Coupling Petri nets with Deterministic Formalisms Using Co-simulation

David P.Y. Lawrence⋆, Cláudio Gomes‡, Joachim Denil‡, Hans Vangheluwe‡†
⋆University of Geneva, Carouge, Switzerland
‡University of Antwerp, Antwerp, Belgium
†McGill University, Montreal, QC, Canada
david.lawrence@unige.ch, {claudio.gomes, joachim.denil, hans.vangheluwe}@uantwerpen.be

ABSTRACT
Modelling cyber-physical systems is often seen as a highly multi-disciplinary activity. Therefore, efficient methodologies are required to be able to represent environment, plant and control models using the most appropriate formalisms. On the one hand, the Petri net formalism is appropriate to model the environment of a complex cyber-physical system, especially since it inherently supports non-determinism and concurrency. On the other hand, plant models, based on the laws of physics, are often represented using differential equations. In order to produce a relevant simulation of the overall system, the controller, the plant and the environment models are composed. Co-simulation and its industrial standard, the Functional Mock-up Interface, is one such generic technique that allows for the coupling of different executable models. However, coupling a non-deterministic model with other types of models would be equivalent to build the state space of the given composed system, which in some cases is too complex. In this paper we discuss the co-simulation of a Petri net environment model with a system described by causal block diagrams. To reduce the size of the state space explored, we use a constraint language based on automata and trace matching. The approach is applied to the simulation of a simplified train system.

Author Keywords
co-simulation; fmi; non-determinism; petri-nets; causal block diagrams.

ACM Classification Keywords
I.6.1 SIMULATION AND MODELING (e.g. Model Development).: See: `http://www.acm.org/about/class/1998/` for more information and the full list of ACM classifiers and descriptors.

1. INTRODUCTION

Cyber-physical systems are characterized by the tight integration of computation, communication and physical components. Such systems can be found in areas like automotive, aero-space, etc. Model-based techniques have become a popular approach for the development of these systems. In fact, these techniques allow for early decision making, by integrating and simulating a virtual prototype of the system under design. The models do not solely represent the control algorithm and the physical plant to control but also the environment, in which the system operates. This environment is by nature concurrent and non-deterministic. For example, autonomous driving systems work in a highly dynamic environment where pedestrians and other traffic participants act independently and asynchronously from each other [18].

Because Petri nets inherently support concurrency and non-determinism, it is a very appropriate formalism to model the environment of a complex cyber-physical system. The plant model, based on the laws of physics, is often represented using differential equations. The control model is mostly represented using differential or difference equations and/or discrete-event formalisms. Because of the heterogeneity of these different types of models, techniques are needed to integrate and simulate them together.

Co-simulation in general, and the Functional Mock-up Interface (FMI) specifically, is a technique to couple multiple simulators by establishing a common communication interface [5]. The need for co-simulation is usually a pragmatic one. In an industrial setting, many different tools used by different, specialized teams need to be integrated [4]. Co-simulation offers a scalable solution for this problem at any stage of the development process [22]. The use of co-simulation has another advantage. Because of intellectual property rights of the different stakeholders involved in building the system, the implementation details of the model can be hidden. FMI hides these implementation details by exposing only a common interface and extra information to allow communication between simulators.

Unfortunately, the co-simulation of the system and the non-deterministic environment is equivalent to building all possible scenarios of environment signals sequences. In other words, this is equivalent to constructing the entire state space. Obviously, this is highly impractical due to the state space explosion problem. However, one can guide the co-simulation to specific signals sequences of the environment and therefore constrain the scenarios selected, reducing the quantity of paths explored during the co-simulation. Additionally, this trimming process can realize simulation intents, hiding therefore uninteresting behaviors.

We summarize the contributions of this paper: (a) We demonstrate the meaning of coupling a non-deterministic formalism with deterministic formalisms using co-simulation. (b) We

SpringSimSim-TMSDEVS 2016 July 21-26 Pasadena, CA, USA

show how to construct a master algorithm to couple these co-simulation units together using the FMI standard. (c) Finally, we define a constraint language, based on automata and trace matching, that helps in pruning part of the state space.

The rest of the paper is organized as follows: Section 2 gives an overview of the work related to this contribution. Section 3 introduces the running example of this paper, the coupling of an environmental model of the railway signaling system with a model of the physics and control of a train. Section 4 introduces the different FMUs of the running example. Section 5 shows how to compose the non-deterministic and deterministic FMU. Section 6 defines a constraint language to prune part of the state space. We discuss the approach in Section 7 Finally, we conclude in Section 8.

2. RELATED WORK

To the best of our knowledge, there is no approach that leverages black box co-simulation for the verification of systems, effectively lowering the entry barrier for the application of verification techniques in a tool agnostic manner.

