HintCO – Hint-based Configuration of Co-simulations

Cláudio Gomes[1,2], Bentley James Oakes[1,2], Mehrdad Moradi[1,2],
Alejandro Torres Gámiz[3], Juan Carlos Mendo[3], Stefan Dutré[4], Joachim Denil[1,2]
and Hans Vangheluwe[1,2]
[1]*University of Antwerp, Belgium*
[2]*Flanders Make vzw, Belgium*
[3]*Boeing Research & Technology Europe, Madrid, Spain*
[4]*Siemens, Belgium*

Keywords: Co-simulation, Hints, Functional Mockup Interface Standard, Semantic Adaptation, Configuration.

Abstract: Simulation-based analyses of Cyber-Physical Systems are fundamental in industrial design and testing approaches. The utility of analyses relies on the correct configuration of the simulation tools, which can be highly complicated. System engineers can normally judge the results, and either evaluate multiple simulation algorithms, or change the models. However, this is not possible in a co-simulation approach. Co-simulation is a technique to perform full-system simulation, by combining multiple black-box simulators, each responsible for a part of the system. In this paper, we demonstrate the difficulty of correctly configuring a co-simulation scenario using an industrial case study. We propose an approach to tackle this challenge by allowing multiple engineers, specialized in different domains, to encode some of their experience in the form of hints. These hints, together with state-of-the-art best practices, are then used to semi-automatically guide the configuration process of the co-simulation. We report the application of this approach to a use case proposed by our industrial partners, and discuss some of the lessons learned.

1 INTRODUCTION

Many industries, including the aeronautical domain, increasingly design and test Cyber-Physical Systems (CPS), which marry the complexities of software with the realities of the physical world (Lee, 2008). A major issue in CPS is that components can span multiple domains, which may require their own simulation tools, knowledge of how to configure those simulators, and the ability to interpret the results.

Co-simulation is a technique to combine multiple simulators, each responsible for a model, in order to compute the behavior of the combined models over time (Kübler and Schiehlen, 2000). The simulators are coupled using a *master algorithm* that communicates with each simulator via its interface, which comprises functions for setting/getting inputs/outputs, and computing the associated model behavior over a given interval of time. An example of such an interface is the Functional Mockup Interface (FMI) Standard (FMI, 2014, Blockwitz et al., 2012).

By allowing sub-systems to be modelled and simulated with the best formalisms and solvers, and taking a black-box approach to interact with these sub-models, co-simulation is a promising solution to serve the needs of more complex engineering processes, as described in Section 2.

However, the black-box nature of co-simulation amplifies the difficulties in ensuring that the results can be trusted. Similarly to other simulation-based analyses, there is often no reference solution to judge the co-simulation results against. Moreover, co-simulation is typically used as part of an optimization loop (e.g., design space exploration), physical system integration analysis (e.g., hardware-in-the-loop), and/or impact analysis of sub-model refinements. In any of these use cases, the coupled model(s) being simulated is frequently changed. Therefore, the person interested in the results of the analysis, henceforth denoted as the user, may be unable to properly configure each individual co-simulation.

When a co-simulation result is incorrect (see Section 3 for a more rigorous definition), there can be multiple causes (Gomes et al., 2018c, Schweizer et al.,

Gomes, C., Oakes, B., Moradi, M., Gámiz, A., Mendo, J., Dutré, S., Denil, J. and Vangheluwe, H.
HintCO – Hint-based Configuration of Co-simulations.
DOI: 10.5220/0007830000570068
In *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2019)*, pages 57-68
ISBN: 978-989-758-381-0

2015, Arnold et al., 2014): (i) Sub-models are incorrect; (ii) FMUs use the incorrect simulation algorithm; and (iii) The master algorithm is incorrect. This means that the user has to be familiar with a wide range of domains in order to correctly configure the co-simulation. For example, a software controller model should be interacted with in a way that reflects how the actual software will be interacted with, such that the sample rate and its causality should be respected. Further examples are given in Section 2.

A recent survey (Schweiger et al., 2018) corroborates prior research work (see Section 6) in defining the main challenge: *users do not always know how to configure the co-simulation*. This challenge is formalized in Section 3.

In this work, we propose a way to tackle this challenge which is motivated by discussions with our industrial partners. We hypothesize that while users may not know how to configure the co-simulation, the engineers have intuition about the behavior of the system and when the simulation's result is not correct. Hence, even when there is no reference solutions, engineers can usually tell when a result is not correct.

Our concrete contributions are: a language to describe hints, i.e., properties about the co-simulation scenario; and a framework, called *HintCO*, that uses those hints to propose co-simulation master algorithms that are good candidates to produce correct results.

The implementation of the framework is available online[1].

1.1 Overview of HintCO Framework

There are three main components:

a) HintCO Hint Language: which allows the user to specify their intuition about the system (Section 4). Common hints are provided in a built-in library so the user may easily choose and adapt them to a specific co-simulation.

b) Generation of Candidate Master Algorithms: which is the method for mapping a given set of hints to sets of master algorithms (Sections 5.1 and 5.2). In short, the hints provided induce a *search space* of possible master algorithms, and a ranking of the most important features for a good master algorithm.

c) Execution of the Master Algorithms: where each master algorithm produced by the search is executed (Section 5.3). The results are then presented to the user for inspection.

In Section 6, we discuss other approaches that complement our own, and Section 7 summarizes of our research and the steps to extend our framework further.

[1] https://msdl.uantwerpen.be/git/claudio/HintCO

2 INDUSTRIAL EXAMPLE

This section describes the added value of co-simulation for our industrial partners. Then, we introduce the case study made available by Boeing and illustrate the challenge of finding the correct configuration for the co-simulation. Finally, we argue that to know which configuration is likely to be the best, we need domain knowledge.

2.1 Value of Co-simulation for Boeing

Boeing's vision on the Digital Twin era of aviation involves the integration of models coming from different physical domains, software environments and numerical characteristics into a single virtualized aircraft (Boeing, 2017). This vision requires the creation of unified modeling environments, where engineers can seamlessly evaluate the impact of a local modifications in the global system. However, these systems are comprised of many heterogeneous models, which cannot be integrated seamlessly in a single monolithic simulation. As such, Boeing regards co-simulation as one of the key technologies to enable the Digital Twin vision.

