Model Checking FMI Co-simulation Using Timed Automata

Kaiqiang Jiang, Chunlin Guan, Jiahui Wang, Dehui Du*
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, 200062, China
Email: dhdu@sei.ecnu.edu.cn

***Abstract*—The growing complexity of Cyber-physical Systems (CPSs) increasingly challenges existing methods and techniques. CPSs are often treated modularly to tackle both complexity and heterogeneity. A promising approach for verifying CPSs is to use Functional Mock-up Interface (FMI) co-simulation techniques. However, the master algorithm for co-simulation may be livelock or deadlock. The architectural modelling of CPSs may also introduce an algebraic loop which is a feedback loop resulting in instantaneous cyclic dependencies. To solve these problems, we propose a novel approach for model checking several properties of FMI co-simulation such as deadlock, liveness, reachability. We model the architecture of CPSs with SysML block diagrams, which captures the dependence of Functional Mock-up units (FMUs) and the orchestration of the master algorithm. To analyse the correctness of the system, we encode FMUs components and three various master algorithms with timed automata separately. With the help of encoding, we verify the master algorithms for co-simulation and the absence of algebraic loops in the architecture with UPPAAL. To illustrate the feasibility of our approach, the case study water tank is presented. The experiments show that our approach facilitates model checking FMI co-simulation.**

***Keywords*—Co-simulation, Master algorithm, Functional Mock-up Interface, Timed automata, Model checking.**

I. Introduction

Cyber-physical systems (CPSs) are integration of computation with physical processes whose behavior is defined by both computational and physical parts of the system [1]. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The heterogeneity is one of the main characteristics of CPSs. The components of CPSs are of various types, requiring interfacing and interoperability across multiple platforms and different models of computation. Verification of CPSs as a whole requires the use of heterogeneous simulation environments. One emerging industry standard is the Functional Mock-up Interface (FMI) [2] [9]. It is a standard designed to support simulation of complex systems comprising heterogeneous components, by coupling the different models with their own solver in a co-simulation environment.

The FMI standard was first developed in the MODELISAR project started in 2008 and supported by a large number of software companies and research centers. FMI offers the means for model based development of systems and is particularly appropriate way to develop complex CPSs. The FMI standard supports both co-simulation and model exchange. In this paper, we focus on the co-simulation in FMI 2.0. Compared with the FMI 1.0, there are two important additions: *fmiGetFMUstate* and *fmiSetFMUstate*, which allow the master to copy and restore the complete state of an FMU slave. These two functions provide an ordinary mechanism for rollback.

The soul of FMI-based co-simulation is master algorithm (MA) [3], which provides the orchestration of the entire co-simulation. However, the master algorithm is not a part of the FMI standard. This implies that the user or tool vendor needs to develop a sophisticated orchestration algorithm for the problem at hand. The correctness of the master algorithm also needs to be analysed. In this paper, we verify three versions of master algorithms [9]: fixed step algorithm, rollback algorithm, predictable step size algorithm. Rollback and predictable step size algorithms are based on the extension of FMI 2.0, which supports the rollback and predict function. PG Larsen et al. [24] formally analysed the fixed step and rollback algorithms with the FDR3 refinement checker. Timed automaton (TA) [6] is a finite automaton extended with a finite set of real-valued clocks. During a run of a timed automaton, clock values increase all with the same speed. We model three versions of master algorithms with timed automata, and verify the algorithms with UPPAAL. Once the correctness of master algorithm is ensured, in the remainder of paper we model the co-simulation of CPSs to model check several properties of co-simulation such as deadlock, liveness and reachability.

To achieve the goal, we present a novel approach to model and verify the properties of co-simulation with timed automata [4]. The schematic view of our approach is shown in Fig.1. At the design phase, we construct the architecture of CPSs with SysML block diagrams [5]. Each block represents a component and the communication between components is modeled with SysML connector. To simulate the whole system with co-simulation techniques, the block can be modeled with a Functional Mock-up (FMU) and the connector can be modeled with a master algorithm. The MA orchestrate FMUs to accomplish the communication between FMUs. To verify the correctness of the architecture, we encode the FMU and model the master algorithm by timed automata, which facilitates the verification of livelock or deadlock with the model checker UPPAAL [6].

