Short Term Scientific Report COST IC1404

April 26, 2018

Title of the STSM: A Survey and Benchmark for Hybrid Co-simulation Working Group: WG2 (Techniques)
Beneficiary: Cláudio Gomes
Host: Aarhus University
Period: 18/03/2018 - 26/03/2018
Reference Code: 40960
1. Initial purpose of the visit We plan to survey hybrid co-simulation techniques, focusing on how time is handled, how events are detected, and the terminology used. To evaluate the multiple approaches, we intend to develop a benchmark case study, borrowing from the industrial experience of the partners of the INTO-CPS Project. Furthermore, we will use a taxonomy created in the context of a previous STSM to classify the new approaches.
2. Description of the work carried out during the STSM

We found an interesting benchmark, and applied model checking techniques to prove that one of the well known synchronization algorithms is wrong. Out of the 6 tasks identified in the planning phase, we realized 5. The benchmark will be made available online if the submitted workshop paper is accepted.
3. Description of the main results obtained

We submitted a workshop paper with the main results of this research, and we have other potential benchmarks to study.
4. Future collaboration with the host institution

The collaboration is ongoing, we one of the participants in the STSM (Casper Thule) is doing a PhD in this topic.
5. Foreseen publications/articles and other contributions

A workshop paper has been submitted (see the appendix). We intend to write a conference paper with an extension of this work.
6. Confirmation by the host institution

A Submitted Workshop Paper

Towards the Verification of Hybrid Co-simulation Algorithms*

Casper Thule[1](✉), Cláudio Gomes[26], Julien Deantoni[3], Peter Gorm Larsen[1], Jörg Brauer[4], and Hans Vangheluwe[256]

[1] Aarhus University, Denmark
{casper.thule,pgl}@eng.au.dk
[2] University of Antwerp, Belgium
{claudio.gomes,hans.vangheluwe}@uantwerp.be
[3] Polytech Nice Sophia, France
julien.deantoni@polytech.unice.fr
[4] Verified Systems International GmbH, Germany
brauer@verified.de
[5] McGill University, Canada
[6] Flanders Make, Belgium

Abstract. Engineering modern, hybrid systems is getting increasingly difficult due to the heterogeneity between different subsystems. Modeling and simulation techniques have traditionally been used to tackle complexity, but with increasing heterogeneity of the subsystems, it becomes impossible to find appropriate modeling languages and tools to specify and analyze the system as a whole. Co-simulation is a technique to combine multiple models and their simulators in order to analyze the behavior of the whole system over time. Past research, however, has shown that the naïve combination of simulators can easily lead to incorrect simulation results, especially when co-simulating hybrid systems. This paper shows (i) how co-simulation of a family of hybrid systems can fail to reproduce the events that should have occurred (event preservation); (ii) how to prove that a co-simulation algorithm is correct (w.r.t. event preservation), and if it is incorrect, how to obtain a counterexample showing how the co-simulation goes wrong; and (iii) how to correct an incorrect co-simulation algorithm. We apply the above method to two well known co-simulation algorithms, and we show that one of them is incorrect for the family of hybrid systems under study.

Keywords: hybrid co-simulation, hybrid systems, model checking

* This work was started in the CAMPaM 2017 Workshop, executed under the framework of the COST Action IC1404 – Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS), and partially supported by: Flanders Make vzw, the strategic research centre for the manufacturing industry; and PhD fellowship grants from the Agency for Innovation by Science and Technology in Flanders (IWT, dossier 151067).

1 Introduction

Engineered systems are becoming increasingly complex while market pressures shorten the available development time [24]. There are many causes for increase in complexity, but to some extent, it is caused by the number of interacting subsystems and differences between their domains [30], which is not sufficiently addressed by the established techniques. There is thus a need for an improved development cycle, with better tools, techniques, and methodologies [31]. While modeling and simulation has been successfully applied to reduce development costs, these fall short in fostering more integrated development processes [5].

A promising concept for the simulation of systems consisting of decoupled components is co-simulation (collaborative simulation) [23], which is based on the idea that the interacting subsystems of a coupled system are best modelled and simulated by dedicated tools and formalisms [32]. Each subsystem is then modelled by a specialized team using mature tools, tailored to the domain of the allocated subsystem. Further, each subsystem internally uses its own simulation engine, so that the most appropriate approximation techniques can be employed. The behavior of the coupled system is computed by having the simulation tools communicate with one another, exchanging their outputs over time.

