a co-simulation scenario* $\langle C, L\rangle$*, as defined in Definition 2, the co-simulation state is a member of the set* $S^R_C = \prod_{c \in C} S^R_c$*.*

For the scenario introduced in Figure 1, the run-time state set is $S^R_a \times S^R_b$, with $S^R_a = \mathbb{R}_{\geq 0} \times S^R_{u_a} \times S^R_{y_a}$, and $S^R_b = \mathbb{R}_{\geq 0} \times S^R_{u_b} \times S^R_{y_b}$. Before initialization (recall Figure 2), every port has not yet been defined, and the timestamp of the ports and FMUs is 0. To improve readability, we will use a visual notation to represent the state, as illustrated in Figure 6.

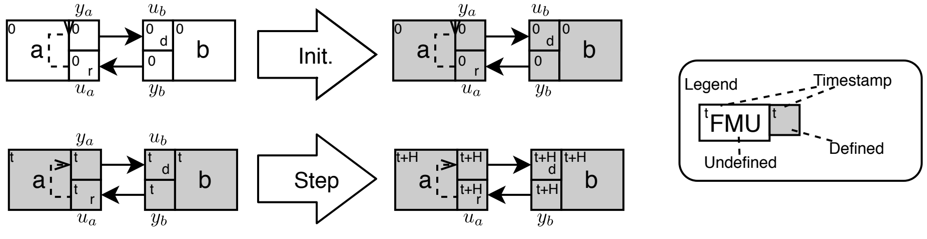

Fig. 6. Visual representation of the state before and after initialization/step.

Definition 10 (Consistent State). *A co-simulation run-time state* S^R_C *is consistent when all FMUs and ports have the same timestamp, and all ports are either defined, or undefined. We shall use the notation* $\mathit{Consistent}(S^R_C) = (\mathit{defined}, t)$ *or* $\mathit{Consistent}(S^R_C) = (\mathit{undefined}, t)$*, when such is the case, respectively.*

Using these definitions and notation, we can summarize the purpose of the initialization process, depicted in Figure 2, and defined in Definition 4: to take a run-time state S^R_C where $\mathit{Consistent}(S^R_C) = (\mathit{undefined}, 0)$, and transform it into a run-time state $S^{R'}_C$ where $\mathit{Consistent}(S^{R'}_C) = (\mathit{defined}, 0)$. Similarly, one can summarize the purpose of the step process, defined in Definition 3: to take a run-time state S^R_C where $\mathit{Consistent}(S^R_C) = (\mathit{defined}, t)$, and transform it into a run-time state $S^{R'}_C$ where $\mathit{Consistent}(S^{R'}_C) = (\mathit{defined}, t+H)$. Figure 6 illustrates this for the example in Figure 1.

We use structural operational semantics (SOS) notation to represent the run-time state evolution rules.

Definition 11. *Given a scenario $\langle C, L \rangle$, a set of contracts $\mathcal{C} = \bigcup_{c \in C} \{(R_c, D_c)\}$ a finite sequence of operations $(f_i)_{i \in \mathbb{N}} = f_0, f_1, \ldots$, with $f_i \in F$ as used in Definitions 3 and 4, and a run-time state S_C^R as in Definition 9, we define the application of (f_i) to S_C^R in SOS as*

$$\frac{\left\langle C; L; \mathcal{C}; S_C^R; f_0 \right\rangle \Rightarrow S_C^{R'}}{\left\langle C; L; \mathcal{C}; S_C^R; f_0, f_1, \ldots \right\rangle \rightarrow \left\langle C; L; \mathcal{C}; S_C^{R'}; f_1, \ldots \right\rangle}$$

$$\frac{\left\langle C; L; \mathcal{C}; S_C^R; f \right\rangle \Rightarrow S_C^{R'}}{\left\langle C; L; \mathcal{C}; S_C^R; f \right\rangle \rightarrow \left\langle C; L; \mathcal{C}; S_C^{R'}; \emptyset \right\rangle}$$

The following definitions detail the $\Rightarrow$ reduction (not to be confused with the application operation $\rightarrow$, specified in Definition 11). Examples of this reduction are shown in Figure 7. We will use the notation $\_$ for variables that need not be named.