Contributions. Our contributions are as follows:

- We present a novel approach to model check several properties of the co-simulation based on timed automata. With the help of model checker, the property such as livelock, deadlock and reachability are veri-

*Corresponding Author

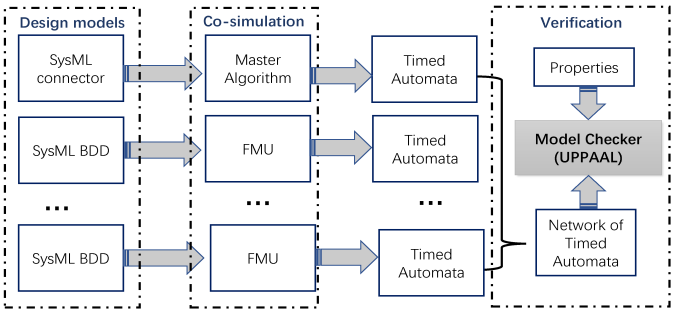


Fig. 1. A schematic view of our approach.

fied.

- We model and verify three various master algorithms to ensure the correctness of the co-simulation process. Besides, FMU is encoded into timed automata with encoding rules and the orchestration between FMUs is modeled with a network of timed automata. With the help of UPPAAL, we analyse the reachability, livelock and deadlock of the architecture.
- The prototype for model checking co-simulation of CPSs is developing, which is integrated in our Mondana platform [7](https://github.com/ECNU-MODANA/AL-Modana.git). We have developed the SysML modelling environment and the *co-simulator* for simulating CPSs [8].

The remainder of this paper is organized as follows. In Section II, we briefly review the technical background including FMI, FMU and timed automata, and then provide encoding FMU by timed automata. Section III models three versions of master algorithms with timed automata and verify properties such as the livelock and deadlock. Section IV presents the architecture modelling of water tank case study with SysML block diagrams, and then obtains the FMUs and FMUs connection of water tank system. We encode the FMUs with the timed automata, and model check FMI co-simulation of the water tank with UPPAAL. The experimental results show that our approach is feasible. Finally we position our work with respect to related work before concluding and discussing possible future extensions.

II. Encoding FMUs by timed automata

We give the syntax and semantics of FMU and timed automata. In order to verify the execution of FMUs. We propose to encode FMUs by timed automata. In section IV, we verify the network of timed automata with UPPAAL.

A. FMU

FMU is the model component which implements the methods defined in the FMI API [10]. Here, we present the syntax and semantics of FMU. The aim is to encode FMU into timed automata based on their semantics.

Definition 1. FMU syntax We recall the definition of FMU. An FMU is a tuple $F = (S, U, Y, D, s_0, set, get, doStep)$, where:

- S denotes the set of states of F.
- U denotes the set of input port variables of F. Note that an element $u \in U$ is a variable, not a value, which ranges over a set of values $\mathbb{V}$.
- Y denotes the set of output port variables of F. Each $y \in Y$ ranges over the same set of values $\mathbb{V}$.
- $D \subseteq U \times Y$ denotes a set of input-output dependencies. $(u, y) \in D$ means that the output y is directly dependent on the value of u. The I/O dependency information is used to ensure that a network of FMUs does not contain cyclic dependencies, and also to identify the order in which all variables are computed during a step.
- $s_0 \in S$ denotes the initial state of F.
- $set : S \times U \times \mathbb{V} \to S$ denotes the function that sets the value of an input variable. Given current state $s \in S$, input variable $u \in U$, and value $v \in \mathbb{V}$, it returns the new state obtained by setting u to v.
- $get : S \times Y \to \mathbb{V}$ denotes the function that returns the value of an output variable. Given state $s \in S$ and output variable $y \in Y$, $get(s, y)$ returns the value of y in s.
- $doStep : S \times \mathbb{R}_{\geqslant 0} \to S \times \mathbb{R}_{\geqslant 0}$ denotes the function that implements one simulation step. Given current state s, and a non-negative real value $h \in \mathbb{R}_{\geqslant 0}$, $doStep(s, h)$ returns a pair (s', h') such that:
  When $h' = h$, it indicates that F accepts the time step h and reaches the new state s';
  When $0 \leqslant h' < h$, this means that F rejects the time step h, but making partial progress up to h', and reach the new location s'.

Definition 2. FMU semantics Given the FMU $F = (S, U, Y, D, s_0, set, get, doStep)$,

The behavior of F depends on the functions $doStep$, which is a function of a timed input sequence (TIS). A TIS is an infinite sequence $v_0 h_1 v_1 h_2 v_2 h_3 ...$ of alternating input assignments v_i, and time delays h_i. An input assignment is the value of function $v : U \to \mathbb{V}$. That is, v assigns a value to every input variable in U. A TIS denotes a running of FMU, which is an infinite sequence of quadruples (t, s, v, v'), where $t \in \mathbb{R}_{\geqslant 0}$ is a time instant, $s \in S$ is a state of F, v is an input assignment, and $v' : Y \to \mathbb{V}$ is an output assignment