Co-simulation foments a more integrated development process by allowing different teams to observe how their subsystem behaves when coupled to the rest of the system, while reusing the work made by the other teams. Furthermore, it improves the relationship between external suppliers and system integrators, where the system integrators can use virtual surrogates of the subsystems produced by the suppliers, to test their adequacy. With the appropriate Intellectual Property protection in place, these virtual surrogates can even be developed by the supplier, for increased validity.

In order to run a co-simulation, all that is required is that the participating simulation tools expose the inputs and expose the outputs, of the allocated subsystem, over time. A co-simulation engine then synchronizes the interface values of the different subsystems. This powerful approach eases the integration of subsystems simulated by different tools, also but also poses some difficulties. In particular, subsystems are modelled and treated as black boxes, and it is difficult in some cases to understand how the coordination of the subsystems—a functionality provided by the co-simulation engine—affects the behavior of the co-simulated system [19].

One might be tempted to expect that the behavior computed via co-simulation matches the behavior of the coupled system. In practice, however, this expectation turns out overly optimistic, and significant deviations may become visible, which could, for example, be caused by discretization or timings. This not only due to the inherent limitations of approximate simulations [10], but also due to the internals of the subsystem simulations. It is therefore important to study how a faulty co-simulation can be distinguished from a correct one. We approach this problem by defining a set of properties, which need to be satisfied by the co-simulated system if they are satisfied by the ideal system. If co-simulation

preserves these properties, we then say that the properties of the system are preserved under co-simulation.

One of the fundamental research topics in co-simulation is to decide whether a given property is preserved under co-simulation, which naturally leads to the connected problem: If co-simulation fails to preserve a property, how can the co-simulation be changed, so that the property is preserved? This paper contributes to this line of research as follows:

- We identify a novel property called *event preservation property*, which is often implicitly required to be preserved by co-simulations of systems that combine software with physical subsystems.
- We present a characterization of the event preservation property as a model checking problem so as to automatically decide whether a given co-simulation satisfies the event synchronic property for a restricted class of coupled systems.
- We show how, if the event preservation property is not preserved under co-simulation, the co-simulation coordination algorithm can be adapted in order to preserve event preservation.

One of the strengths of our approach is that, when a property is violated, our approach yields a counterexample that includes a *co-simulation scenario* and an *execution trace* of the co-simulation. It is difficult to overestimate the value of counterexamples produced by model checkers [11], and also in our work counterexamples provide valuable insight into how the co-simulation violates the event preservation property. The Maestro orchestration algorithm [29] serves as a case study for our approach.

The remainder of this paper is structured as follows. First, Section 2 discusses the related work, followed by a primer on co-simulation and co-simulation properties in Section 2.1. Afterwards, in Section 3, the event preservation property is demonstrated and described along with an encoding of the problem as a model checking instance. Finally, the paper presents a discussion and perspective on future work (Section 4) before concluding in **??**.

2 Property Preservation in Co-simulation

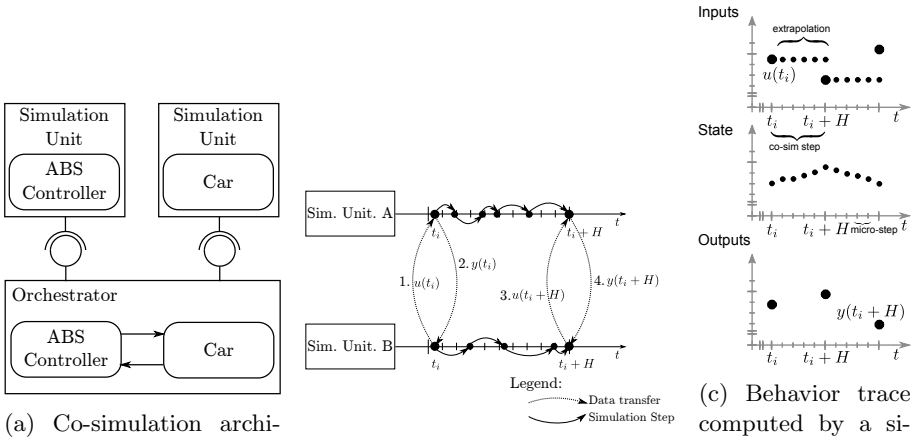
In this section, we present some background concepts, and review some of the works focusing on specific properties that co-simulation should preserve. We refer the reader to [18], and references thereof, for an introduction to co-simulation.

2.1 Background Concepts

A co-simulation is the behavior trace of a coupled system, produced by the coordination of simulation units. The behavior trace is a set of points over time. A simulation unit is an executable software entity responsible for simulating a part of the system. Furthermore, a simulation unit implements a predefined interface, allowing an orchestrator, described below, to communicate with it.