Definition 12 (Output Computation). *The reduction*

$$\left\langle C; L; \mathcal{C}; S_C^R; \mathtt{get}_c(\_, y_c) \right\rangle \Rightarrow S_C^{R'}$$

represents the effect on the run-time state of operation $\mathtt{get}_c(\_, y_c)$. The reduction is valid if, in S_C^R, all inputs that feed-through to y_c are defined and have the same timestamp t. In that case, $S_C^{R'}$ is obtained by setting the run-time state of y_c in S_C^R to defined with timestamp t.

Definition 13 (Input Computation). *The reduction*

$$\left\langle C; L; \mathcal{C}; S_C^R; \mathtt{set}_c(\_, u_c, v) \right\rangle \Rightarrow S_C^{R'}$$

represents the effect on the run-time state of operation $\mathtt{set}_c(\_, u_c, v)$. The reduction is valid if, in S_C^R, all outputs connected to u_c are defined and have the same timestamp t. In that case, $S_C^{R'}$ is obtained by setting the run-time state of u_c in S_C^R to defined with timestamp t.

Definition 14 (Step Computation). *The reduction*

$$\left\langle C; L; \mathcal{C}; S_C^R; \mathtt{doStep}_c(\_, H) \right\rangle \Rightarrow S_C^{R'}$$

represents the effect on the run-time state of operation $\mathtt{doStep}_c(\_, H)$. Let t denote the timestamp of c in S_C^R. The success of this reduction depends on satisfying all the following conditions:

- *For every input port u_c that has a non-reactive contract in $\mathcal{C}$, S_C^R must contain the state of u_c as defined at timestamp t.*
- *For every input port u_c that has a reactive contract in $\mathcal{C}$, S_C^R must contain the state of u_c as defined at timestamp $t + H$.*

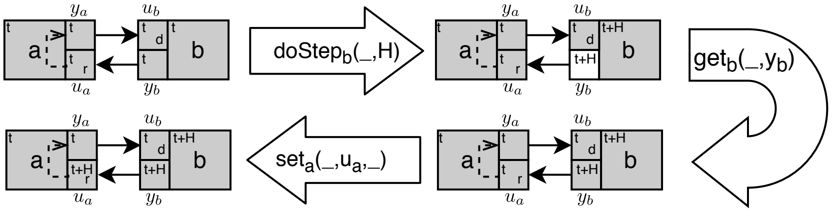

Fig. 7. Example application of operations on the run-time state.

If such conditions hold, then $S_C^{R'}$ is obtained by setting the run-time state of c, and all its output ports, to timestamp $t + H$, and by setting all the output ports to undefined.

Figure 7 shows the application of the first three operations of Algorithm 3 on the run-time state of the co-simulation at the beginning of a step.

Our problem can now be formalized.

Problem 1. Given a scenario $\langle C, L\rangle$, and the set of contracts $\mathcal{C} = \bigcup_{c\in C} \{(R_c, D_c)\}$, generate a master algorithm

$$\mathcal{A} = \left\langle C, L, H, (I_i)_{i\in\mathbb{N}}, (f_i)_{i\in\mathbb{N}}\right\rangle,$$

such that:

$$\left\langle C; L; \mathcal{C}; S_C^R; (I_i)_{i\in\mathbb{N}}\right\rangle \to^* \left\langle C; L; \mathcal{C}; S_C^{R(0)}; \emptyset\right\rangle \text{ and}$$
$$\left\langle C; L; \mathcal{C}; S_C^{R(j)}; (f_i)_{i\in\mathbb{N}}\right\rangle \to^* \left\langle C; L; \mathcal{C}; S_C^{R(j+1)}; \emptyset\right\rangle,$$

where S_C^R, $S_C^{R(0)}$, $S_C^{R(j)}$, and $S_C^{R(j+1)}$, are such that

$$\begin{aligned} &\mathit{Consistent}(S_C^R) = (\mathit{undefined}, 0), \quad &&\mathit{Consistent}(S_C^{R(0)}) = (\mathit{defined}, 0),\\ &\mathit{Consistent}(S_C^{R(j)}) = (\mathit{defined}, t), \quad &&\mathit{Consistent}(S_C^{R(j+1)}) = (\mathit{defined}, t + H).\end{aligned}$$