TIS := $(t_0, s_0, v_0, v'_0)(t_1, s_1, v_1, v'_1)(t_2, s_2, v_2, v'_2)...$ is defined as follows:

- $t_0 = 0$ and s_0 is the initial state of F.
- For each $i \geqslant 1$, $t_i = t_0 + \sum_{k=1}^{i} h_k$
- Given the current state s_i, the function set is used to set all input variables to the values specified by v. Then F reaches a new state s'_i. The function get is used to get the values of all output variables v'_i.
- We assume that $doStep(s_i, h_{i+1}) = (s_{i+1}, h_{i+1})$ based on the assumption that every h_i is accepted by F, F will reach the next state s_{i+1}.

B. Timed Automata

Timed automata (TA) [6] is a theory to model the behavior of real-time systems. Its definition provides a powerful way to annotate state-transition graphs with many real-valued clocks. In this section, we introduce the syntax and semantics of timed automata. In section IV, we will encode the FMUs of our case study with the network of TA, so that we can use the model checker UPPAAL to analyse models.

Definition 3. Timed automata syntax A timed automaton is a tuple $A = (L, X, l_0, E_i, E_o, I)$, where:

- L is a finite set of locations;
- X is a finite set of clocks;
- $l_0 \in L$ is the initial state;
- The set of guards $G(x)$ is defined by the grammar $g := x \bowtie c \mid g \wedge g$, where $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leqslant, \geqslant, >\}$. $E \subseteq L \times G(X) \times 2^X \times L$ is a set of edges labelled by guards and a set of clocks to be reset;
- E_i is a set of input events.
- E_o is a set of output events.
- $I : L \to G(X)$ assigns invariants to clocks.

A clock valuation is a function $v : X \to \mathbb{R}_{\geqslant 0}$. If $\delta \in \mathbb{R}_{\geqslant 0}$, then $v + \delta$ denotes the valuation such that for each clock $x \in X$, $(v + \delta)(x) = v(x) + \delta$. If $Y \subseteq X$, then $v[Y := 0]$ denotes the valuation such that for each clock $x \in X$ Y, $v[Y := 0](x) = v(x)$ and for each clock $x \in Y$, $v[Y := 0](x) = 0$. The satisfaction relation $v \models g$ for $g \in G(x)$ is defined in the natural way.

Definition 4. Timed automata semantics The semantics of a timed automaton $A = (L, X, l_0, E, E_i, E_o, I)$ is defined by a transition system $L_A = (L, l_0, \to)$,

where $L = L \times \mathbb{R}^X_{\geqslant 0}$ is the set of locations, $l_0 = (l_0, v_0)$ is the initial location, $v_0(x) = 0$ for all $x \in X$, and $\to \subseteq L \times L$ is the set of transitions defined by :

- $(l, v) \xrightarrow{\in(\delta)} (l, v+\delta)$ if $\forall 0 \leqslant \delta' \leqslant \delta : (v+\delta') \models I(s)$;
- $(l, v) \to (l', v[Y := 0])$ if there exists $(l, g, Y, l') \in E$ such that $v \models g$ and $v[Y := 0] \models I(l')$.

The reachability problem for an automaton A and a location l is to decide whether there is a state (l, v) reachable from (l_0, v_0) in the transition system L_A. As usual, for verification purposes, we define a symbolic semantics for timed automata. For universality, the definition uses arbitrary sets of clock valuations.

Consider a location l such that for any $t \in X$, for fixed constant $x \in X$, clock valuation $x + t \in X$. A possible execution fragment starting from this location is

$(l, t) \xrightarrow{x_1} (l, t + x_1) \xrightarrow{x_2} (l, t + x_1 + x_2) \xrightarrow{x_3} (l, t + x_1 + x_2 + x_3) \xrightarrow{x_4} ...$

where $x_i > 0$ and the infinite sequence $x_1 + x_2 + ...$ converges toward x.

C. Encoding FMUs into timed automata

As we can see, there is a semantic gap between FMU and TA. The former focus on the execution sequence of FMU, which specifies the state change process with time passing. Essentially, the execution trace of TA is semantic equivalence to the execution sequence of FMU. Therefore, we can encode FMU into TA to analyse the behavior of FMU component without exploring its internal structure.