One such communication interface is the Functional Mockup Interface (FMI) standard [6]. A simulation unit implementing the FMI interface is called a Functional Mock-up Unit (FMU). The main functionality of an FMU concerns calculating outputs based on inputs and time. This is represented in FMI as three functions: a function to set inputs, a function to perform a step with a given step size, and a function to get outputs.

An orchestrator is a software component that sets/gets inputs/outputs of each simulation unit, and ask them to compute the behavior trace of its allocated subsystem over a requested time interval. From a co-simulation step to the next, the orchestrator follows the co-simulation scenario to know the order in which it asks each simulation unit to simulate and where to copy their outputs. A co-simulation scenario is a description of how the subsystems are interconnected. When asked to simulate for an interval of time, a simulation unit will typically perform multiple micro-steps, and employ an input approximation technique to compute the behavior trace of its subsystem. Figures 1a to 1c summarize these concepts.



(a) Co-simulation architecture.

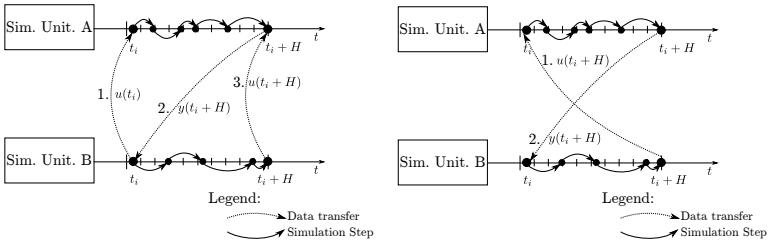
(b) Jacobi orchestration algorithm.

(c) Behavior trace computed by a simulation unit.

Fig. 1: Co-simulation main concepts.

There are three main orchestration algorithms: Jacobi, Gauss-Seidel, and Strong-coupling. The Jacobi co-simulation algorithm proceeds by asking all simulators to produce outputs, then it computes and sets the inputs that all simulators need. Afterwards, it asks all simulators to simulate their corresponding subsystem until the next communication time point, after which the process repeats. The Gauss-Seidel algorithm assigns an order to each simulator, and, in that order, computes the inputs of the simulator, then asks the same simulator to simulate to the next time point, obtains its output, and uses that output to compute the input to the next simulator. These steps are repeated until all simulators have simulated until the next time point, and then the process starts over

again. The strong coupling algorithm implements one of the above algorithms, except that it asks the simulators to rollback to a previous time point, and uses the most recently computed outputs as new inputs to these simulators, before asking them to simulate again. The essence of these algorithms, applied to two simulation units, is illustrated in Figures 1b and 2.



(a) Gauss-seidel. (b) Strong coupling.

Fig. 2: Coupling algorithms.

2.2 Stability

A coupled system is stable when it eventually comes to a rest. Since many systems are engineered to be stable [3], it is important that this property is preserved under co-simulation. The works in [9,27,22,1,16] study the conditions under which the stability property is conserved for selected physical coupled systems.

The main conclusions regarding stability preservation in co-simulation are: strong coupling algorithms are more likely to preserve the property; and sequential co-simulation algorithms should start the iteration with the simulation unit that have higher inertia [1].

2.3 Energy Conservation

Systems whose models account for the flow of energy follow the principle of conservation of energy. That is, no energy is lost when flowing between subsystems.

This property is not preserved in naive co-simulation algorithms because of the input approximations, and the non-negligible communication step size. The work in [4], extended in [26], demonstrates a co-simulation algorithm that monitors the power flow between simulators and employs a correction scheme to account for the artificial energy introduced by the co-simulation. The work in [25] complements the above work by showing how the energy residual can be used as an error indicator to control the communication step size.

2.4 Event Synchrony

A co-simulation preserves event synchrony when any event happening at a specific time in the original hybrid system is also reproduced by the co-simulation at the same time. A hybrid system is a system comprising software and physical subsystems. This is one of the properties studied in [14], in the context of co-simulations involving two simulation units: one responsible for the software subsystem, and the other for a continuous subsystem.

In order to enable an easier comparison of event timestamps, [12] proposes the use integers, instead of floating point numbers, to represent time. The same work proposes changes to the FMI Standard, so that orchestration algorithms that guarantee event synchrony can be built.

The correct handling of events is paramount to the preservation of the energy and stability properties in a co-simulation. As such, the work in [15] relates these by exploring 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 maximum event detection delay so that the stability is preserved in the co-simulation.