2.2 Boeing's Case Study

The case study presented was developed by Boeing. It constitutes a representative generic Flight Controls System, in the form of a co-simulation scenario. The FMUs are black boxes, having only the description of the input/output variables, and parameters. Moreover, no source code was made available, thereby protecting Boeing's Intellectual Property (IP), and the parameters given do not represent accurate values.

The IP-protected case study was shared with the University of Antwerp for co-simulation optimization, and the circumstances represent a faithful reproduction of Supplier–OEM relationships, where IP management tends to be an issue.

Case Study. Consider a control system, represented as a co-simulation scenario in Figure 1. The `Controller` FMU represents a software controller, the `Plant` and `Load` FMUs represent the physical subsystems. The `Environment` FMU produces a constant signal for `psu`, and a step signal for `ref`.

We do not have access to the correct behavior of the co-simulation scenario described in Figure 1. However, the `Load` FMU has been designed and tested against abstractions of the `Plant` and `Load` FMUs, hence we can assume that its behavior should not be fundamentally different in the co-simulation. Moreover, the `Plant`

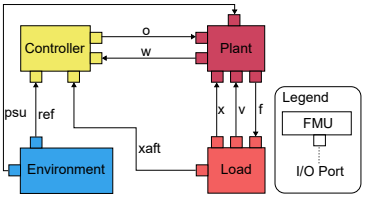

Figure 1: Case study co-simulation scenario.

and `Load` FMUs are produced by specialized teams, which know how the behavior of the corresponding subsystems should look like. Therefore, we make the following assumptions.

Assumption 1. *The `Controller` FMU is a software controller designed for a sample rate of* 1×10^6 Hz. *The `Plant` and `Load` FMUs model physical subsystems connected by a power connection, where `v` represents the effort, and `f` the flow.*

Assumption 1 represents the domain specific knowledge that users of co-simulation use to judge the correctness of the results. For example, the movement of the `Plant` and `Load` subsystems should be smooth.

Assumption 2. *The FMUs do not support rollback, never reject a step size, and do not have I/O feedthrough information.*

Assumption 2 reflects the fact that the FMU providers have implemented only the mandatory part of the FMI Standard.

Assumption 3. *The `Controller`, `Plant`, and `Load` FMUs are correct in the sense that, if they are provided with valid inputs, they will produce valid outputs with respect to their intended function. That is, the FMUs are correctly built.*

Assumption 3 means that, if the co-simulation results are not accurate, it is only because the co-simulation is not correctly configured.

The precise definition of co-simulation configuration is found in Section 3. In brief, the configuration includes the order in which the outputs are propagated to the inputs, the order of execution of each FMU, and the size of communication step. Even for small systems, the number of possible configurations can be infinite.

2.3 Analysis

The paragraphs that follow discuss why most co-simulation algorithms will fail to accurately reproduce the behavior of the co-simulation scenario in Figure 1. This is illustrated by showing two representative co-simulation algorithms that produce incorrect results with respect to assumption 1, which details information about FMUs and signals.

Experiment 1. Taking the hint that the `Controller` needs to sample the system every 1×10^{-6} s, the first co-simulation algorithm we apply is the fixed-step-Jacobi. This algorithm keeps the FMUs in sync by propagating outputs to inputs before asking each FMU to compute the next interval (Bastian et al., 2011).

Figure 2 shows the output computed by this algorithm for the `Load` FMU, which is examined because it is the most sensitive to the master algorithm. As clearly evident in the figure, the trace produced is not smooth.

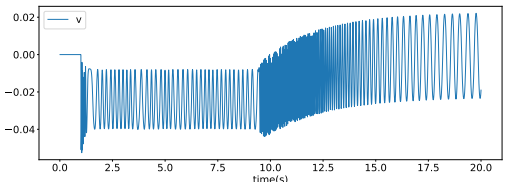

Figure 2: Output of `Load` FMU in experiment 1. Step size is 1×10^{-6} s.

Experiment 2. A user's intuition may be that the `Load` and `Plant` FMUs need to communicate at a higher rate than the sample of the `Controller` FMU due to assumption 1. Hence, we apply a multi-rate co-simulation algorithm such as the one described in (Van Acker et al., 2015), to produce the results shown in Figure 3. In this trial, we chose the communication rate of the `Load` and `Plant` FMUs to be ten times higher. However, the result is still not smooth.

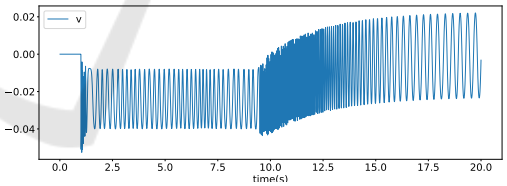

Figure 3: Output of `Load` FMU in experiment 2. Communication rate between `Plant` and `Sensor` is 1×10^7 Hz.

The results of experiments 1 and 2 suggest that an adaptive step size co-simulation algorithm (e.g., (Busch and Schweizer, 2012, Blockwitz et al., 2012, Arnold et al., 2014, Sadjina et al., 2017)) will fail: the step size of 1×10^{-7} s is already the minimum that can be used before the run-time execution time becomes intolerable by our industrial partners. For reference, the result in Figure 3 takes on average 32 minutes to compute on a Core i7 3.5GHz laptop.

To address the run-time execution issue, we turn to corrective co-simulation approaches: either a global error correction technique is used, or the input approximation of each FMU can be improved, so that less error is introduced. However, the global error

correction technique cannot be applied, because the co-simulation scenario includes FMUs whose output is discontinuous (`Scenario`, and `Controller`). This thus violates the continuity assumptions that both these techniques make.

Experiment 3. To improve the input approximations on each FMI, a Gauss-Seidel co-simulation algorithm (Bastian et al., 2011) can be employed to determine whether interpolations can be used instead of extrapolations on some of the FMUs. This algorithm executes each FMU in order, using the most recently computed outputs to feed the FMUs that still need to be executed.

For example, at the beginning of each co-simulation step, the `Load` FMU is given the output produced by the `Plant` and asked to compute the next interval. Then, the output `xaft` is propagated to the `Controller`, which is then asked to computed the next interval, and so on.

Figure 4 shows the results of this experiment. As can be seen, the trajectory is preferable to the other experiments, but is still not smooth enough for the system to be considered properly configured.