Given an FMU $F = (S, U, Y, D, s_0, set, get, doStep)$, we encode the FMU into a timed automaton $A = (L, X, l_0, E, E_i, E_o, I)$, the congruent relationship between them is as following:

- L is a set of finite states. Note that a location of A is the abstraction of a state in F.
- The initial location of TA l_0 which $x := 0 | x \in X$ is such that s is set to s_0 of F.
- Each input variable $u \in U$ ranges over $E_i \cup \{absent\}$.
- Each output variable $y \in Y$ ranges over $E_o \cup \{absent\}$.
- An input event in $e \in E_i$ is such that the function set of F sets the input variable u to a given value.
- An output event in $e \in E_o$ indicates that the function get of F gets the output variable y. The set of values in the E_i can be seen as Y of F.
- The communication between the network of TA is the same as the I/O dependencies information in FMU. $(u, y) \in D$ denotes that output y depend on input u. The output events also depend on the input events in TA.
- For any $e \in E$ of A, there is a transition $s \xrightarrow{e} s'$, which may be found after the function $doStep$ is executing. For instance, if there is a transition $l \xrightarrow{e} l'$ in A, at the same time $doStep(s, h)$ may be called which indicates that F accepts the time step h and reaches the new state s'. However, F maybe rejects the time step, if there is a rollback behavior happens, the transition in TA could be a edge $l' \xrightarrow{e} l$, which denotes that a location travels to the former location.

It is not easy to translate FMU to TA directly, we propose some encoding rules from FMU to TA. As we can see in the Fig.2, given a state s_i at t_1 in FMU, the operation $Dostep$ makes FMU reach a new state s_{i+1} at $t_1 + step$. This situation can be encoded into a transition in TA, in which a location L_i delays $step$ time and goes to a new location L'_i.

For the operation $Rollback$, given a state si at t_1 in FMU, the FMU will do a step1 to s_{i+1} at $t_1 + step1$, and then, the operation $rollback$ makes FMU reach the former state s_i. For this situation, it can be encoded as: location L_i delays step1 time and reach a new location L'_i after a transition, next returns to the former Location L_i.

For the operation $prediction$, given a state s_i, FMU can get max step size ($step$) for next step, and then reach a new state s_{i+1} at $t_1 + step$. For TA, it gets max step size in location L_i, then it delays $step$ time and reach a new location L'_i .

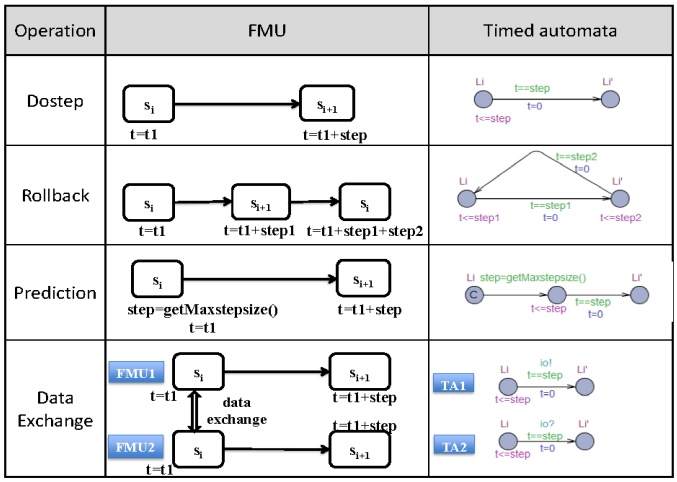


Fig. 2. Encoding rules from FMU to TA.

For data exchange between two FMUs in state s_i at t_1, they exchange data at t_1 and then do the same step to s_{i+1}. In TA, there will be a signal io to make the two FMUs do the same step from L_i to L_{i+1} after data exchange.

Although there are semantic gaps between FMUs and timed automata, we provide appropriate encoding rules to formalism FMU with timed automata. It lays the foundation for analyse FMI co-simulation with timed automata-based model checking.

III. Modelling and Analysis of Master Algorithm

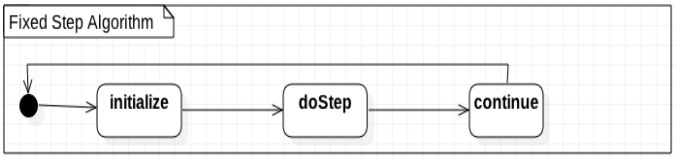
The master algorithm (MA) provides the orchestration of FMUs, which denotes the co-simulation of various FMUs. To ensure the correctness of co-simulation execution process, it is necessary to verify certain properties of the master algorithm. In this section, we utilize timed automata to model three versions of master algorithms and verify some expected properties of master algorithm such as deadlock, liveness and reachability with UPPAAL.