In the domain of non-exhaustive system verification, non-deterministic models have been used extensively. For a broad overview of approaches for the verification of hybrid systems, please see [1]. In this domain, – in which our approach belongs –, the work closest to ours is [14], extended in [15]. In [14], the system to be verified is also assumed to be a black-box, i.e., only its outputs are observed, and all possible traces of the simulation across a bounded amount of time are also computed in a parallel architecture. They deviate from our work because they require the disturbance model to be based in Finite State Automata, that has to be available to the tool. In contrast, we use Petri nets but most importantly, we present in detail the framework (synchronization algorithm and semantic adaptations) for co-simulation that can be reused to couple other non-deterministic simulators for verification.

The advantage of the work developed in [14] is that, because they have knowledge about the disturbance model, they develop sophisticated techniques for optimizing the simulation and they can predict the total number of scenarios up-front, thus allowing for a graceful degradation (see [15]) in case the verification is aborted. The graceful degradation provides a useful upper-bound on the probability that the system will fail the verification process with the scenarios that where not simulated yet. These techniques make their approach scalable for large disturbance models. In contrast, because we allow verification without access to the disturbance model, it is hard to optimize the number of possible scenarios. To mitigate this, we provide a mechanism to trim the scenarios explosion but that still requires some knowledge about the simulators involved, at least the possible outputs of the disturbance model.

We distinguish our work from [9] by the fact that a stochastic process (see [24] for a good introduction) is used to select from all possible scenarios, a relevant few. Because we do not assume any knowledge about the disturbance model, it is hard to apply these techniques in our work. Thus, at every simulation step, we explore all possible cases, that can be filtered by the constraint language that we support.

In the software testing domain, Petri nets have been used extensively to generate test scenarios for the system under verification, as it was done with high level Petri nets such as CO-OPN in [13, 16], for example.

In the works of [23], a predicate/transition net is used to model the behavior of the system under test. The test scenarios are then generated from this model. To contrast with our approach, we only need information about the outputs and inputs of the disturbance model. This also means that we are not capable of automatically selecting interesting scenarios to simulate the system.

In the domain of model checking, Edmund Clarke et al. addressed the idea of using a *CEGAR* approach to perform model checking on hybrid automata in [10], therefore transforming the infinite state space of the hybrid system to a finite one through abstraction. The key of this approach is obviously the abstraction methods, that we are unable to perform efficiently in our approach considering that models are seen as black box, and have been extensively studied in various papers [2, 3, 19].

Although the work presented in [17] uses model checking techniques, it isn't actually exploring exhaustively the state space of a hybrid system. In previous work, they use motion planning to explore the state space of a hybrid system in their approach *HyDice*. To improve that approach, they proposed *TemporalHyDICE* in which they extrapolate atomic propositions from hybrid traces and perform LTL falsification on designed properties, improving therefore their performance. This work is close to what was explored in this paper, even though the two approaches diverge by the way the state space is explored.

In summary, our work presents interesting challenges in the application of the techniques of the state of the art in system verification and software testing. The distinguishing factors of our approach are:

1. Both the disturbance model and the system under verification are black boxes.
2. The output of the system can influence the disturbance model through feedback loops.
3. By taking advantage of the FMI standard, we lower the barrier for these techniques to be applied in the industry.

3. RUNNING EXAMPLE

To show the contribution of this paper, we use a running example of a driverless train that reacts to the traffic lights signals at each railway segment. The control system and the plant of the train, are modeled with Causal Block Diagrams (CBD). CBDs are a general-purpose formalism used for the modeling of causal systems. The traffic light as the environment for this system is represented using a Petri net.

3.1 Train Control System and Plant

The control system regulates the speed of the train and ensures a comfortable ride for the passengers. As a simplification we assume that the railway is a straight line, divided into equally sized segments. At the end of each segment there is a

traffic light that informs the train if it can enter the next segment. We abstract a train as a mass m_{train} moving along one dimension, under the influence of a traction force $F_{traction}$, with a single standing passenger in it. The passenger is modeled as a mass-spring-damper system $m_{passger}$ attached to a pole, which in turn is fixed to the train. There is no friction between the masses $m_{passger}$ and m_{train}. If the maximum displacement of the passenger exceeds $0.6m$ from the equilibrium, then she/he has fallen. The control system is a simple PID Controller that regulates the traction force $F_{traction}$ given the ideal velocity of the train v_{ideal}.

In summary, we have the following system of first order differential equations:

$$\begin{cases} e & = v_{ideal} - v \\ F_{traction} & = K_p \cdot e + K_i \cdot \int e\, d\tau + K_d \cdot \frac{de}{dt} \\ F_{spring} & = k(-x_{passger}) \\ F_{damper} & = c(-v_{passger})) \\ \frac{dv_{passger}}{dt} & = \frac{F_{spring}+F_{damper}-m_{passger}*\frac{F_{traction}}{(m_{train}+m_{passger})}}{m_{passger}} \\ \frac{dv_{train}}{dt} & = \frac{F_{traction}}{(m_{train}+m_{passger})} \\ \frac{dx_{passger}}{dt} & = v_{passger} \\ \frac{dx_{train}}{dt} & = v_{train} \end{cases}$$

Note that the solutions for these equations describe the dynamics of the train as functions of time (signals). The ideal train velocity v_{ideal} is given as an input to the CBD, adapted from the traffic light system. The parameters that we found sensible for this scenario are: $m_{passger} = 73kg$; $m_{train} = 6000kg$; $k = 300$; $c = 150$; $K_p = 500$; $K_i = 0$; $K_d = 100$. Because of the space constraints we do not include the complete CBD model but we refer the reader to **http://msdl.cs.mcgill.ca/people/joachim/publ/train_control_system.pdf**. The CBD model contains a fixed-step numerical solver to discretize the differential equation. Therefore, the CBD model is executed with a relatively small time step (0.001s).