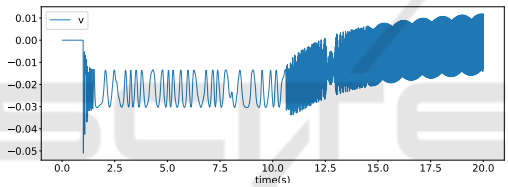

Figure 4: Output of `Load` FMU in experiment 3. Step size is 1×10^{-6} s, the signals `x` and `v` are extrapolated, and `f` is interpolated.

The Gauss-Seidel algorithm is more difficult to configure because it assumes that some FMUs can be executed with inputs "from a future simulated time". This is not the case for the `Controller`, because a software controller cannot predict the state of the `Load` or `Plant` FMU 1×10^{-6} s ahead. Similarly, the `Plant` can only react to a change in the output of the software controller after this change has occurred (this is formalized in Section 3). For the remaining FMUs used in the case study, causality is not an issue, because we are anyhow relaxing the higher frequencies at which they affect each other in reality (Krus, 1999). Moreover, higher-order input approximation techniques can only be applied to signals that are continuous (i.e., signals `x`,`v`, and `f`).

These restrictions imply that most state-of-the-art algorithms relying on correction of signals (e.g., (Ben Khaled-El Feki et al., 2017, Benedikt et al., 2013,González et al., 2019)) cannot be applied without some form of configuration.

3 PROBLEM FORMULATION

In this section, we detail our research problem. In brief, we search through the configuration parameters of the co-simulation master algorithm in order to generate good co-simulation candidates. Preferable candidates more closely match the user's intuition of the properties of the system, as expressed through the hints.

3.1 Co-simulation Formalization

In this section, we define some core concepts that allows us to formalize the notion of model behavior approximations, when there is no reference solution.

A *dynamical model* has the purpose of approximating the relevant behavior of the original system with respect to some properties of interest, denoted by P. We assume every dynamical model has a *behavior trace*, which is the set of trajectories followed by the state (and outputs) of a dynamical model. We refer to the time variable $t \in \mathbb{R}$ as *simulated time*—or simply *time*, when no ambiguity exists.

When the behavior trace of dynamical model M satisfies property $p \in P$, we write $M \vDash p$.

Definition 1. A dynamic model is *valid* when its behavior satisfies the same properties as the original system.

Remark 1. The satisfaction relation $\vDash$ used in Definition 1 provides a binary result. In practice, a model M can partially satisfy a property p. For simplicity, we assume that the set P is such that the partial satisfaction can be encoded with the $\vDash$ relation.

A *simulator* (or solver) $\mathcal{A}$ is an algorithm that computes the behavior trace of a dynamical model.

For a given model M and simulator $\mathcal{A}$, we denote the induced model by $[\![M]\!]_{\mathcal{A}}$. With this notation, the behavior trace computed is exact iff $[\![M]\!]_{\mathcal{A}} = M$, and approximate otherwise.

Because we do not always have access to the behavior trace of the model, it is more realistic to redefine the notion of accuracy in terms of properties.

Definition 2. Given a set of properties P, a simulator $\mathcal{A}$ is accurate when it satisfies the same subset of properties as the model:

$$\forall p \in P, M \vDash p \Leftrightarrow [\![M]\!]_{\mathcal{A}} \vDash p \qquad (1)$$

The measure of accuracy can then be the number of properties that satisfy Equation (1).

Remark 2. Definition 1 excludes performance related properties (e.g., execution time of the simulation algorithm). We argue that these are secondary properties whose satisfaction only makes sense when the primary

properties P are satisfied (which means the simulation results can be trusted).

We use the term FMU to denote an executable artifact that produces a behavior trace, when inputs are provided. The FMU combines a simulation $\mathcal{A}$ with a model M, and produces the behavior trace of $[\![M]\!]_{\mathcal{A}}$.

A *simulation* is the behavior trace obtained with an FMU. The correctness of the simulation depends on the accuracy of the simulator (Definition 2) and the validity of the dynamical model (Definition 1).

A coupled model is a dynamical model that is comprised of sub-models. When the sub-models are represented by different FMUs, we need co-simulation to approximate the behavior trace of the coupled model.

A co-simulation is the behavior trace of a coupled model approximated by a master algorithm applied to a co-simulation scenario. A co-simulation scenario is a set of FMUs and their I/O mappings (e.g., see Figure 1). A master algorithm represents the approach to compute the co-simulation. It typically determines the communication rate, and which data is exchanged between FMUs. When an FMU represents a continuous sub-model, its inputs need to be approximated. As such, we consider the input approximation schemes as being part of the master algorithm.

In the following, we formalize the concepts of FMU, co-simulation scenario, and master algorithm, with the intent of exposing the nuances in configuring a co-simulation.

We adapt the notations introduced in (Broman et al., 2013). To simplify and follow assumption 2, we leave out the notation for the initialization and feed-through. However, our implementation accounts for these omissions.

Definition 3. An FMU with identifier c is a structure $\langle S_c, U_c, Y_c, R_c, \texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c \rangle$, where: • S_c represents the state space; • U_c and Y_c the set of input and output variables, respectively; • $R_c : U_c \to \{true, false\}$ the reactivity of each input (see Definition 5); • $\texttt{set}_c : S_c \times U_c \times \mathcal{V} \to S_c$ 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 duration.

The following definition reflects the fact that the FMI Standard leaves implicit the current time of each FMU. However, this information is crucial to correctly configure the co-simulation.

Definition 4 (State timestamp)**.** Given a communication step size $H \in \mathbb{R}_{\geq 0}$ and $H > 0$, we say that the state $s_c \in S_c$ of an FMU c has timestamp t, denoted as $\varphi(s_c) = t$ when $\texttt{doStep}_c$ has been called $\frac{t}{H}$ times with H as parameter.

According to Definition 4, if an FMU is in state s_c at time t, $\texttt{doStep}_c(s_c, H)$ approximates the state of the corresponding model at time $t + H$. If this model is a continuous one, the FMU will approximate the evolution of the state in the interval $[t, t+H]$, using an approximation function to estimate the values of the inputs in that interval. In our notation, we choose to leave this function implicit in the $\texttt{doStep}_c$, as reflected in the current version of the FMI Standard. However, we make explicit the requirements of each kind of input approximation in the form of the reactivity R_c.