3 Verification of Master Algorithms

The previous section introduced multiple properties that should be preserved in a co-simulation. In particular, it introduced the event synchrony property. This property states that every event happening in a hybrid system, happens at the exact same time in the corresponding co-simulation.

In order to preserve this property, because the exact time of the event is often difficult to predict, the orchestration algorithm only detects the event after it occurs, and then restores the simulation to a prior state (where the event has not yet happened) and proceeds with more caution. This is repeated until the time of the event is known with sufficiently high accuracy [33].

In practice, due to the lack of rollback capabilities and/or performance constraints, it might be hard to guarantee that this property is preserved. Instead, it might be more useful to just require that the sequence of events is preserved, even if their timestamps do not coincide exactly. One can see the preservation of this property as the preservation of the untimed behavior of the software subsystem.

3.1 Hybrid System under Study

In this work, we focus on a restricted class of hybrid systems, in order to study one essential challenge encountered while attempting the preserve this property on event synchrony. The system under study is illustrated in Figure 3. It consists of a software part, and a physical part. The software part is represented as Statechart [20], and the physical part is represented by a differential equation.

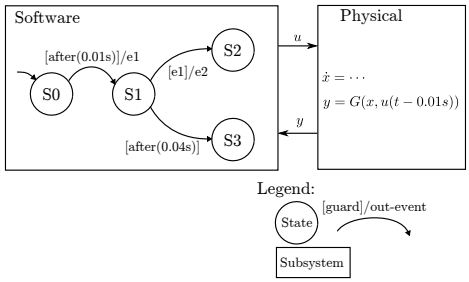


Fig. 3: Hybrid systems under study.

The software part is representative of a control systems that has a timeout mechanism, triggered whenever the continuous plant fails to react to some stimuli. The details of the dynamics of the physical subsystem are not important. What is important is that its output is a delayed function of the input, so that any change in the input is reflected on the output 0.01s later.

An execution of the software subsystem is plotted in Figure 4. At time 0.01s, the output event e1 is produced. This event affects the output of the physical system 0.01s later, which is picked up by the software unit, causing it to change to S2.

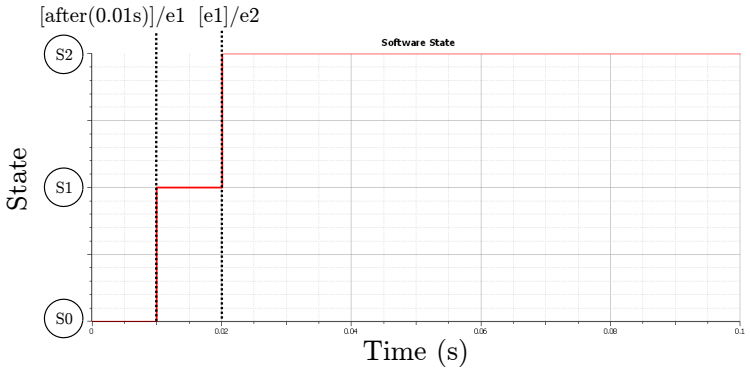


Fig. 4: Sample execution of the system in Figure 3. Produced with Open Modelica [13].

For the purposes of co-simulating the above system using the FMI Standard, suppose that the physical subsystem is decomposed into $N > 1$ FMUs, connected sequentially, as shown in Figure 5. The Software FMU implements the simulation of the software subsystem, in Figure 3. FMU 1 is responsible for the dynamics

of the physical subsystem in the same figure, and the other FMUs are identity functions and referred to as propagate FMUs.

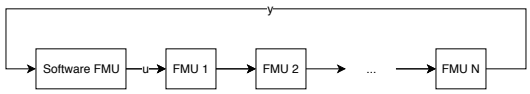


Fig. 5: Co-Simulation Scenario.

Using the Jacobi orchestration algorithm, introduced in Section 2.1 and summarized in Figure 1b, to co-simulate the scenario in Figure 5, with $N = 3$ and co-simulation step size $H = 0.01$, leads to the software execution trace depicted in Figure 6. The events produced in this trace are the same as the ones in the correct execution in Figure 4, but their timestamps are different. Event e1 is produced at time 0.02s instead of 0.01s, and detected later at time 0.06s, instead of 0.02s.