3 Generation of Co-simulation Algorithms

In this section, we propose a graph-based master generation algorithm, with complexity that is linear in the number of ports and FMUs in a given co-simulation scenario. We will focus on the generation of the step procedure, as the generation of the initialization procedure can be easily derived. The key insight in our contribution is the following.

Proposition 1. *For each $c \in C$ of a given co-simulation scenario:* `doStep`$_c(\_, H)$ *needs to be executed once, and only once; for each $y_c \in Y_c$,* `get`$_c(\_, y_c)$ *needs to be executed once; and for each $u_c \in U_c$,* `set`$_c(\_, u_c, \_)$ *needs to be executed once.*

Proposition 1 allows us to build a graph representing every operation that might be executed in a step procedure. The edges of this graph represent precedence constraints, and a topological sorting of the graph yields a valid step procedure. We provide a proof sketch of this claim in Section 3.1.

Definition 15 (Step Operation Graph). *Given a co-simulation scenario $\langle C, L\rangle$, and a set of contracts $\mathcal{C} = \bigcup_{c \in C} \{(R_c, D_c)\}$, we define the step operation graph where each node represents an operation $\texttt{set}_c(\_, u_c, \_)$, $\texttt{doStep}_c(\_, H)$, or $\texttt{get}_c(\_, y_c)$, of some fmu $c \in C$, $y_c \in Y_c$, and $u_c \in U_c$. The edges are created according to the following rules:*

1. *For each $c \in C$ and $u_c \in U_c$, if $L(u_c) = y_d$, add an edge $\texttt{get}_d(\_, y_d) \rightarrow \texttt{set}_c(\_, u_c, \_)$;*
2. *For each $c \in C$ and $y_c \in Y_c$, add an edge $\texttt{doStep}_c(\_, H) \rightarrow \texttt{get}_c(\_, y_c)$;*
3. *For each $c \in C$ and $u_c \in U_c$, if $R_c(u_c) = true$, add an edge $\texttt{set}_c(\_, u_c, \_) \rightarrow \texttt{doStep}_c(\_, H)$;*
4. *For each $c \in C$ and $u_c \in U_c$, if $R_c(u_c) = false$, add an edge $\texttt{doStep}_c(\_, H) \rightarrow \texttt{set}_c(\_, u_c, \_)$;*
5. *For each $c \in C$ and $(u_c, y_c) \in D_c$, add an edge $\texttt{set}_c(\_, u_c, \_) \rightarrow \texttt{get}_c(\_, y_c)$.*

Figure 8 shows an example graph, constructed from the example in Figure 5.

3.1 Correctness

We now provide a proof sketch of the claim that a topological sorting of the above defined graph will yield a valid step procedure. The proof is divided into two parts. The first part proves that each operation is invoked in the correct order, with respect the conditions for its execution, detailed in Definitions 12 to 14. The second part proves Proposition 1, which essentially means that the graph is complete. The proof sketch also establishes that the successive application of the operations, in the topological order, satisfy the conditions in problem 1, thereby transforming a consistently defined run-time state with timestamp t into a consistently defined run-time state at time $t + H$.

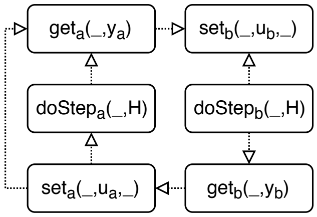

Fig. 8. Example step operation graph.

We will assume a given non-trivial co-simulation scenario. A non-trivial scenario contains at least two fmus, every input is connected to an output, there are no self-connections, and it is possible to construct a topological ordering of the step operation graph.