Intuitively, an FMU with a reactive input must wait until the FMU that feeds that input executes a step before getting that input value. The reactivity therefore imposes an order in the execution of the FMUs. This concept was first introduced in (Gomes et al., 2018b).

Definition 5 (Reactivity)**.** For a given FMU c with input $u \in U_c$, $R_c(u) = true$ if the function $\texttt{doStep}_c$ makes use of an interpolation of input u. Formally, let t be the timestamp of the state s_c prior to a call to $\texttt{doStep}_c(s_c, H)$, and let d denote the FMU whose output $y \in Y_d$ is connected to u. Then, $R_c(u) = true$ means that s_c must have been produced from a call to $\texttt{set}_c\big(\ldots, u, \texttt{get}_d(s_d, y)\big)$ where the state s_d of FMU d satisfies $\varphi(s_d) = t + H$. Conversely, $R_c(u) = false$ means that s_c must have been produced from a call to $\texttt{set}_c\big(\ldots, u, \texttt{get}_d(s_d, y)\big)$ where $\varphi(s_d) = t$.

Since knowing the reactivity of each FMU is related to having access to the input approximation implementation, and since the FMI Standard version 2.0 does not include information about reactivity, we make the following assumption.

Assumption 4. *If an FMU c does not disclose its input approximation scheme for an input u, then we assume that u is approximated with a constant extrapolation. Therefore,* $R_c(u) = false$.

Fortunately, the input approximation scheme of an FMU input, and therefore its reactivity, can be controlled by semantic adaptation.

Definition 6. *Semantic adaptation* is a technique that allows a new FMU c to be constructed from an old set of FMUs, using a custom implementation of the $\texttt{set}_c$,$\texttt{get}_c$, and $\texttt{get}_c$ functions (Gomes et al., 2018a).

Definition 7. A co-simulation scenario is a structure $\langle C, L \rangle$ where each FMU identifier $c \in C$ is associated with an FMU, as defined in Definition 3, 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 $L : U \to Y$.

A master algorithm is considered here as everything that influences the co-simulation result. The

following concepts are a way to isolate and formalize these different components.

Definition 8 (Co-simulation Step)**.** Given a co-simulation scenario $\langle C, L\rangle$, a co-simulation step is an ordered sequence of FMU function calls $(f)_{i\in\mathbb{N}}$ with

$$f \in F = \bigcup_{c\in C} \{\texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c\},$$

and i denoting the order of the function call. A function call f_i comes before a function call f_j, written as $f_i \twoheadrightarrow f_j$, if $i < j$, and comes immediately before, written as $f_i \rightarrow f_j$, if $i = j-1$.

It is important that the co-simulation step respects the reactivity of each FMU (recall Definition 5), and the couplings of the FMUs.

Definition 9 (Valid Co-simulation Step)**.** Given a co-simulation step size $H > 0$, a co-simulation is valid with respect to reactivity and couplings if it satisfies the following conditions:

1. Each function call uses the most recent FMU State as parameter. For example, if $f_j = \texttt{get}_c(s_c, y)$ then s_c must be the result of the most recent call to $\texttt{set}_c$ or $\texttt{doStep}_c$, that is, the maximal i such that $i < j$, and $f_i = \texttt{set}_c(\ldots)$ or $f_i = \texttt{doStep}_c(\ldots)$.
2. For every $c \in C$, there exists one, and only one, call to $\texttt{doStep}_c$, and it is done with argument H.
3. Each call to $\texttt{doStep}_c$ for $c \in C$ must come after every call to $\texttt{set}_c$ on the input variables of c.
4. Each call to `get` is immediately followed by a sequence of calls to `set` to set the affected input variables.
5. For each $c \in C$ and $u \in U_c$ satisfying $R_c(u) = \mathit{true}$, $\texttt{doStep}_d \twoheadrightarrow \texttt{get}_d(L(u), \ldots)$, where $L(u) \in Y_d$ and $d \in C$.
6. For each $c \in C$ and $u \in U_c$ satisfying $R_c(u) = \mathit{false}$, the call to $\texttt{set}_c(\ldots, u)$ $\texttt{set}_c(\ldots, u) \twoheadrightarrow \texttt{doStep}_d$, where $L(u) \in Y_d$ and $d \in C$.

Remark 3. Regarding Definition 9:

- The most common master algorithms will satisfy conditions 1–3;
- Condition 4 is not mandatory but it facilitates the description of Conditions 5 and 6. Furthermore, it makes the implementation simpler.
- Conditions 5 and 6 ensure that the reactivity of each input is respected, according to Definition 5.
- This definition is consistent with assumption 2. Relaxing this assumption requires modifications that are outside the scope of this work.

In addition to Definition 9, we make the following assumption, which is not strictly required to ensure a valid co-simulation step, but makes the description of the techniques employed in later sections simpler.

Assumption 5. *In a valid co-simulation step there is only one call to* $\texttt{get}_d(y, \ldots)$

Assumption 5 restricts the reactivity of every two input variables u, v that are fed by the same output variable, that is $P(u) = P(v)$, to be the same. We do not lose generality by making this assumption since one can perform multiple calls to $\texttt{get}_d(y, \ldots)$, before and after $\texttt{doStep}_d$ to get the right values.

Executing a co-simulation step in a co-simulation scenario $\langle C, L\rangle$ where all FMUs $c \in C$ have a state s_c satisfying $\varphi(s_c) = t$, will update each FMU state s_c to satisfy $\varphi(s_c) = t + H$, where H is the argument of every call to `doStep`.

Definitions 8 and 9 and assumption 5 purposefully exclude the case where a group of FMUs needs to communicate more frequently per co-simulation step, as in experiment 2. This is because this group can be transformed to a single FMU using semantic adaptation (Definition 6). Therefore, we do not lose generality. We revisit this in Section 5.4.

Definition 10. Given a co-simulation scenario $\langle C, L\rangle$, a co-simulation step size H, and a co-simulation step $(f)_{i\in\mathbb{N}}$, a master algorithm is a structure defined as $\mathcal{A} = \langle C, L, H, (f)_{i\in\mathbb{N}}\rangle$.

With this formalization, we can summarize the configuration parameters of a master algorithm:

- co-simulation step size H;
- FMUs and their semantic adaptations C, L; and
- co-simulation step $(f)_{i\in\mathbb{N}}$.

Each different parameter induces a model that is likely to be different that the original coupled model.