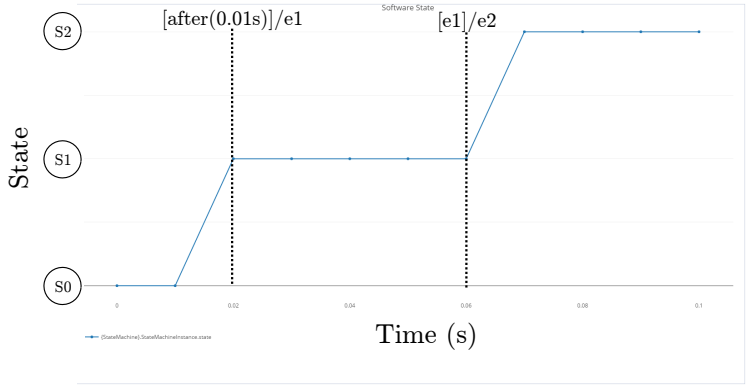


Fig. 6: Co-simulation using the Jacobi algorithm of the scenario in Figure 5. Parameters: $N = 3, H = 0.01$. Produced with Maestro from INTO-CPS [29].

The extra delay in the events is a well known feature of Jacobi based co-simulation. It is also well known that the smaller the communication step size H, the smaller the delay introduced. What this example illustrates is that the size of the co-simulation scenario also plays a role in the delay introduced. In fact, by adding more propagate FMUs to the example scenario, we get a qualitatively different software execution trace, as shown in Figure 7, where the final state of the software subsystem is S3, instead of S2. The excessive delay, accidentally introduced by the Jacobi algorithm, caused the software timeout to be triggered.

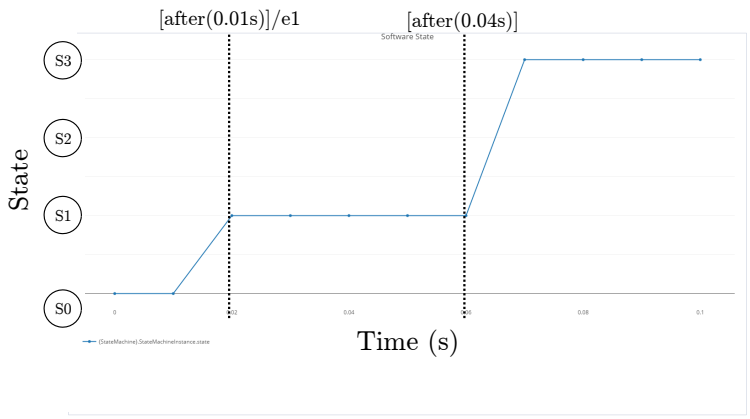


Fig. 7: Co-simulation using the Jacobi algorithm of the scenario in Figure 5. Parameters: $N = 6, H = 0.01$. Produced with Maestro from INTO-CPS [29].

In general one would like to have co-simulations that either do not introduce an artificial delay, or that, at least, introduce a delay that preserves the sequence of events. In the following subsections we use model checking to formally study the preservation of this property for the hybrid system shown in Figure 3, with a variable structure co-simulation scenario illustrated in Figure 5. In the experiments the co-simulation step size is kept the same, although it is straightforward to take its variation into account.

3.2 Model Checking the Jacobi Algorithm

We use the ProMeLa [21] notation to model the FMUs, and the orchestration algorithm. The Promela language uses a textual syntax to describe parallel and sequential processes, communication channels, and non-determinism.

The Promela model follows closely the co-simulation scenario sketched in Figure 5. The communication between the orchestration algorithm and the FMU's is made via three channels: one to set inputs, one to set outputs, and one to perform a co-simulation step. These channels are detailed in Listing 1.1. The `in` and `step` channels are read by the FMU, while the `out` channel is read by the orchestration algorithm.

Listing 1.1: Channels

```
1 mtype:events = {e0, e1};
2 typedef channels {
3     chan in = [0] of {mtype:events};
4     chan out = [0] of {mtype:events};
5     chan step = [0] of {int};
6 }
```

The FMU corresponding to the software subsystem is modelled in ProMeLa by implementing the reaction to events in the channels `in` and `step`. When an

event is present in channel `in`, it is just stored in an interval variable. When an event is present in channel `step`, the FMU just follows the state machine of the software subsystem, taking into account that the time is represented as an integer and the communication step size is 0.01s. Listing 1.2 presents this model.