Ordering We now sketch the proof that each operation in the topological ordering satisfies the conditions for its execution, as detailed in Definitions 12 to 14.

Given a non-trivial scenario $\langle C, L\rangle$, and a set of contracts $\mathcal{C}$, consider a topological ordering $(f_i)_{i \in \mathbb{N}}$ of the graph constructed as in Definition 15, and let us prove the correct ordering by induction on i. Let S^R_C be the given run-time state before the step procedure is executed. It satisfies $\mathit{Consistent}(S^R_C) = (\mathit{defined}, t)$ for some t.

$i = 1$. Let us now consider the form of f_1:

- $f_1 = \mathtt{get}_c(\_, y_c)$, for some $c \in C$ and $y_c \in Y_c$. This case is impossible, as every output operation in c must be preceded by a step operation of c.
- $f_1 = \mathtt{set}_c(\_, u_c, \_)$, for some $c \in C$ and $u_c \in U_c$. This case is impossible as every input operation is preceded by at least one output operation.
- $f_1 = \mathtt{doStep}_c(\_, H)$, for some $c \in C$. This case is possible, and every input of c is delayed (otherwise there would be no topological sort). Since every input u_c is delayed, and since $\mathit{Consistent}(S^R_C) = (\mathit{defined}, t)$, the conditions in Definition 14 hold.

$i > 1$. Let $S^{R^{(i-1)}}_C$ denote the run-time state after the (successful) invocation of operations $f_1, \ldots, f_{i-1}$ in the topological order. Let us now consider the form of f_i:

- $f_i = \mathtt{get}_c(\_, y_c)$, for some $c \in C$ and $y_c \in Y_c$. In this case, according to Definition 15, it must be the case that: $\mathtt{doStep}_c(\_, H)$ has been invoked successfully; $\mathtt{set}_c(\_, u_c, \_)$ has been invoked successfully, for each $(u_c, y_c) \in D_c$; and, the timestamp of y_c in $S^{R^{(i-1)}}_C$ is $t+H$. Since S^R_C contains all inputs defined at timestamp t, the only reason to invoke $\mathtt{set}_c(\_, u_c, \_)$ on any input u_c is to set its timestamp to $t + H$. Therefore, every $(u_c, y_c) \in D_c$ has the same timestamp $t + H$. This satisfies the conditions in Definition 12.
- $f_i = \mathtt{set}_c(\_, u_c, \_)$, for some $c \in C$ and $u_c \in U_c$. With a similar argument to the previous case, we conclude that every y_d connected to u_c is defined and has the same timestamp $t + H$.
- $f_i = \mathtt{doStep}_c(\_, H)$, for some $c \in C$. In this case, according to Definition 15, we know that: $\mathtt{set}_c(\_, u_c, \_)$ has been invoked successfully, for each $u_c \in U_c$ such that $R_c(u_c) = \mathit{true}$; and, $\mathtt{set}_c(\_, u_c, \_)$ has not been invoked yet, for each $u_c \in U_c$ such that $R_c(u_c) = \mathit{false}$. Therefore, for every $u_c \in U_c$, if $R_c(u_c) = \mathit{true}$, then its timestamp is $t + H$, and if $R_c(u_c) = \mathit{false}$ then its timestamp is t. This satisfies the conditions in Definition 14.

Since all possible options are satisfied, the topological ordering is correct.

Completeness (Proof of Proposition 1) Given a non-trivial scenario $\langle C, L \rangle$, and a set of contracts $\mathcal{C}$, consider a topological ordering $(f_i)_{i \in \mathbb{N}} = f_1, \ldots, f_N$ of the graph constructed as in Definition 15. Let S^R_C be the given run-time state before the step procedure is executed. It satisfies $\mathit{Consistent}(S^R_C) = (\mathit{defined}, t)$ for some t. Let us define a function that assigns a natural number to each run-time state: $\mathit{Remaining}(x, t)$ is the number of ports and fmus whose timestamp is less than t. For example, $\mathit{Remaining}(S^R_C, t + H)$ denotes the number of ports and the number of FMUs in the scenario $\langle C, L \rangle$, and $\mathit{Remaining}(S^R_C, t) = 0$. Let $S^{R^{(i-1)}}_C$ denote the run-time state after the (successful) invocation of operations $f_1, \ldots, f_{i-1}$ in the topological order. Now we show by induction on i that