A. I/O Dependency Information

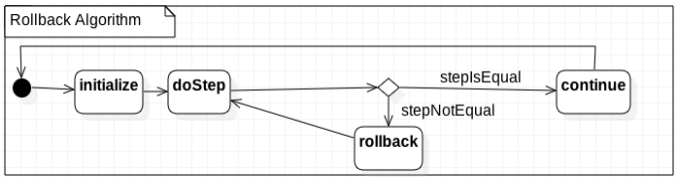
When it comes to co-simulation, I/O dependency information [9] is inevitably required to be well considered. The master algorithm calls function *Set* to provide input value to an FMU and function *Get* to retrieve an output value. So it is essential to know which outputs of an FMU depend immediately on which inputs. In the design of a MA, the direct dependency information can be used to call the function *Set* and *Get* in a well-defined order. In FMI 2.0, this information can be provided using the element *ModelStructure* [11]. However, sometime there may be an algebraic loop in the dependency information, which may not converge. Since we are interested in non-diverging and deterministic composition of FMUs, we need to distinguish these two cases.

B. Master Algorithm

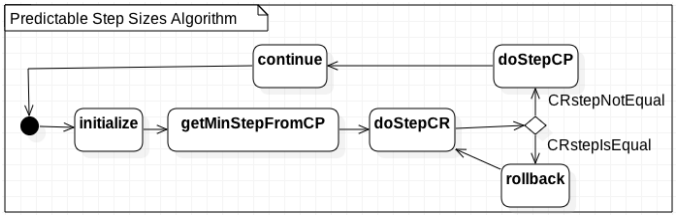
The master algorithm is to orchestrate the execution of different subsystems. Each subsystems serves as an FMU block whose simulation is triggered by a particular MA. FMUs can be seen as black boxes. FMU can be simulated independently until it needs to exchange data or implement synchronization. There are three versions of master algorithm, which is shown in Fig.3.



(a) Fix step size algorithm



(b) Rollback algorithm



(c) Predictable step sizes algorithm

Fig. 3. Activity diagrams for three versions of master algorithms.

1) Fixed Step Algorithm: For fixed step algorithm, all FMUs have the same step size. When master algorithm calls *doStep* with the step size *h*, it will advance from a communication point *t* to the next communication point *t+h*. During the simulation step, an FMU with it's own solver will simulate independently according to its input value and generate a running result as output value. MA will wait until all FMUs finish their simulation step and then get their output values to exchange data for preparing next simulation step. The activity diagram of fixed step algorithm is illustrated in Fig.3(a). There are mainly three activities in the control flow: *initialize*, *doStep* and *continue*. In the fixed step algorithm [9], the process can maintain correctness when all FMUs are reliable. When some error happens during a simulation step, the process will be affected after the wrong simulation step. To overcome the shortcoming of the fixed step algorithm, it needs rollback mechanism.

2) Rollback Algorithm: There are some important features proposed in the FMI 2.0. It supports to save the FMU state if necessary and the saved state can be restored. For example, MA calls *doStep* on FMU_1 and FMU_2 while FMU_1 can accept the request or FMU_2 can reject it. If we save the state of FMU_1 and FMU_2 at the communicating point t, we can restore the scene after FMU_2 rejects *doStep*. The activity diagram of rollback algorithm is clearly shown in Fig.3(b). Compared with the fixed step algorithm, all FMUs are required to support *rollback* mechanism, that is, all FMUs need to return to the previous state if the simulation step sizes of all FMUs are not equal.

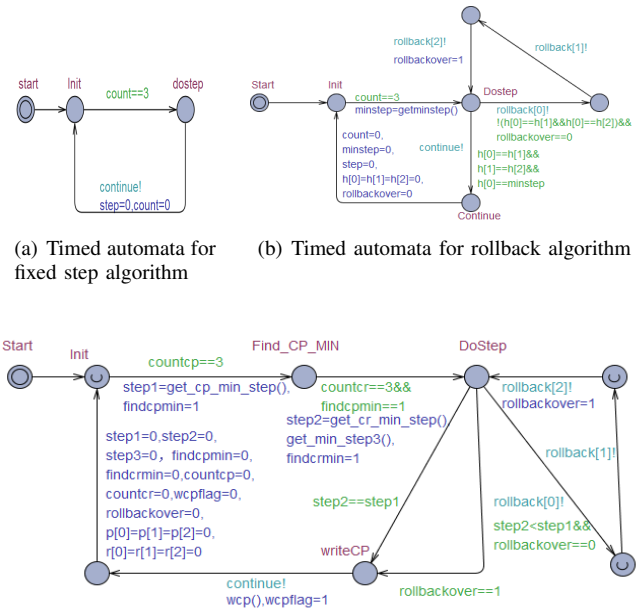
3) Predictable Step Size Algorithm: To improve the efficiency of MA, it is important to predict step size. So predictable step size algorithm is proposed. The function *GetMaxStepSize* was introduced to optimize the performance of rollback algorithm. This function returns the maximum step size and state flag of a predictable FMU. Maximum step is the largest step that a predictable FMU can perform. State flag includes *ok*, *discard* and *error*. *OK* denotes the predictable FMU can accept the simulation step size. *Discard* denotes the predictable FMU only implement partial step during simulation. *Error* denotes the predictable FMU can't continue the simulation because of its unacceptable state or unreasonable input value. Also, when *discard* and *error* happen, the FMU needs to rollback to the previous saved state. Whether an FMU is a predictable FMU or not should be indicated in FMU's *xml* file. Moreover, if an FMU supports rollback and predictable step size at the same time, the predictable step size algorithm only uses predictable ability to get the maximum step of a predictable FMU. On the other hand, a predictable FMU can accept any step size less than or equal to the maximum step returned by *GetMaxStepSize*.