Listing 1.2: Statechart FMU

```
proctype stateFMU(channels chans) {
  int t_time = 0;
  mtype:events input;
  do
  :: chans.step ? t_time ->
    if
    /* if state is 0 and 2 time units have passed, then change the state to 1 and
        ↪ output an event. */
    :: (state == 0) ->
      if
      :: (t_time == 1) ->
          state=1;
          chans.out ! e1; /* e1 is the output that we are interested in receiving
              ↪ again */
      :: else -> chans.out ! e0;
      fi;

    /* If the state is 1 and 4 time units have passed, then change to state 3 */
    :: (state == 1) ->
      if
      :: t_time == 4 & input != e1 -> state = 3;
      :: input == e1 -> state = 2;
      :: else -> skip;
      fi;
      chans.out ! e0;
    :: (state == 2) -> chans.out ! e1;
    :: else -> chans.out ! e0;
    fi;
  :: chans.in ? input
  :: (terminate == 1) -> break;
  od;
}
```

The other FMUs are abstractions of the physical subsystem, containing only the behavior that is of interest to verify this property. We are interested in the propagation of any change in the input. As such, the FMU model shown in Listing 1.3 just stores and outputs whatever input it receives.

Listing 1.3: Propagate FMU

```
proctype propFMU(channels chans){
  mtype:events inp;
  int t_time = 0;
  do
  :: chans.in ? inp
  :: chans.step ? t_time ->   chans.out ! inp;
  :: (terminate == 1) -> break;
  od;
}
```

The Jacobi master algorithm essentially sends events through the `in` channel of each FMU, asks the FMU to step via the `step` channel, and stores the output events at the `out` channels. The non-deterministic aspect of this model is encoded in the choice of the number of propagate FMUs that can be added to

the scenario. The number of FMUs (maxN) is limited to 10, as it is enough to prove this property. The implementation is shown in Listing 1.4.

Listing 1.4: The Jacobi Master Algorithm in ProMeLa

```
proctype MAJacobi(){
  int propagateCount;
  select ( propagateCount : 1 .. (maxN-1) );
  int FMUCount = propagateCount + 1;

  channels fmuChannels[maxN];
  mtype:events inputs[maxN];

  smpid = run stateFMU(fmuChannels[0]);

  int i;
  for(i : 1 .. propagateCount){
    run propFMU(fmuChannels[i]);
  }

  do
  :: time < endTime  ->
    /* Step the FMUs */
    for(i : 0 .. FMUCount-1){
      fmuChannels[i].step ! time+1;
    }

    /* Retrieve the outputs */
    for(i : 0 .. FMUCount-1){
      fmuChannels[i].out ? inputs[(i + 1) % (FMUCount)];
    }

    /* Set inputs */
    for(i : 0 .. FMUCount-1){
      fmuChannels[i].in ! inputs[i]
    }

    time++;
  :: else ->
    terminate = 1;
    break;
  od;
}
```

The event preservation property can be encoded in this model as a reachability property: the Statechart FMU eventually reaches S2. This is shown in Listing 1.5. The `state` variable is global, and is set as part of the execution of the FMU in Listing 1.2.

Listing 1.5: Eventually Correct LTL formula.

```
ltl eventuallyCorrect { <> (state == 2)}
```

Using SPIN to carry out the verification of this property, applied to Listing 1.4, quickly shows that it cannot be verified. The error trail provides a counter example execution, by showing that S3 is reached when there are three propagate FMUs. Informally, the error trail is the following: At step 1 (0.1ms), e1 is outputted from the Statechart FMU. At step 2 (0.2ms) it is outputted from the following propagate FMU. At step 3 it is outputted from the second propagate FMU. Finally, at step 4 it is outputted from the last propagate FMU but this is the same time as the Software FMU transitions to S3. Therefore, the Statechart FMU never reaches S2.

3.3 Model Checking the Gauss-Seidel Algorithm

The Gauss-seidel orchestration algorithm is introduced in Section 2.1 and illustrated in Figure 2a. The main difference between this algorithm and the Jacobi is in the timestamp of the outputs and inputs provided to the simulation units. From the perspective of a simulation unit, the Gauss-seidel provides future inputs to the unit, before asking it to compute a co-simulation step. This allows the unit to react to the inputs without any delay [17]. Its implementation is detailed in Listing 1.6.