3.2 Light Signaling System

Figure 1 depicts a simplified model of the railroad traffic lights signals in Belgium[1]: 1. the red means to stop; 2. the green means to continue at full speed; 3. the yellow means that at the next segment, a red *might* be shown so the train must adapt its speed. We opted for the simplified traffic light signal to keep the meaning of the Petri net clear. We implemented the full traffic light model (with 12 possible signals). The model is available at **http://msdl.cs.mcgill.ca/people/joachim/publ/PetriNetTrainModel.zip**.

The Petri net model, shown in Figure 1, is non-deterministic, i.e., multiple transitions can be fired from some markings. For example, in its current state, two transitions can fire: `r2g` and `r2y`. This means that two possible markings can be reached from the current state of the Petri net. The operational semantics of a Petri net specify that the next state after a transition is the set of all possible markings. A marking is a set of integer variables, one for each place, indicating the number of tokens in that place. For the considered Petri net model, the marking

[1] **http://www.infrabel.be/en/about-infrabel/safety/railway-signalling-systems**

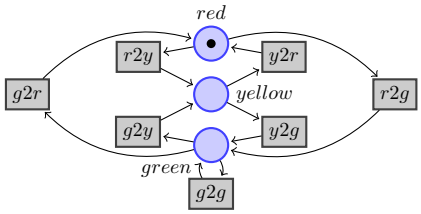


Figure 1: Non-deterministic environment model

consists of a single token in the place named *red* and none in the others.

3.3 Co-Simulation Scenario

To simulate the interactions between the two models, two challenges remain: 1. Define the meaning of the coupling between a deterministic model – the CBD – with a non-deterministic one – the Petri net. 2. Connect and adapt the inputs and the outputs of both models.

In this work we couple the two models by exporting them, along with the simulator code, as Functional Mockup Units (FMU) [2] [5].

Intuitively, the Petri net output is the set of markings in which the Petri net is. Because each of those markings dictates the ideal velocity to be set in the CBD model, we shift from a linear time model - traditional to co-simulation scenarios - to a branching time model. This implies that the meaning of a co-simulation scenario that contains one or more Petri nets in it, is no longer a single trace for each signal, but the set of all possible traces. Operationally, each time the traffic light model is simulated, it fires all the enabled transitions; and for each possible marking, an alternate co-simulation *branches* off the current one, with that marking as the input for the CBD.

At time t, and for each marking m of the Petri net, the input to the CBD model (v_{ideal}) is given as:

$$v_{ideal}(t) = \begin{cases} 27m/s & \text{if } m(t) = (1,0,0) \\ 15m/s & \text{if } m(t) = (0,1,0) \\ 0m/s & \text{if } m(t) = (0,0,1) \end{cases} \quad (1)$$

In our traffic light example, it does not make sense to perform a simulation step in the Petri net every time the train model is simulated because the traffic light would change *at every simulation step* and the train model needs to be simulated at very small time intervals. The simplest way to solve this problem is to perform a multi-rate adaptation: the Petri net model is executed every 60 seconds whereas the train model is executed every 0.001 seconds. However this adaptation is not valid for our example because we assume that the segments of the railway are equidistant and that there is a traffic light at the end of each segment. This means that, depending on the velocity, the train can reach the traffic light sooner or later (or never, in case the traffic light was red), but not exactly every 60 seconds. The distance covered by the train x_{train}

[2] Functional Mockup Units are, in a very simplified way, executable units that obey the Functional Mockup Interface standard, making it possible to link them together, plus a master algorithm, to co-simulate them.

thus controls the execution of the Petri net simulator: whenever the train crosses 500 meters, the Petri-net simulator is executed and synchronized with the CBD simulator.

Because we simulate the traffic light model at a lower rate than the train model, the output of traffic light $m(t)$, and consequently v_{ideal}, is undefined most of the time steps in which the CBD is simulated. To ensure that we have well defined inputs whenever we simulate the train model, the most recently defined output of the traffic light model is used[3]

All possible traces of the co-simulation scenario described in this section, across 80 seconds, are depicted in Figure 3a. The branching time of the traces becomes obvious in the plot of the displacement of the train.

4. FUNCTIONAL MOCKUP UNITS

In this section we introduce the two FMU classes of the co-simulation scenario.