First, the master algorithm chooses the maximum step size of all predictable FMUs and find the smallest communication step size *h* that all predictable step size can be accepted. Then, we save the states of all FMUs. MA calls *doStep(h)* on FMUs supporting rollback. The function *doStep()* will return the real performed step size. If all performed step sizes are equal to *h*, MA will call *doStep(h)* on FMUs. Otherwise, MA will find the smallest performed step h_{min}, then all FMUs will restore the state saved before the co-simulation. Finally, MA will invoke $doStep(h_{min})$ on all FMUs. The control flow of predictable step size algorithm is shown in Fig.3(c). For example, *getMinStepFromCP* is an activity that MA will call *GetMaxStepSize* on all predictable FMUs to find their maximum simulation step size and then return the smallest one of them.

C. Modelling and Analysis of MA

UPPAAL [6] is a toolset for verification of real-time systems represented by (a network of) timed automata which is extended with integer variables, structured data types, and channel synchronization. To verify the correctness of three versions of master algorithms, we model the master algorithms using timed automata in UPPAAL. The Fig.4 shows the timed automata model of three master algorithms, respectively. Fixed step algorithm has *Init*, *doStep* states and synchronize with FMU by channel *continue*. Rollback algorithm has *Init*, *DoStep*, and *Continue* states. If all FMUs don't have the same step size, rollback algorithm will communicate with FMUs by *rollback* signal, otherwise. It will send *continue* signal and move to *Continue* state. Predictable step size algorithm has *Init*, $find\_CP\_MIN$, *DoStep*, *writeCP* states. It obtains the minimal step size (i.e., *step2*) of FMUs supporting *GetMaxStepSize* function and the maximal step size (i.e., *step1*) of FMUs supporting rollback. If *step1* is greater than *step2*, FMUs receive *rollback* signal and return to *DoStep* state. Otherwise, FMUs receive *continue* signal and do the next step.

We verify the properties of master algorithms including reachability, liveness and deadlock. Experimental results are shown in Table I, where:



(a) Timed automata for fixed step algorithm

(b) Timed automata for rollback algorithm

(c) Timed automata for predictable step size algorithm

Fig. 4. Timed automata for three versions of master algorithms.

TABLE I. EXPERIMENTAL RESULTS

MA	Property	Result
Fixed Step	$A[]\ not\ deadlock$	True
	$master.Init \rightarrow master.dostep$	True
	$E\langle\rangle\ master.dostep$	True
Rollback	$A[]\ not\ deadlock$	True
	$master.Init \rightarrow master.Continue$	True
	$E\langle\rangle\ master.Continue$	True
Predictable	$A[]\ not\ deadlock$	True
	$master.Init \rightarrow master.writeCP$	True
	$E\langle\rangle\ master.writeCP$	True

- $E\langle\rangle\ master.dostep$, $E\langle\rangle\ master.Continue$ and $E\langle\rangle\ master.writeCP$ are reachability properties checking whether the model can reach these states;
- $master.Init \rightarrow master.dostep$, $master.Init \rightarrow master.Continue$ and $master.Init \rightarrow master.Continue$ are liveness property. If the master algorithm arrives at the former state, it eventually reaches the latter state;
- $A[]\ not\ deadlock$ is safety property, which means whether the model will be deadlock.

In Table I, we can find that the properties such as deadlock, liveness and reachability are satisfied, which proves the correctness of three master algorithms. For example, $A[]\ not\ deadlock$ is satisfied, which means there is no deadlock in the execution of the master algorithm. $master.Init \rightarrow master.doStep$ is satisfied, which means if the model reach the former state $Init$, it will eventually reach the state $doStep$. $E\langle\rangle\ master.doStep$ is satisfied, which means there exists a reachable state $doStep$.