Definition 11. The induced coupled model, denoted by $[\![M]\!]_{\mathcal{A}}$, is the model whose behavior trace is computed by a given master algorithm $\mathcal{A}$, with the intent of approximating the behavior trace of a coupled model M.

3.2 Research Problem

With these assumptions and definitions, we can formalize our main goal.

Problem 1. *For a given set of properties P, a coupled model M, find a master algorithm*

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

that maximizes the size of the set

$$\{p : p \in P, M \vDash p \Leftrightarrow [\![M]\!]_{\mathcal{A}} \vDash p\},$$

such that $(f)_{i\in\mathbb{N}}$ *satisfies Definition 9 and assumption 5.*

Multiple solutions to Problem 1 are possible. In this work, we provide a way to generate multiple potential solutions for the user to evaluate.

Note that if the coupled model M is invalid (recall Definition 1), the optimal solution $\mathcal{A}$ would have to

```
Hint ExecRate{
    description "Controller FMU is software."
    statements {
        Property ExecRate :=
            FMUProperty FMU1.exec_rate == val 1.0e+6 hz
    }
    scope Globally
    pattern Universality:always-the-case-that ExecRate holds
}
Hint PowerBond{
    description "Plant/Load FMUs share a power connection."
    statements {
        Property PowerBond :=
            Plant.f == PowerBondSuggestions with Load.v
    }
    scope Globally
    pattern Universality:always-the-case-that PowerBond holds
}
```

Figure 5: The *ExecRate* and *PowerBond* hints.

not satisfy the same properties that the model does not satisfy. Therefore, we make the following assumption:

Assumption 6. *When solving Problem 1, we assume that M is valid, according to Definition 1.*

As well, in practice the set of properties P is not completely specified. This motivates our proposal of using *hints* as an approximation of P, to be derived from requirements, or declared by engineers. Assumption 7 reflects that we must rely on these hints to obtain information about M.

Assumption 7. *We assume that M's behavior satisfies the hints provided.*

In the next section, we describe how these hints are represented such that the user does not need to understand the co-simulation domain. Then, in Section 5 we discuss our approach to solve Problem 1.

4 HINT LANGUAGE

In this section, we describe how to represent the hints used to configure co-simulation as defined in Problem 1. This is done through the creation of a small *domain-specific language* (DSL). DSLs allow experts in the problem space (the system engineers) to describe hints, without having to become experts in the solution space (the co-simulation domain) (Vangheluwe et al., 2002).

As an example, Figure 5 show the hints described in assumption 1. Each hint has a number of fields. The *description* field is an unstructured text, as commonly seen in industrial requirements. Following this are *statements*, which can be *events* or *properties*. Finally, the *scope* and the *pattern* specify when the hint is valid.

Statements. As seen in Figure 5, *Statements* define the *Events* and *Properties* which refer to FMUs and their signals in the system.

For brevity, we omit the description of several other operators that can be used as statements. For example, hints can be specified over the average or derivative of a signal. The language is defined to be easily extensible, and we are collaborating with our industrial partners to define further useful operators.

Scopes and Patterns. The example hints in Figure 5 are applicable throughout the entire simulated time. This is denoted by the *Globally scope* of the hint, and the *Universality pattern*. These scopes and patterns are sourced from (Autili et al., 2015).

Scopes define when the hint is valid: *Globally*, *Before an event*, *After an event*, *Between two events*, and *After an event until another event*.

Patterns define the precise manner in which a statement holds. We refer the reader to (Autili et al., 2015) for a full description and syntax of the patterns. As an example, the *MinDuration* pattern means *once [an event] holds, it will hold for [an amount of time]*. This allows the engineer to precisely define how the statements defined in the hint should hold.

Implementation. The hint language was created with the XText DSL framework[2], which produces a XML Metadata Interchange (XMI) file representing each hint. These files are then used for the generation of the master algorithms that are candidate solutions to Problem 1.

5 MASTER GENERATION

This section describes the search algorithm that enumerates potential solutions to Problem 1 using a given set of hints.

5.1 Search Space Representation

Given a co-simulation scenario, the search space is the set of all master algorithms that can be used to compute the co-simulation. According to Definition 10, we identify the following dimensions of the search space:

- the set of all communication step sizes;
- the set of all co-simulation steps;
- the set of all semantic adaptations applied to the FMUs.

The search space is therefore infinite, though as shown below we consider only a finite subset of this space. The problem of representing a search space is not new, and there is a rich literature in design space exploration from where we draw inspiration (Kang et al., 1990,

[2]https://www.eclipse.org/Xtext/

Van Tendeloo and Vangheluwe, 2017, Vanherpen et al., 2014).

Figure 6 shows an excerpt metamodel of the search space representation. The search space, represented by class *Candidates*, comprises multiple *RootCandidateScenarios*. The latter represent alternative master algorithms. The co-simulation step operations are left implicit, restricted by the semantic adaptations used (see Section 5.3).

Semantic adaptations can be applied to FMUs and input ports, and since FMUs can be hierarchical, this enables the representation of multi-rate master algorithms. Of the currently supported semantic adaptations, we highlight:

Extrapolation/Interpolation Adaptation. Applies the approximation to the affected FMU input port.

MultiRateAdaptation. Makes an FMU perform multiple steps per co-simulation step. Can be combined with *Extrapolation/Interpolation* adaptations, and can also be applied to hierarchical FMUs.

PowerBond Adaptation. Whenever two FMUs share a power connection, the *PowerBond* adaptation changes one of the FMU's input ports to correct for the energy dissipated, using the technique introduced in (Benedikt et al., 2013).

XOR Adaptation. Can be combined with other adaptations to represent alternatives.

The variability of the co-simulation candidates is therefore encoded in alternative root candidate scenarios and *XorAdaptations*.

Example 1. Figure 7 shows an example search space. The `Load` and `Plant` FMUs have a *PowerBond* adaptation, and the `Environment` FMU has an *XorAdaptation* with two alternative multi-rate adaptations. This search space represents four alternative master algorithms, because of the two `Environment` FMU rates, represented as $R = \{100, 10\}$ in the figure, and two possible communication step sizes, represented as $H = \left\{1 \times 10^{-7}, 1 \times 10^{-6}\right\}$.

Given a set of hints, our tool generates a search space representation, according to the algorithm described in Section 5.5. In order to explain the algorithm, we first have to describe how different *master algorithm variants* are generated and executed.