4.1 Petri Nets

The formalization of a subset of FMU was already presented in [20]. Hence, the latter can be adapted to represent Petri nets.

DEFINITION 1 (PNFMU). *A PNFMU is a tuple* $N = \langle S, U, Y, D, s_0, set, get, doStep \rangle$. *In order to completely define a PNFMU, universes of input and output variables must be defined as follow:*

- $\mathbf{U}$ *is the universe of input variables;*
- $\mathbf{Y}$ *is the universe of output variables.*
- $\mathbf{U} \cap \mathbf{Y} = \emptyset$

A PNFMU is therefore defined as follow:

- *a set of markings* S*;*
- *a set of input places* $U \subseteq \mathbf{U}$*;*
- *a set of output places* $Y \subseteq \mathbf{Y}$*;*
- *a set of output-input dependencies* D*;*
- *an initial marking* $s_0 \in S$*;*
- *a function* $set : \mathcal{P}(S) \times U \times \mathbb{N} \to \mathcal{P}(S)$ *that sets the number of tokens of an input place;*
- *a function* $get : \mathcal{P}(S) \times Y \to \mathcal{P}(\mathbb{N})$ *that returns the number of tokens of an output place;*
- *a function* $doStep : \mathcal{P}(S) \times \mathbb{R} \to \mathcal{P}(S) \times \mathbb{R}$ *that makes an evolution step of the FMU. Given a set of states* $s \in \mathcal{P}(S)$ *and a time step* $h \in \mathbb{R}$*,* $doStep(s, h)$ *returns a pair (s',h'),* $s' \in \mathcal{P}(S)$ *and* $h' \in \mathbb{R}$*, such that* $h' = h$ *and s' is the set of reachable markings after firing all possible transitions once.*

Please take note of the following remarks: 1. The semantics of the $doStep$ is the Petri net semantics; 2. The time step h isn't taken into account in the $doStep$: it will still fire all enabled transitions, once; 3. The state of a PNFMU is a set of markings since at each $doStep$ all enabled transitions are fired; 4. The function set will set a number of tokens of an input place to all markings of the Petri net. As for the get function, it returns a set of values and not a single value; 5. Since a PNFMU can encapsulate a set of markings, we defined the functions set, get and $doStep$ with the power set $\mathcal{P}(*)$ in their domain and co-domain.

[3]This technique is also known as zero-order hold extrapolation.

4.2 Causal Block Diagrams

As we desired to model our system using Causal block diagrams, one must also formalize CBD as it was done with Petri nets.

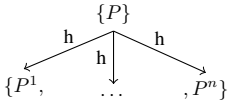
DEFINITION 2 (CBDFMU). *A CBDFMU is a tuple* $C = \langle S, U, Y, D, s_0, set, get, doStep \rangle$. *In order to completely define a CBDFMU, we will reuse universes of inputs and outputs defined previously for PNFMU. A CBDFMU can be defined as follow:*

- *a set of states* S*;*
- *a set of input variables* $U \subseteq \mathbf{U}$*;*
- *a set of output variables* $Y \subseteq \mathbf{Y}$*;*
- *a set of output-input dependencies* D*;*
- *an initial state* $s_0 \in S$*;*
- *a function* $set : S \times U \times \mathbb{V} \to S$ *that sets the value of an input variable;*
- *a function* $get : S \times Y \to \mathbb{V}$ *that returns the value of an output variables;*
- *a function* $doStep : S \times \mathbb{R}_{\geq 0} \to S \times \mathbb{R}_{\geq 0}$ *that makes an evolution step of the FMU. Given a state* $s \in S$ *and a real time step* $h \in \mathbb{R}_{\geq 0}$*,* $doStep(s, h)$ *returns a pair (s',h'),* $s' \in S$ *and* $h' \in \mathbb{R}_{\geq 0}$*, such that s' is the resulting CBD state after a simulation step of time h from the state s.*

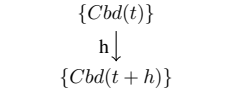
5. COMPOSITION

5.1 Non-determinism

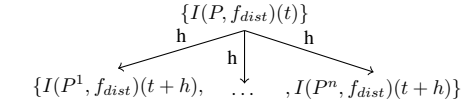
Based on the previous formalization, we illustrate here how a single *PNFMU*, denoted as P, evolves through time. At each step of the evolution, every enabled transitions, up to n, is fired from the current marking, therefore creating new *PNFMU*s encapsulating the new markings:



As for a CBD, if a (time)step h is simulated, only a single state is produced:



Because the *PNFMU* is composed with the *CBD*, we need to perform two semantic adaptations on the Petri net. The first adaptation changes the output data from the Petri net to the CBD. The second adaptation corresponds to a control/time adaptation. Indeed, we have to define when environment signals are forwarded to the CBD. In the light signaling system, we send environment signals whenever the train crosses a segment $(500m)$. This adaptation is therefore a function f_{dist} based on the distance travelled by the train. This set of adaptations are represented here with the function I:



where applying the function f_{dist} returns a time of h.