IV. CASE STUDY

To illustrate our approach, we take an example (water tank) inspired by [12]. According to the I/O dependency information

between FMUs, the architectural model for water tank is constructed using SysML. The aim of using SysML is to design the architecture of the system with a more high-level modeling language. It helps to show the components and their connection.

The water tank system is our running example. A source of water flows into the water tank whose water flows into the drain. The source is controlled by a valve; when the valve is open, the water flows into the water tank. The valve, managed by a software controller, is opened or closed stochastically or depending on the water level. There are three various water tank systems depending on various connections between controller, valve and tank.

A. Architecture Modelling in SysML

SysML is a general purpose domain-specific language (DSL) [13] for model-based systems engineering (MBSE) [14], which is originated as an initiative of the International Council on Systems Engineering (INCOSE) [15] in January 2001. SysML is implemented as a UML profile. The *Block Definition Diagram*(BDD) describes the system blocks and their features (structural and behavioural). The*Connection Diagram* (CD) describes the internal structure of blocks. The ports of blocks are connected by the connector. The I/O dependence of blocks describes the communication between blocks. SysML block diagrams are usually used to describe the architecture of systems.

Figure 5 shows the block definition diagram for the water tank system. The system consists of three blocks, i.e., *Valve*, *Tank* and *Controller*, in which *Valve* and *Tank* are physical components. *Controller* is the cyber component. Each component has its own input and output. For instance, the input interface of *Valve* is named as *vin*, which is used to input the *Open-Closed* signal. Figure 6 shows the connection diagram

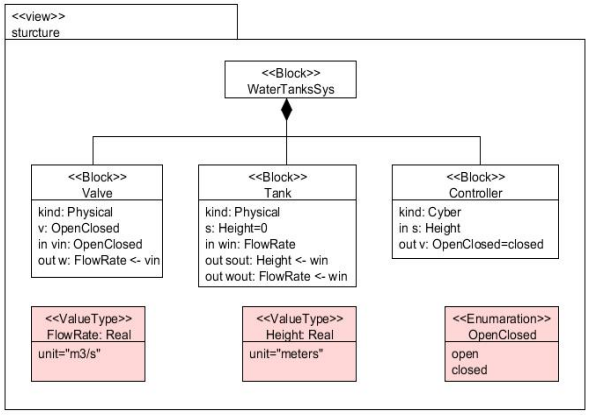
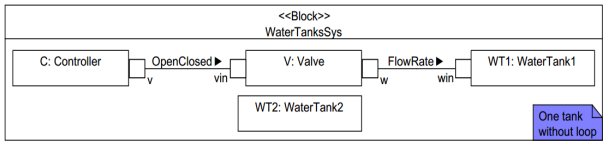
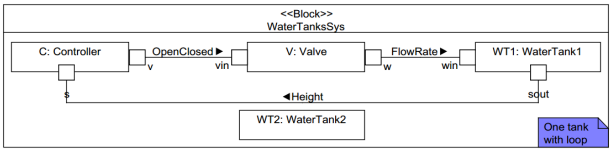


Fig. 5. SysML BDD for water tank system.

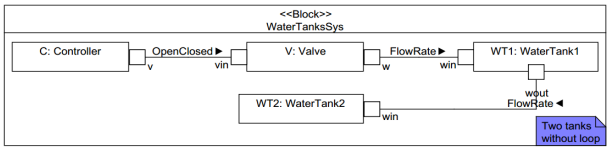
for the system. There are three cases for connections. The first case is that the system has one valve, one controller and one tank. The controller sends stochastic signals to control the valve on/off leading to various rate of water flow. The second case is that the signal from the controller is affected by the water level of the tank. The last case is on the basis of the first case and adds another tank2 which is affected by the flow rate of the tank1.



(a) Connection case 1



(b) Connection case 2



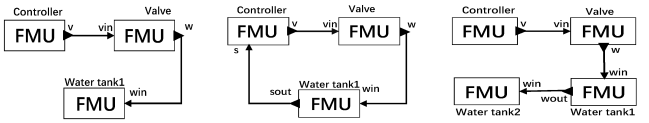
(c) Connection case 3

Fig. 6. SysML CD for water tank system.

We model the architecture with SysML which is a high-level modeling language. The SysML BDD shows the blocks of system and SysML CD shows the connection between blocks. In next section, we abstract each block as a FMU, and obtain the connection between FMUs based on the SysML CD.