$\mathit{Remaining}(S_C^{R(i)}, t+H) = N - i$. This will establish that $\mathit{Remaining}(S_C^{R(N)}, t+H) = 0$, which implies that $\mathit{Consistent}(S_C^{R(N)}) = (\mathit{defined}, t+H)$.

$i = 1$. As established in Section 3.1, $f_1 = \texttt{doStep}_c(\_, H)$, and the conditions in Definition 14 hold. Therefore, the operation is executed and the timestamp of c becomes $t + H$ in $S_C^{R(1)}$.

$i > 1$. Assume that $\mathit{Remaining}(S_C^{R(i-1)}, t+H) = N - (i-1)$. As in Section 3.1, every possible form of f_i can be executed. Hence, let us consider the effects of each execution in turn:

- $f_i = \texttt{get}_c(\_, y_c)$, for some $c \in C$ and $y_c \in Y_c$. In this case, it is easy to see that the $\texttt{get}_c(\_, y_c)$ operation has not been executed before, therefore the timestamp of y_c in $S_C^{R(i-1)}$ is t. After execution, the timestamp of y_c in $S_C^{R(i)}$ is $t+H$. Therefore, $\mathit{Remaining}(S_C^{R(i)}, t+H) = \mathit{Remaining}(S_C^{R(i-1)}, t+H) + 1 = N - i$.
- $f_i = \texttt{set}_c(\_, u_c, \_)$, for some $c \in C$ and $u_c \in U_c$. The argument is similar to above.
- $f_i = \texttt{doStep}_c(\_, H)$, for some $c \in C$. The argument is similar to above.

This concludes our proof sketch for the completeness of the graph based approach to generating master algorithms.

3.2 Optimization

In this section, we describe a simple optimization to the topological ordering that leverages the fact that the FMI standard allows multiple ports to be set and get in bulk. Essentially, we have modified the topological ordering procedure to group operations that can be executed *in parallel*. If multiple set or get operations on the same FMU belong to the same group, then these can be merged into a single operation.

The correctness of this optimization can be established by noting that this grouping procedure amounts to representing all possible topological orderings, and Section 3.1 establishes that each ordering is correct. The next section shows an example application of this procedure.

3.3 Application

Our contribution has been implemented in Prolog and is available online[5]. Consider the scenario in Figure 9. The optimized generated step procedure is shown in Algorithm 4.

[5] http://msdl.cs.mcgill.ca/people/claudio/projs/PrologCosimGeneration.zip

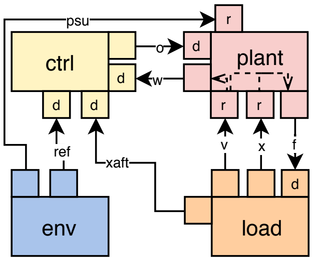

Fig. 9. Case study scenario. Based on [12].