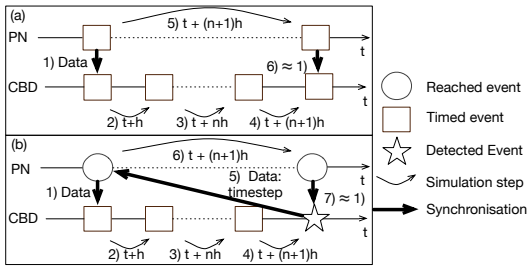
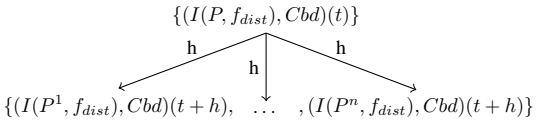


Figure 2: Synchronization Diagrams for the co-simulation

Therefore, the co-evolution of the Petri net and the CBD composition will end up in a set of tuple of FMU instances:



5.2 Master Algorithm

The master algorithm is responsible for the orchestration of the co-simulation. We describe two different master algorithms based on the canonical synchronization algorithm described in [12]. The first one, illustrated in Figure 2 (a) describes the naive approach described in Section 3.3. This situation occurs when no feedback loop is present between the system and the environmental model. The master algorithm communicates the adapted Petri net state to the CBD. In Figure 2 (a) the Petri net and CBD simulators are synchronized at time t. Then, the CBD solver is allowed to compute its state until the time $t + (n + 1)h$, where n is a parameter denoting the number of steps allowed before synchronization occurs. Afterwards, the Petri net catches up to the CBD. The second synchronization algorithm, shown in Figure 2 (b), ensures a more realistic co-simulation by detecting when the train crossed a segment (state event detection), and instructing the Petri net to catch up. A feedback loop is thus created between the system and its environment. The master algorithm detects the crossing of the threshold and communicates the size of the time step required in the Petri net. The synchronization algorithm can be extended with state event location, the detection of the event within a certain boundary. This requires the roll-back capabilities in the CBD simulator. When the semantic adaptation requires both an event-based and time-based adaptation, both synchronization algorithms must be combined.

Algorithm 1 shows the simplified pseudocode to orchestrate the complete co-simulation with the second synchronization mechanism. In this case we use a threaded approach to orchestrate the multiple co-simulation instances. If another parallelization mechanism is preferred, the algorithm should be adapted accordingly. Each thread executes the *masterFunction* and has its own *CoSimulationInstance* data, containing the FMUs and other needed data. The thread runs until it reaches the experiment *STOPTIME*. In each step, the CBD simulation is executed. A state event detector checks if the train exceeded the predefined distance. If an event is detected, the Petri net FMU is executed. We use the *fmi2GetFMUstate* to get the different markings of the Petri net. The first time the master calls this function, the FMU returns the state of the first marking (used by the current thread). The subsequent calls return the other markings until it becomes undefined. For each of these subsequent states, we create a new thread with that Petri net state and a copy of the CBD state. Because the new thread executes the same step again, we cannot execute the PN FMU and CBD FMU again.

```
Function masterFunction(master: coSimInstance) : int is
    pnData ← Get and adapt PN output data;
    while master.currentTime < STOPTIME do
        if not master.cbdExecuted then
            set CBD input data to pnData;
            doStep(master.CBD, ...);
            master.cbdExecuted ← true;
            cbdData ← Get and adapt CBD output data;
        end
        if eventDetection(cbdData) then
            doStep(master.PN,...);
            selfNewState ← fmi2GetFMUstate(master.PN);
            branchState ← fmi2GetFMUstate(master.PN);
            while branchState is defined do
                new ← createNewCoSimInstanceFromCurrent();
                CBDState ← fmi2GetFMUState(master.CBD);
                fmi2SetFMUState(new.CBD,CBDState);
                fmi2SetFMUState(new.PN,branchState);
                createThread(masterFunction,new);
                ...;
                branchState ← fmi2GetFMUstate(master.PN);
            end
            pnData ← Get and adapt PN output data;
        end
        master.cbdExecuted ← false;
        Increment time;
        ...;
    end
    wait for joining threads;
end
```

Algorithm 1: Master Algorithm Example

Figure 3a shows the plots of executing the case study with this described master algorithm.

6. CONSTRAINT LANGUAGE

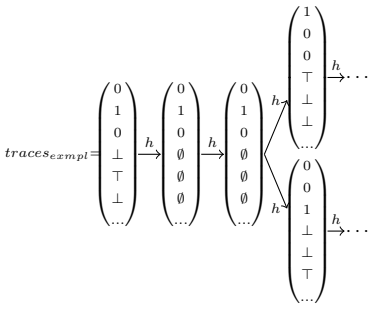
Due to the non determinism in the Petri net, the number of different scenarios exponentially increases. We must therefore be able to reduce the amount of branching.