5.2 Variant Generation

Definition 12. Given a search space representation, a *master algorithm variant* represents a set of decisions made about each variation point represented in the search space.

Definition 13. A *variant diagram* represents all possible variants in the search space. It is a rooted, connected, directed, and acyclic graph, where each node has the same children as every other sibling node. Each node is associated with a weight and represents a decision relevant to the master algorithm. The children of the same node represent a set of mutually exclusive choices, where the weight of each child represents the preference for each choice. The higher the weight of a node, the closer to the root that node and its siblings will be.

Example 2. Figure 8 shows a variant diagram representing the variants encoded in Example 1. The co-simulation step nodes are just below the root because the co-simulation step 1×10^{-7} s has the maximal weight of the whole tree.

For now, we assume for brevity that these co-simulation scenarios do not have hierarchical FMUs. This assumption is relaxed in Section 5.4.

Definition 14. Given a variant diagram, a master algorithm variant is defined as being a path from the root node to a leaf node.

The number of possible variants in a diagram is the number of different paths from the root to a leaf node. Since it is not feasible to try all variants for a large co-simulation scenario, we rank them.

The variants are ranked according to the weight on each node, using the search procedure defined below. These weights are set by the procedure that generates the search space from the hints (Section 5.5).

Procedure 1. *Starting at the root, each child node with the highest weight is visited first. A variant is created when the search reaches a leaf of the tree. The search then continues in the tree using backtracking, heading up to the closest sibling node with the highest weight (among siblings) not yet chosen.*

Example 3. Procedure 1 generates the following variants, in order: $\langle H = 1 \times 10^{-7}, R = 100 \rangle$; $\langle H = 1 \times 10^{-7}, R = 10 \rangle$; $\langle H = 1 \times 10^{-6}, R = 100 \rangle$; and $\langle H = 1 \times 10^{-6}, R = 10 \rangle$.

5.3 Variant Execution

A variant encodes the co-simulation scenario, semantic adaptations to be applied, and parameter values. A master algorithm comprises the co-simulation step, as defined in Definitions 8 and 10, which needs to satisfy the conditions in Definition 9 and assumption 5, which in turn depend on the semantic adaptations selected for the variant. In order to represent all the constraints in the execution order of operations, we introduce the following structure.

Definition 15. Given a variant, we define the corresponding operation schedule as a directed graph where

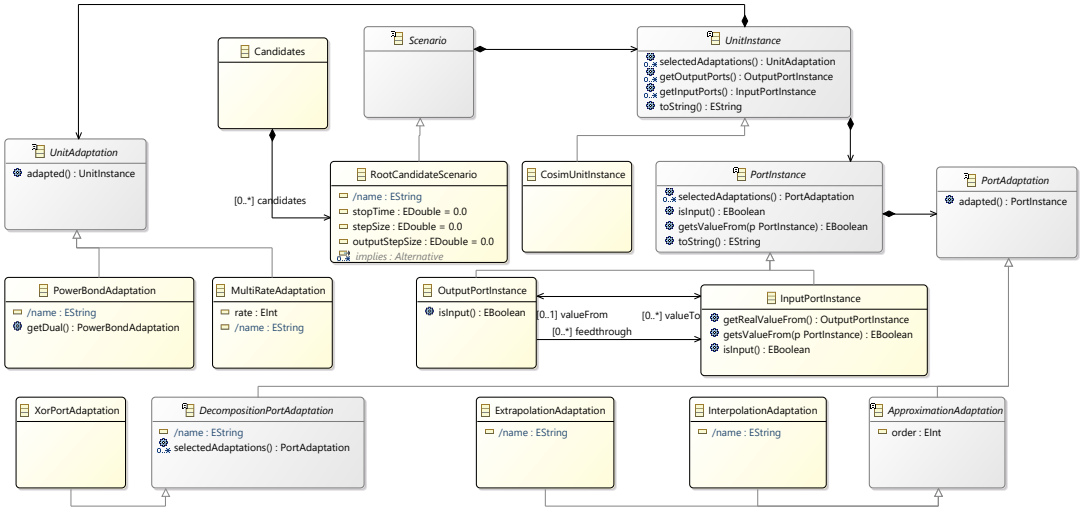

Figure 6: Excerpt of search space representation metamodel.

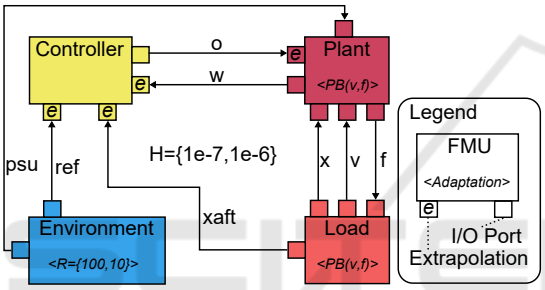

Figure 7: Example search space.

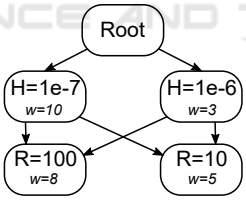

Figure 8: An example variant diagram.

each node represents an operation in F (Definition 8), and each edge between nodes i and j means that the operation represented by i must be executed before the operation represented by node j. The edges are created according to the semantic adaptations selected for each unit and port, as described in Definition 9.

Example 4. Figure 9 shows the operation schedule of all variants described in Example 2.

Definition 16. A variant is executable if the corresponding operation schedule has a topological sort.

Furthermore, according to the assumptions we have made, if there are multiple topological orderings, they are all behaviorally equivalent.

5.4 Hierarchical FMUs

To keep the explanation simple, we omitted how hierarchical FMUs are handled. Since a Hierarchical FMU represents essentially a co-simulation scenario, they are treated the same way when it comes to variant representation and generation. The main difference lies in the creation of the operation schedule.

Procedure 2. *Given a variant that contains hierarchical FMUs, we build the operation schedule according to Definition 15, but excluding the operations that correspond to child FMUs of the hierarchical FMUs. Then, we recursively create the operation schedule of the FMUs inside each hierarchical FMU. The operation schedule of each Hierarchical FMU c is run whenever the* `doStep`$_c$ *operation is invoked.*

5.5 Search Space Generation

In this section, we focus on how the search space is created. To be succinct, we will only focus on the hints that were applied to the case study.