Algorithm 4

```
$s^{(1)}_{load} \leftarrow \texttt{doStep}_{load}(s^{(0)}_{load}, H)$
$s^{(1)}_{env} \leftarrow \texttt{doStep}_{env}(s^{(0)}_{env}, H)$
$[v_x, v_v, v_{xaft}] \leftarrow \texttt{get}_{load}(s^{(1)}_{load}, [x, v, xaft])$
$[v_{psu}, v_{ref}] \leftarrow \texttt{get}_{env}(s^{(1)}_{env}, [psu, ref])$
$s^{(1)}_{plant} \leftarrow \texttt{set}_{plant}(s^{(0)}_{plant}, [psu, x, v], [v_{psu}, v_x, v_v]),$
$s^{(2)}_{plant} \leftarrow \texttt{doStep}_{plant}(s^{(1)}_{plant}, H),$
$[v_w, v_f] \leftarrow \texttt{get}_{plant}(s^{(2)}_{plant}, [w, f]),$
$s^{(1)}_{ctrl} \leftarrow \texttt{set}_{ctrl}(s^{(0)}_{ctrl}, w, v_w)$
$s^{(2)}_{load} \leftarrow \texttt{set}_{load}(s^{(1)}_{load}, f, v_f),$
$s^{(2)}_{ctrl} \leftarrow \texttt{doStep}_{ctrl}(s^{(1)}_{ctrl}, H),$
$v_o \leftarrow \texttt{get}_{ctrl}(s^{(2)}_{ctrl}, o)$
$s^{(3)}_{ctrl} \leftarrow \texttt{set}_{ctrl}(s^{(2)}_{ctrl}, [ref, xaft]),$
$s^{(3)}_{plant} \leftarrow \texttt{set}_{plant}(s^{(2)}_{plant}, o, v_o),$
$s^{(0)}_{plant} \leftarrow s^{(3)}_{plant}, s^{(0)}_{load} \leftarrow s^{(2)}_{load}$
$s^{(0)}_{env} \leftarrow s^{(1)}_{env}, s^{(0)}_{ctrl} \leftarrow s^{(3)}_{ctrl}$
```

4 Related Work

The closest work to our own is reported in [21], where a formalization of the semantics of FMI is proposed. However, our work differs in two key aspects: first, the objective of [21] is to prove properties about the system being co-simulated, whereas our goal is to guarantee certain basic properties of the co-simulation; second, the aforementioned work does not accommodate for simulator contracts, but includes the rollback operation. Ongoing work is revising the contracts and semantics in order to accommodate the rollback operation.

Prior work [5–8, 15] is focused on the correct synchronization of a discrete event simulator with a continuous one. This seminal work assumes a standard synchronization algorithm, where, in the presence of possible state events, the discrete simulator is always one step behind the continuous simulator, to avoid rollbacks. This is an example of reactive simulator contract. Ongoing work is exploring how to accommodate step rejection in the simulator contracts, to allow for hybrid co-simulation master algorithms.

Instead of enforcing a correct synchronization, some work has focused on finding the maximum allowed delay in the event detection. For instance, the work in [9] explores how the energy of a hybrid system can be increased when state events are not accurately reproduced by the co-simulation. It presents a way to find the largest co-simulation step that prevents this from happening.

5 Conclusion

Driven by the need to make explicit information regarding the implementation of black box simulators, we have built on a prior formalization of the FMI standard to derive a procedure that generates master algorithms that respect such implementations. This algorithm consists of constructing a topological ordering of a precedence graph. An optimization, whereby input/output operations on the same simulator are clustered together, was proposed.

The key insight that makes this algorithm possible is the fact that each co-simulation operation on each port and fmu needs to be executed only once. In the future, when we consider more complex master algorithms, such as the ones supporting rollback and step size adaptation operations, this will no longer be true, and more research will be required to efficiently derive these master algorithms.

Our algorithm was applied successfully to an industrial case, developed in prior work [12]. The code to reproduce the experiments in this paper is available for download[6].