Let us consider our traffic light example. The railway expert's knowledge must be used in order to distinguish interesting scenarios that need to be simulated. For example, the expert might know that exploring scenarios where two subsequent green lights is not relevant to the study. Furthermore, the expert might want to explore scenarios starting with a given sequence of traffic lights, i.e. three green lights.

From the previous paragraph, two main classes of constraints are distinguished: 1. Constraints that guide the co-simulation, called *prefixes*; 2. Constraints that reduce the number of scenarios explored, called *global constraints*; We want to express constraints using the number of tokens of the Petri net output places and the output values of the CBD. In other words, we create signals Sig from expressions based on the FMUs output variables U, boolean comparators $comp = \{=, \geq, \leq, >, <\}$ and real values. In the railway example, the set of signals Sig is defined as follows:

- $g \equiv (g_v = 1), g \in Sig$;
- $r \equiv (r_v = 1), r \in Sig$;
- $y \equiv (y_v = 1), y \in Sig$;

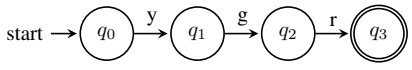
As mentioned before, one can see the composition co-simulation as a set of traces. Assuming a variables ordering $\langle r_v, y_v, g_v, r, y, g \rangle$ for the traces where r_v, y_v, g_v are output variables of the Petri net and r, y, g are signals created from Petri net output variables, one can consider the following set of traces:



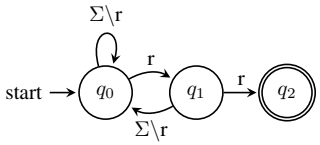
Note that we show a reduced number of steps between two consecutive Petri net steps. A few remarks can be made on the example set of traces: 1. The time semantic adaptation of the Petri nets is equal to $3 \times h$. In fact, the first $doStep$ of the Petri net is performed after a time of $3 \times h$. Therefore, $\emptyset$ is used to symbolize the fact that the created signals aren't forwarded to the CBD; 2. After a time of $3 \times h$, a branching is produced in the algorithm. In fact, in one case, the traffic light went from yellow to red, whereas it went from yellow to green in another scenario; 3. One can see that the semantic adaptation of the output variables are forwarded only at the time in which the $doStep$ is called for the Petri net.

In our running example, only constraints on the signals resulting from expressions on Petri net output variables were defined. Nevertheless, extending constraints using signals defined on CBD output variables can be made effortlessly.

Because the sequence of signals is represented as traces, one can define the latter constraint language using automata. For example, the co-simulation starting with the sequence of signals yellow, green, red can be defined by a *prefix* as follow:



As for *global constraints*, one can define two constraints, hence avoiding scenarios in which two red are encountered in a row:



The figure 3b depicts results of an execution with two trivial constraints applied. To show the effectiveness of the constraints language, we performed a small benchmark between five distinct executions without and with constraints. For this two sets of executions, we used the following simulation parameters:

- Stop time = 250 seconds;
- Step size = 0.001 seconds;
- Train segment length: 500 meters.

The mean execution time on a Intel Core i5 processor at 1.7 GHz with 8 GB of RAM running Windows 8 is as follows:

Without constraints	Std. Deviation	With constraints	Std. Deviation	Speedup
283.16 s	34.79	42.9 s	1.785	6.6

Note that we removed the best and worst execution time to average the values. As for the number of threads, we clearly see the branching reduction:

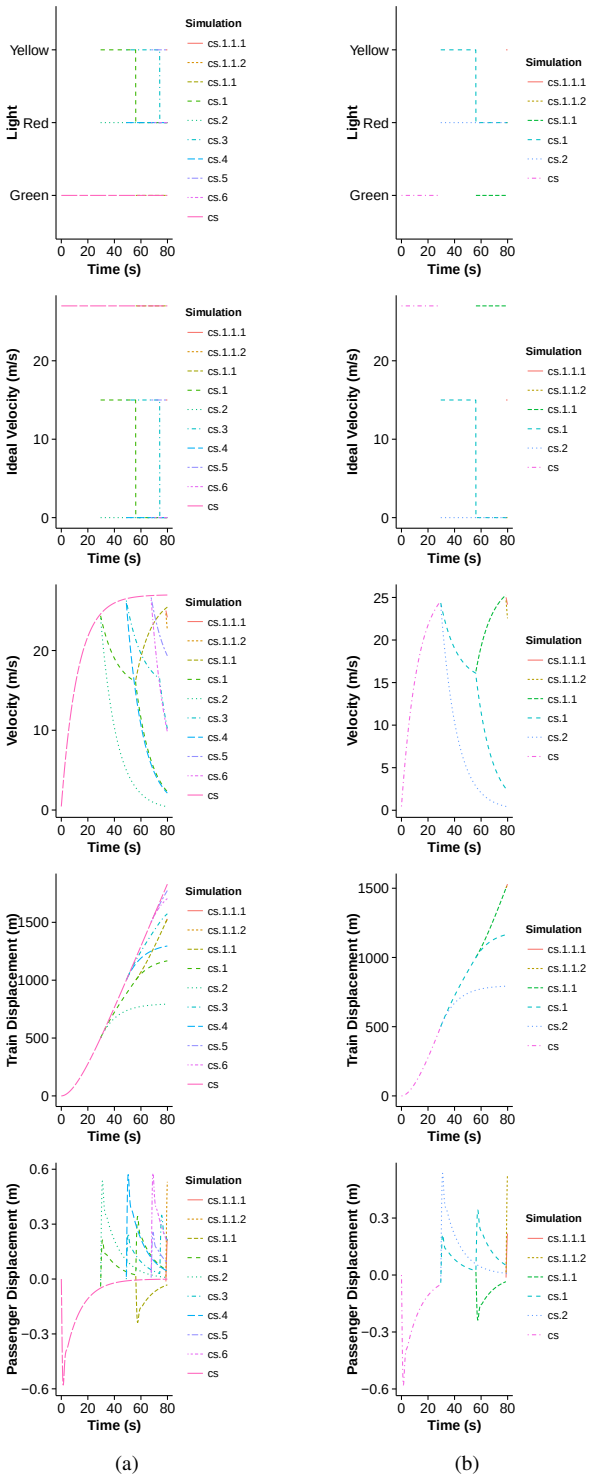
Without constraints	With constraints
2322	285