Procedure 3. *Given a set of hints and a co-simulation scenario, the search space is created as follows:*

1. *For each FMU with a software controller hint, add an extrapolation adaptation to each of its input ports and to each of the input ports that are connected to its outputs.*
2. *If there are multiple software hints with different configured frequency rates, define the scenario step size to be the inverse of the minimum of the frequency rates, and define the appropriate multi-rate adaptations on the software FMUs.*
3. *For each PowerBond hint, add a power bond adaptation to each of the FMUs sharing the bond.*

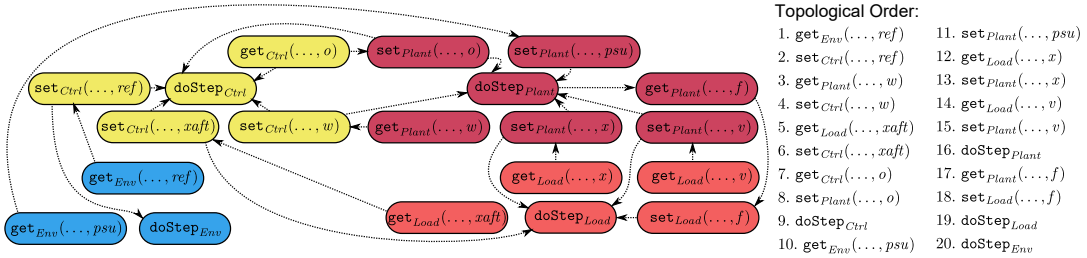

Figure 9: Example operation schedule for variants in Figure 8. The edges represent ordering constraints, as in Definition 15. A possible topological order is displayed on the left.

4. *Select the FMUs that are not affected by any hint, and add a multi-rate adaptation (if not already defined) with alternative step sizes at different orders of magnitude, and two alternative first order input approximations (interpolation and extrapolation). Higher weights are given to smaller step sizes and interpolations.*
5. *If the co-simulation step has not been defined yet, define multiple alternative co-simulation steps with different orders of magnitude. Higher weights are given to smaller step sizes.*

5.6 Results

We applied the above procedure to the co-simulation scenario introduced in our motivating example in Section 2, with the *ExecRate* and *PowerBond* hints described in Section 4. This resulted in the search space in Figure 7, and the four possible master algorithms, described in Example 3.

All variants produced a smooth signal, shown in Figure 10. The fastest variant took 6 minutes, on average, to produce the result. The slowest variant took 38 minutes.

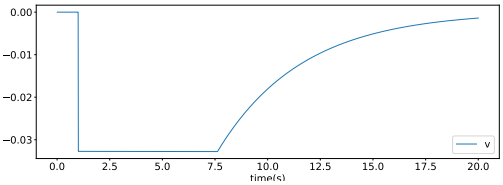

Figure 10: Co-simulation computed from the hints provided in Figure 5, with the variant $\langle H = 1 \times 10^{-6}\,\mathrm{s}, R = 10 \rangle$.

6 RELATED WORK

The problem of adequately configuring a co-simulation is not new. We can classify the approaches in two categories: adaptive and static configuration.

In the adaptive category, we highlight the adaptive co-simulation schemes: Input Approximation/Correction (Ben Khaled-El Feki et al., 2017, Benedikt et al., 2013, González et al., 2019); and Step-size Correction (Busch and Schweizer, 2012, Blockwitz et al., 2012, Arnold et al., 2014, Sadjina et al., 2017). Our approach complements the state of the art in providing a way to select which methods are best applicable. More importantly, our approach acknowledges that the same technique cannot be applied to every FMU in the co-simulation scenario.

In the static configuration category, the following works have the same goal as this paper.

The authors in (Krammer et al., 2015) propose to use system models to configure the co-simulation. However, their approach to configure the co-simulation differs from ours by not attempting to generate multiple candidate master algorithms. They also focus only on ensuring the configuration is syntactically correct (e.g., the FMU connections are not consistent with the model). In principle, it is possible to adapt their proposed language to include hints and explore multiple good master algorithms.

The work in (Benedikt and Holzinger, 2016) recognizes the need to take into account the input/output feed-through, and the kind of model underlying the FMU, in order to configure the co-simulation. It shows that the input approximation techniques and the step size adaptation can interfere with each other. We complement this work by showing that there is more information that can be used to configure the co-simulation. Additionally, we acknowledge that multiple master algorithms can perform well, instead of focusing on generating just one.

The approach in (Holzinger and Benedikt, 2019) is similar to our own, as it formulates an optimization problem to find the best co-simulation step (as defined in Definition 8). However, in contrast with our work, they make the assumption that FMUs can perform interpolation or extrapolations equally well. As well, we formulate an optimization problem whose solution attempts to satisfy the hints provided by the user.

7 CONCLUSION

Due to the circumstances in which co-simulation is applied, there are often no analytical results to guide the search for a correct configuration of the co-simulation.

This paper has presented a framework, available online[3], that allows users to explore various options for configuring their co-simulation scenarios, by describing hints. These hints are used to produce a search plan for the different co-simulation variants, taking into account the best practices in the state of the art. Each variant is then evaluated and the results are presented to the user.

A major assumption that we have made is that the coupled model being co-simulated satisfies the hints described by the engineers (assumptions 6 and 7). However, it is often the case that the users of co-simulation are trying to understand the model and therefore may not describe correct hints. Our contribution can help by showing whether there is agreement between master algorithms on the satisfiability of a particular set of hints. A hint that is not satisfied by any of the co-simulation algorithms generated suggests that perhaps the problem is in the model and not in the co-simulation.

Ongoing work is focused on supporting more hints, including proprietary hints. In the future, we intent to formalize some of the hints in Signal Temporal Logic, and compute a degree of satisfaction per hint (Remark 1 on page 4). This will allow us to use global optimization techniques to improve the recommended master algorithms. Another direction is to derive the hints from prior experience with the components. For example, descriptions of previous physical experiments, in the form of experimental frames (Denil et al., 2017), could be used.

For Boeing, the candidate master algorithms suggested by the HintCO tool represent a first step towards enabling seamless integrated large scale simulation of heterogeneous models. However, there are still challenges that need to be overcome. For example, tool vendors need to improve the maturity level of FMU support, to focus on enhanced cross compatibility and integration checks (recall assumption 2). Furthermore, there is a need to develop a common methodology and modeling guidelines for industrial applications.