References

1. Arnold, M., Clauss, C., Schierz, T.: Error Analysis and Error Estimates for Co-Simulation in FMI for Model Exchange and Co-Simulation V2.0 **LX**(1), 75. https://doi.org/10.2478/meceng-2013-0005
2. Bastian, J., Clauß, C., Wolf, S., Schneider, P.: Master for Co-Simulation Using FMI. In: 8th International Modelica Conference. pp. 115–120. Linköping University Electronic Press, Linköpings universitet. https://doi.org/10.3384/ecp11063115
3. Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In: 9th International Modelica Conference. pp. 173–184. Linköping University Electronic Press. https://doi.org/10.3384/ecp12076173
4. FMI: Functional Mock-up Interface for Model Exchange and Co-Simulation
5. Gheorghe, L., Bouchhima, F., Nicolescu, G., Boucheneb, H.: Formal Definitions of Simulation Interfaces in a Continuous/Discrete Co-Simulation Tool. In: Rapid System Prototyping, 2006. Seventeenth IEEE International Workshop On. pp. 186–192. https://doi.org/10.1109/RSP.2006.18
6. Gheorghe, L., Bouchhima, F., Nicolescu, G., Boucheneb, H.: A Formalization of Global Simulation Models for Continuous/Discrete Systems. In: Summer Computer Simulation Conference. pp. 559–566. Society for Computer Simulation International San Diego, CA, USA, series Title: SCSC '07
7. Gheorghe, L., Bouchhima, F., Nicolescu, G., Boucheneb, H.: Semantics for Model-based Validation of Continuous/Discrete Systems. In: Design, Automation and Test in Europe. pp. 498–503. ACM. https://doi.org/10.1145/1403375.1403493, series Title: DATE '08
8. Gheorghe, L., Nicolescu, G., Boucheneb, H.: Semantics for Rollback-Based Continuous/Discrete Simulation. In: Behavioral Modeling and Simulation Workshop, 2008. BMAS 2008. IEEE International. pp. 106–111. https://doi.org/10.1109/BMAS.2008.4751250
9. Gomes, C., Karalis, P., Navarro-López, E.M., Vangheluwe, H.: Approximated Stability Analysis of Bi-modal Hybrid Co-simulation Scenarios. In: 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems. pp. 345–360. Springer, Cham. https://doi.org/10.1007/978-3-319-74781-1_24

[6] http://msdl.cs.mcgill.ca/people/claudio/projs/PrologCosimGeneration.zip

10. Gomes, C., Lucio, L., Vangheluwe, H.: Semantics of Co-simulation Algorithms with Simulator Contracts. p. submitted
11. Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., De Meulenaere, P.: Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators **95**(3), 1–29. https://doi.org/10.1177/0037549718759775
12. Gomes, C., Oakes, B.J., Moradi, M., Gamiz, A.T., Mendo, J.C., Dutre, S., Denil, J., Vangheluwe, H.: HintCO - Hint-Based Configuration of Co-Simulations. In: International Conference on Simulation and Modeling Methodologies, Technologies and Applications. p. accepted
13. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A Survey **51**(3), Article 49. https://doi.org/10.1145/3179993
14. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: State of the art, http://arxiv.org/abs/1702.00686
15. Iugan, L.G., Boucheneb, H., Nicolescu, G.: A generic conceptual framework based on formal representation for the design of continuous/discrete co-simulation tools **19**(3), 243–275. https://doi.org/10.1007/s10617-014-9156-3
16. Kübler, R., Schiehlen, W.: Two Methods of Simulator Coupling **6**(2), 93–113. https://doi.org/10.1076/1387-3954(200006)6:2;1-M;FT093
17. Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J.P., Posch, A., Nouidui, T.: An empirical survey on co-simulation: Promising standards, challenges and research needs **95**, 148–163. https://doi.org/10.1016/j.simpat.2019.05.001
18. Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J., Posch, A., Nouidui, T.: Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers. In: The American Modelica Conference. pp. 138–146. Linköping University Electronic Press, Linköpings universitet. https://doi.org/10.3384/ecp18154138
19. Schweizer, B., Li, P., Lu, D.: Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches **10**(5), 051007. https://doi.org/10.1115/1.4028503
20. Thule, C., Gomes, C., Deantoni, J., Larsen, P.G., Brauer, J., Vangheluwe, H.: Towards Verification of Hybrid Co-simulation Algorithms. In: Workshop on Formal Co-Simulation of Cyber-Physical Systems. Springer, Cham. https://doi.org/10.1007/978-3-030-04771-9_1
21. Zeyda, F., Ouy, J., Foster, S., Cavalcanti, A.: Formalising Cosimulation Models. In: Cerone, A., Roveri, M. (eds.) Software Engineering and Formal Methods. vol. 10729, pp. 453–468. Springer International Publishing. https://doi.org/10.1007/978-3-319-74781-1_31