7. DISCUSSIONS

During the construction of the Petri net FMU, great care was taken that the resulting unit is compliant to the FMI standard. This means that integrating this unit in another co-simulation scenario with a generic master algorithm is possible. The coupling in that case, will yield a deterministic simulation of a single trace throughout the whole state space of the model. We refer to [7] for more information to deterministically couple multiple simulators.

However when the Petri net is used to generate multiple scenarios of the behaviour of the coupled system, a special master is required to create the different threads, possibly distributed over different physical computers. The creation of this master is cumbersome and error-prone. The needed semantic adaptation for data, control and time exacerbates this complexity [6]. It is therefore preferable to create a (domain-specific) language to configure and automatically generate the master and semantic adaptation between the different models, similar to [21, 11]. In our running example, there is only a single Petri net being co-simulated. The composition proposed in this paper can be extended to include more than one non-deterministic FMU and/or deterministic FMU. The cascading technique for synchronization has to be extended to work with multiple units. The generalization of the approach together with the creation of this master algorithm from a model in a (domain-specific) language is considered future work.

The presented approach uses Petri nets to express non-determinism and concurrency in the environment of a system. Nevertheless, one could imagine using a set of Petri net



(a) (b)

Figure 3: Execution scenarios without and with constraints applied. Each trace is named according to how it branched out.

models composed together to form a network of discrete systems for the controller and CBDs or differential equations to represent the physical environment surrounding this network. For example, a heater system divided in two subsystems: one subsystem that probes and regulates the temperature of the water and another that is in charge of ensuring that there is no unusual behaviour in the regulation of the temperature. The same type of master algorithms and synchronization algorithms are required to orchestrate the co-simulation.

In our approach, we guide the simulation towards interesting traces, and subsequent properties, using the constraint language. We constrain the state space of the system primarily for performance reasons because of the state space explosion. By reasoning more explicitly about the time-invariance (or shift-invariance) of the system, we can constrain the state space even further. A time-invariant system is a system where the output does not explicitly depend on time. If we know that our system is time-invariant, the initial state of the branch can be compared to the initial states at each of the branches. If the initial value problem is the same, we can safely assume that the branch is already simulated. While this seems easy in theory, in practice it is more intricate. Numerical approximation allows us to simulate continuous-time systems on computers. In the case of differential equations, the simulation is usually done using time discretization. This type of numerical algorithms give rise to local and global truncation errors, and precision errors which accumulate [8]. The amount of tolerated approximation must also be included to actually make a useful comparison between the different states. We consider this future work.

Finally, an FMU could take advantage of the underlying computational infrastructure to improve the computation time. A specific example could be the use of SIMD, Single Instruction - Multiple Data architectures to solve the sorted equations of models of the physics when the causality does not change. The FMU thus allows parallelism within the FMU.

8. CONCLUSION

Petri nets are an appropriate formalism to model the environment of cyber-physical systems because of their inherent non-determinism and concurrency. However, they need to be coupled with deterministic models that represent the plant and control of the system. In this paper we showed the meaning of coupling this non-deterministic model with a deterministic model using co-simulation. Furthermore, we operationalized the co-simulation using the FMI standard. We have shown that we are able to efficiently reduce the number of scenarios explored through simple traces matching constraints. Additionally, the constraint language we have implemented is based on well known and easy to use concepts. Finally, we demonstrated the approach on an example from the railway industry. By leveraging the FMI co-simulation standard to allow for the system verification through simulation, we hope to lower the entry barrier to the usage of these techniques.