ACKNOWLEDGEMENTS

This work is partly supported by INES (Innovation in the Development of Electrical Systems For Aeronautics) under project no. 11172. We thank our colleagues at Siemens Industry Software in Leuven, Belgium for supporting us (Jef Stegen, Jonathan Menu, and Satya Prakash Jha), and our colleagues at Boeing Research & Technology Europe, Madrid, Spain. This research was partially supported by a PhD fellowship grant from the Research Foundation - Flanders (File Number 1S06316N).

[3]https://msdl.uantwerpen.be/git/claudio/HintCO

REFERENCES

Arnold, M., Clauß, C., and Schierz, T. (2014). Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-Simulation v2.0. In *Progress in Differential-Algebraic Equations*, pages 107–125, Berlin, Heidelberg. Springer Berlin Heidelberg.

Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., and Tang, A. (2015). Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar. *IEEE Transactions on Software Engineering*, 41(7):620–638.

Bastian, J., Clauß, C., Wolf, S., and Schneider, P. (2011). Master for Co-Simulation Using FMI. In *8th International Modelica Conference*, pages 115–120, Dresden, Germany. Linköping University Electronic Press, Linköpings universitet.

Ben Khaled-El Feki, A., Duval, L., Faure, C., Simon, D., and Ben Gaid, M. (2017). CHOPtrey: Contextual online polynomial extrapolation for enhanced multi-core co-simulation of complex systems. *SIMULATION*, 93(3).

Benedikt, M. and Holzinger, F. R. (2016). Automated configuration for non-iterative co-simulation. In *17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, pages 1–7, Montpellier. IEEE.

Benedikt, M., Watzenig, D., Zehetner, J., and Hofer, A. (2013). NEPCE-A Nearly Energy Preserving Coupling Element for Weak-coupled Problems and Co-simulation. In *IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems*, pages 1–12, Ibiza, Spain.

Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. (2012). Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *9th International Modelica Conference*, pages 173–184, Munich, Germany. Linköping University Electronic Press.

Boeing (2017). Developing Airplane Systems Faster and with Higher Quality through Model-Based Engineering.

Broman, D., Brooks, C., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M. (2013). Determinate composition of FMUs for co-simulation. In *Eleventh ACM International Conference on Embedded Software*,

page Article No. 2, Montreal, Quebec, Canada. IEEE Press Piscataway, NJ, USA.

Busch, M. and Schweizer, B. (2012). Coupled simulation of multibody and finite element systems: An efficient and robust semi-implicit coupling approach. *Archive of Applied Mechanics*, 82(6):723–741.

Denil, J., Klikovits, S., Mosterman, P. J., Vallecillo, A., and Vangheluwe, H. (2017). The experiment model and validity frame in M&S. In *Proceedings of the Symposium on Theory of Modeling & Simulation*, volume 49, page Article No. 10, Virginia Beach, Virginia, USA. Society for Computer Simulation International.

FMI (2014). Functional Mock-up Interface for Model Exchange and Co-Simulation. Technical report, FMI development group.

Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., and De Meulenaere, P. (2018a). Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators. *SIMULATION*, 95(3):1–29.

Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2018b). Co-simulation: A Survey. *ACM Computing Surveys*, 51(3):Article 49.

Gomes, C., Thule, C., Larsen, P. G., Denil, J., and Vangheluwe, H. (2018c). Co-simulation of Continuous Systems: A Tutorial. Technical Report arXiv:1809.08463 [cs, math], University of Antwerp.

González, F., Arbatani, S., Mohtat, A., and Kövecses, J. (2019). Energy-leak monitoring and correction to enhance stability in the co-simulation of mechanical systems. *Mechanism and Machine Theory*, 131:172–188.

Holzinger, F. R. and Benedikt, M. (2019). Hierarchical Coupling Approach Utilizing Multi-Objective Optimization for Non-Iterative Co-Simulation. In *13th International Modelica Conference 2019*, page 6, Regensburg, Germany. Linköping University Electronic Press.

Kang, K. C., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-Oriented Domain Analysis. Feasibility study,. Technical report, Carnegie Mellon University.

Krammer, M., Fritz, J., and Karner, M. (2015). Model-Based Configuration of Automotive Co-Simulation Scenarios. In *48th Annual Simulation Symposium*, pages 155–162, Alexandria, Virginia. Society for Computer Simulation International San Diego, CA, USA.

Krus, P. (1999). Modeling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints. *Journal of Dynamic Systems, Measurement, and Control*, 121(4):606.

Kübler, R. and Schiehlen, W. (2000). Two Methods of Simulator Coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):93–113.

Lee, E. A. (2008). Cyber Physical Systems: Design Challenges. In *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369.

Sadjina, S., Kyllingstad, L. T., Skjong, S., and Pedersen, E. (2017). Energy conservation and power bonds in co-simulations: Non-iterative adaptive step size control and error estimation. *Engineering with Computers*, 33(3):607–620.

Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J., Posch, A., and Nouidui, T. (2018). Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers. In *The American Modelica Conference*, pages 138–146, Cambridge, MA, USA. Linköping University Electronic Press, Linköpings universitet.

Schweizer, B., Li, P., and Lu, D. (2015). Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches. *Journal of Computational and Nonlinear Dynamics*, 10(5):051007.

Van Acker, B., Denil, J., Meulenaere, P. D., and Vangheluwe, H. (2015). Generation of an Optimised Master Algorithm for FMI Co-simulation. In *Symposium on Theory of Modeling & Simulation-DEVS Integrative*, pages 946–953, Alexandria, Virginia, USA. Society for Computer Simulation International San Diego, CA, USA.

Van Tendeloo, Y. and Vangheluwe, H. (2017). Increasing the performance of a Discrete Event System Specification simulator by means of computational resource usage "activity" models. *SIMULATION*, 93(12):1045–1061.

Vangheluwe, H., de Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*, pages 9–20. SCS.

Vanherpen, K., Denil, J., De Meulenaere, P., and Vangheluwe, H. (2014). Design-Space Exploration in Model Driven Engineering: An Initial Pattern Catalogue. In *1st International Workshop on Combining Modelling with Search and Example-Based Approaches (CMSEBA)*, pages 42–51. CEUR Workshop Proceedings (Vol-1340).