Listing 1.6: The Gauss-Seidel Master Algorithm in ProMeLa

```
1  proctype MAGauss(){
2    int propagateCount;
3    select ( propagateCount : 1 .. (maxN-1) );
4    int FMUCount = propagateCount + 1;
5
6    channels fmuChannels[maxN];
7    mtype:events inputs[maxN];
8
9    run stateFMU(fmuChannels[0]);
10
11   int i;
12   for(i : 1 .. FMUCount-1){
13     run propFMU(fmuChannels[i]);
14   }
15
16   do
17   :: time < endTime  ->
18     for(i : 0 .. FMUCount-1){
19       /* Step the FMU */
20       fmuChannels[i].step ! time + 1;
21
22       /* Retrieve the output */
23       fmuChannels[i].out ? inputs[(i + 1) % FMUCount];
24
25       /* Set the input */
26       fmuChannels[(i + 1) % FMUCount].in ! inputs[(i + 1) % FMUCount]
27     }
28     time++;
29   :: else ->
30     terminate = 1;
31     break;
32   od;
33 }
```

Verifying Listing 1.6 with the LTL formula in Listing 1.5 shows that the Gauss-seidel algorithm correctly preserves the execution sequence of the events.

4 Discussion and Future Work

In this paper we show how a co-simulation of a hybrid system can be incorrect and we sketch a potential solution that respects the black box nature of co-simulation.

The correctness property we used is a weak form of event synchrony: the order of events happening in the software subsystem is preserved, but their time stamp can change. Two orchestration algorithms have been used to study the

property: The Jacobi and the Gauss-Seidel. It is shown that the Jacobi algorithm does not preserve it, in general making it unsuitable for hybrid co-simulation.

Albeit a very simple example, the hybrid system used in the previous section is meant to prove that a co-simulation algorithm is wrong. It can be used to prove that a co-simulation algorithm is correct but the family of hybrid systems it represents is so narrow that the utility of this result is negligible. In the future, we intend to use the non-determinism in the construction of more complex software and physical subsystems. That way, we can increase the usefulness of a positive result. Additionally, we intend to explore how to deal with black box simulation units, so that a conservative abstraction can be built for these.

What our work proves is that in order to safely use the Jacobi algorithm, some knowledge is required about the FMU's. In particular, in black box co-simulation, we hypothesize that knowing the shortest timeout of each software FMU, and knowing the input-to-output propagation time of each FMU, is enough to determine which communication step size can be used in order to ensure the preservation of the event sequence. To see how the step size H can be computed, let T denote the smallest timeout used in the software FMU, and $P(H)$ denote the largest propagation time from any output to itself, for the communication step size H. For the Jacobi algorithm and the scenario in Figure 5, $P(H) = H \times (N+1)$. Then the communication step size must be chosen so that $P(H) < T$.

If the above hypothesis turns out to be correct, then this means that the Jacobi algorithm can still be used in black box co-simulations, since the shortest timeout time does not expose the Intellectual Property of the subsystems.

The above reasoning is common in research on black box co-simulation (e.g., exposing the Jacobian [28], exposing the I/O feedthrough [2], exposing the maximum allowed step size [7]). First, researchers find an example whose co-simulation is wrong. Then they pin-point the minimum information that needs to be exposed in order to have a correct co-simulation, or, at least, to detect the problem.

The FMI webpage[7] contains a list of tools capable of performing co-simulation, and in order to be on this list, a tool must pass some tests. These tests, however, are limited – for example they only concern simulation of a single FMU, and not an actual co-simulation. In the long term, this research aims at producing a set of benchmarks, for various correctness properties, that can be used by the research community in the development of co-simulation tools. This idea is inspired by the work of [8], which defined the building blocks of these benchmarks.

References

1. Arnold, M.: Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models. Journal of Computational and Nonlinear Dynamics 5(3), 9 (may 2010)
2. Arnold, M., Clauß, C., Schierz, T.: Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-Simulation v2.0. In: Schöps, S., Bar-

[7] http://fmi-standard.org/

tel, A., Günther, M., ter Maten, W.E.J., Müller, C.P. (eds.) Progress in Differential-Algebraic Equations. pp. 107–125. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
3. Aström, K.J., Wittenmark, B.: Computer-controlled systems: theory and design. Courier Corporation (2011)
4. Benedikt, M., Watzenig, D., Zehetner, J., Hofer, A.: 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. pp. 1–12. Ibiza, Spain (jun 2013)
5. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauss, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.V., Wolf, S.: The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In: 8th International Modelica Conference. pp. 105–114. Linköping University Electronic Press; Linköpings universitet, Dresden, Germany (jun 2011)
6. 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, Munich, Germany (nov 2012)
7. Broman, D., Brooks, C., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., Wetter, M.: Determinate composition of FMUs for co-simulation. In: Eleventh ACM International Conference on Embedded Software. p. Article No. 2. IEEE Press Piscataway, NJ, USA, Montreal, Quebec, Canada (2013)
8. Broman, D., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., Wetter, M.: Requirements for Hybrid Cosimulation Standards. In: 18th International Conference on Hybrid Systems: Computation and Control. pp. 179–188. HSCC '15, ACM New York, NY, USA, Seattle, Washington (2015)
9. Busch, M.: Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error. ZAMM - Journal of Applied Mathematics and Mechanics 96(9), 1061–1081 (sep 2016)
10. Cellier, F.E., Kofman, E.: Continuous System Simulation. Springer Science & Business Media (2006)
11. Clarke, E.M., Veith, H.: In: Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday. Lecture Notes in Computer Science, vol. 2772, pp. 208–224. Springer (2003)
12. Cremona, F., Lohstroh, M., Broman, D., Lee, E.A., Masin, M., Tripakis, S.: Hybrid co-simulation: it's about time. Software & Systems Modeling (nov 2017), http://link.springer.com/10.1007/s10270-017-0633-6
13. Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman, D., Sandholm, A.: Openmodelica - a free open-source environment for system modeling, simulation, and teaching. In: 2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control. pp. 1588–1595 (Oct 2006)
14. 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. SCSC '07, Society for Computer Simulation International San Diego, CA, USA, San Diego, CA, USA (jul 2007)