ADDITIONAL AUTHORS

1. Didier Buchs, University of Geneva, 7, route de Drize, 1227 Carouge, Switzerland, Didier.Buchs@unige.ch.

REFERENCES

1. Alur, R. Formal verification of hybrid systems. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on* (2011), 273–278.
2. Alur, R., Henzinger, T., Lafferriere, G., Pappas, G. J., et al. Discrete abstractions of hybrid systems. *Proceedings of the IEEE 88*, 7 (2000), 971–984.
3. Antsaklis, P. J., Kohn, W., Nerode, A., and Sastry, S., Eds. *Hybrid Systems IV*, vol. 1273 of *Lecture Notes in Computer Science*, Springer (1997).
4. Bertsch, C., Ahle, E., and Schulmeister, U. The Functional Mockup Interface-seen from an industrial perspective. In *10th International Modelica Conference* (2014).
5. Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *9th International Modelica Conference* (2012).
6. Boulanger, F., Dogui, A., Hardebolle, C., Jacquet, C., Marcadet, D., and Prodan, I. Semantic adaptation using ccsl clock constraints. In *Models in Software Engineering*, J. Kienzle, Ed., vol. 7167 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, 104–118.
7. Broman, D., Brooks, C., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M. Determinate Composition of FMUs for Co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, IEEE Press (Piscataway, NJ, USA, 2013), 2:1—-2:12.
8. Cellier, F. E., and Kofman, E. *Continuous system simulation*. Springer Science & Business Media, 2006.
9. Clarke, E., Donzé, A., and Legay, A. On simulation-based probabilistic model checking ofmixed-analog circuits. *Formal Methods in System Design 36*, 2 (2010), 97–113.
10. Clarke, E. M., Fehnker, A., Han, Z., Krogh, B. H., Ouaknine, J., Stursberg, O., and Theobald, M. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci. 14*, 4 (2003), 583–604.
11. Denil, J., Meyers, B., Pussig, B., De Meulenaere, P., and Vangheluwe, H. Explicit semantic adaptation of hybrid formalisms for fmi co-simulation. In *Proceedings of the Symposium on Theory of Modeling and Simulation TMS-DEVS* (2015).
12. Gheorghe, L., Bouchhima, F., Nicolescu, G., and Boucheneb, H. Formal definitions of simulation interfaces in a continuous/discrete co-simulation tool. In *Rapid System Prototyping, 2006. Seventeenth IEEE International Workshop on*, IEEE (2006), 186–192.
13. Lucio, L. *SATEL - A Test Intention Language for Object-Oriented Specifications of Reactive Systems*. Phd thesis, UNIVERSITE DE GENEVE, 2008.
14. Mancini, T., Mari, F., Massini, A., Melatti, I., Merli, F., and Tronci, E. System Level Formal Verification via Model Checking Driven Simulation. In *Computer Aided Verification SE - 21*, N. Sharygina and H. Veith, Eds., vol. 8044 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, 296–312.
15. Mancini, T., Mari, F., Massini, A., Melatti, I., and Tronci, E. Anytime System Level Verification via Random Exhaustive Hardware in the Loop Simulation. In *Digital System Design (DSD), 2014 17th Euromicro Conference on* (2014), 236–245.
16. Péraire, C. *Formal testing of object-oriented software*. PhD thesis, EPFL, Lausanne, 1998.
17. Plaku, E., Kavraki, L. E., and Vardi, M. Y. Falsification of LTL safety properties in hybrid systems. *STTT 15*, 4 (2013), 305–320.
18. Siegl, S., and Russer, M. Constructive modelling of parallelized environmental models for structured testing ofautomated driving systems. In *Cyber Physical Systems. Design, Modeling, and Evaluation*, M. R. Mousavi and C. Berger, Eds., vol. 9361 of *Lecture Notes in Computer Science*. Springer International Publishing, 2015, 25–39.
19. Tabuada, P., Pappas, G. J., and Lima, P. U. Compositional abstractions of hybrid control systems. *Discrete Event Dynamic Systems 14*, 2 (2004), 203–238.
20. Tripakis, S., and Broman, D. Bridging the semantic gap between heterogeneous modeling formalisms and fmi. Tech. rep., DTIC Document, 2014.
21. Van Acker, B., Denil, J., De Meulenaere, P., and Vangheluwe, H. Generation of an optimised master algorithm for fmi co-simulation. In *Proceedings of the Symposium on Theory of Modeling and Simulation TMS-DEVS* (2015).
22. Van der Auweraer, H., Anthonis, J., De Bruyne, S., and Leuridan, J. Virtual engineering at work: the challenges for designing mechatronic products. *Engineering with Computers 29*, 3 (2013), 389–408.
23. Xu, D. A Tool for Automated Test Code Generation from High-Level Petri Nets. In *Applications and Theory of Petri Nets SE - 17*, L. Kristensen and L. Petrucci, Eds., vol. 6709 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, 308–317.
24. Younes, H. L., Kwiatkowska, M., Norman, G., and Parker, D. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer 8*, 3 (2006), 216–228.