15. 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, Trento, Italy (2018), http://link.springer.com/10.1007/978-3-319-74781-1{_}24
16. Gomes, C., Legat, B., Jungers, R.M., Vangheluwe, H.: Stable Adaptive Co-simulation : A Switched Systems Approach. In: IUTAM Symposium on Co-Simulation and Solver Coupling. p. to appear. No. 1, Darmstadt, Germany (2017)
17. Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., De Meulenaere, P.: Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators. SIMULATION pp. 1—-29 (2018)
18. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: State of the art. Tech. rep. (feb 2017), http://arxiv.org/abs/1702.00686
19. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a Survey. ACM Computing Surveys pp. accepted, to appear. (2018)
20. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (jun 1987)
21. Holzmann, G.: The model checker SPIN. IEEE Transactions on Software Engineering 23(5), 279–295 (may 1997), http://ieeexplore.ieee.org/document/588521/
22. Kalmar-Nagy, T., Stanciulescu, I.: Can complex systems really be simulated? Applied Mathematics and Computation 227, 199–211 (jan 2014)
23. Kübler, R., Schiehlen, W.: Modular Simulation in Multibody System Dynamics. Multibody System Dynamics 4(2-3), 107–127 (aug 2000)
24. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC). pp. 363–369 (2008)
25. Sadjina, S., Kyllingstad, L.T., Skjong, S., Pedersen, E.: Energy conservation and power bonds in co-simulations: non-iterative adaptive step size control and error estimation. Engineering with Computers 33(3), 607–620 (jul 2017)
26. Sadjina, S., Pedersen, E.: Energy Conservation and Coupling Error Reduction in Non-Iterative Co-Simulations. Tech. rep. (jun 2016), http://arxiv.org/abs/1606.05168
27. Schweizer, B., Li, P., Lu, D.: Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches. Journal of Computational and Nonlinear Dynamics 10(5), 051007 (sep 2015)
28. Sicklinger, S., Belsky, V., Engelmann, B., Elmqvist, H., Olsson, H., Wüchner, R., Bletzinger, K.U.: Interface Jacobian-based Co-Simulation. International Journal for Numerical Methods in Engineering 98(6), 418–444 (may 2014)
29. Thule, C., Lausdahl, K., Larsen, P.G., Meisl, G.: Maestro: The into-cps co-simulation orchestration engine (2018), submitted to Simulation Modelling Practice and Theory
30. Tomiyama, T., D'Amelio, V., Urbanic, J., ElMaraghy, W.: Complexity of Multi-Disciplinary Design. CIRP Annals - Manufacturing Technology 56(1), 185–188 (2007)
31. Van der Auweraer, H., Anthonis, J., De Bruyne, S., Leuridan, J.: Virtual engineering at work: the challenges for designing mechatronic products. Engineering with Computers 29(3), 389–408 (2013)
32. Vangheluwe, H., De Lara, J., Mosterman, P.J.: An introduction to multi-paradigm modelling and simulation. In: AI, Simulation and Planning in High Autonomy Systems. pp. 9–20. SCS (2002)

33. Zhang, F., Yeddanapudi, M., Mosterman, P.J.: Zero-Crossing Location and Detection Algorithms For Hybrid System Simulation. In: IFAC Proceedings Volumes. vol. 41, pp. 7967–7972. Elsevier Ltd, Seoul, Korea (jul 2008), http://linkinghub.elsevier.com/retrieve/pii/S1474667016402296