B. The FMUs Connection of Water Tank System

Figure 7 is the FMUs and FMUs connection of water tank system. There are three connection cases between the FMUs according to the SysML CD in the previous section. The first case contains three FMU components (*Controller*, *Valve* and *Tank1*) and two channels(v_vin, w_win) as shown in Fig.7(a). The controller and valve are connected with channel v_vin. The valve and tank1 are connected with channel w_win. The second case is shown in Fig.7(b), there could be a channel $sout\_s$ between tank1 and controller, which means the water level of tank1 affects the control strategy of the controller. Figure 7(c) shows the third case, there could be another (tank2), the tank1 and tank2 are connected by the channel w_out. How can we assure the correctness of the architecture



(a) Connection case 1 (b) Connection case 2 (c) Connection case 3

Fig. 7. FMUs connection of water tank system.

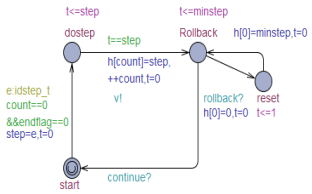
models? We attempt to verify it with model checking based on timed automata. More details on verification process can be found in the next section.

C. Verification and Analysis with UPPAAL

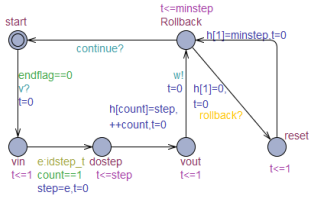
This section performs a formal analysis of the architectures of water tank. First off, we encode FMUs of water tank

system using timed automata and model the master algorithm with timed automata. Therefore, the time automata of FMUs and master algorithm compose a network of timed automata. Next, the verification and analysis checks whether the model is accurate and satisfies certain desired properties with UPPAAL.

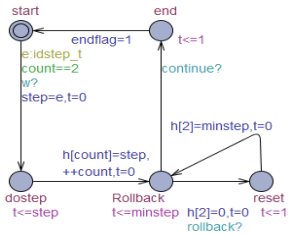
The execution of FMU and co-simulation is time-related, we have proposed the method to encode FMU with timed automata in Section II. In this subsection, we abstract the execution of FMU of water tank system, and encode it with the locations and transitions in timed automata. Besides, we also model the master algorithm as a timed automata to coordinate the execution between several FMUs. The timed automata template for FMUs and master algorithm are shown in Fig.8. In Section III, we verified the correctness of three master algorithms. Here, we choose rollback algorithm as the master algorithm to coordinate the FMUs. The other two master algorithms can be analyzed with the similar way.



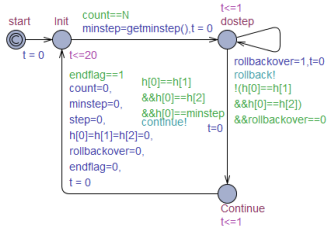
(a) Timed automata template for FMU controller



(b) Timed automata for FMU valve

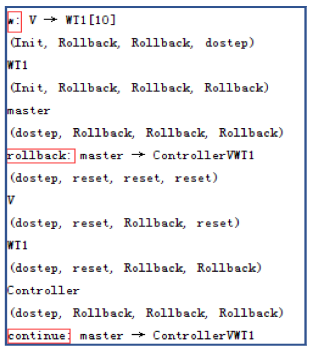


(c) Timed automata for FMU tank1

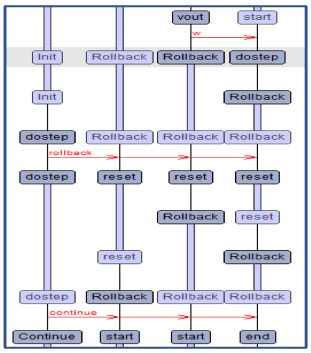


(d) Timed automata for master algorithm

Fig. 8. Network of TA for connection case 1: *TA _controller* || *TA _valve* || *TA _tank1* || *TA _ma*.



(a) Execution trace



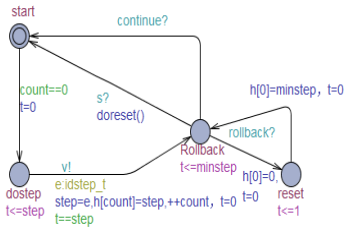
(b) Execution sequence diagram

Fig. 9. The execution fragment of the co-simulation in UPPAAL.

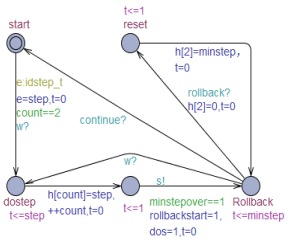
Figure 8(a), 8(b), 8(c) are the templates for controller, valve and tank1 respectively, they model FMUs of water tank system which support rollback function. These FMUs contain four main states, e.g., *start*, *dostep*, *Rollback* and *reset*. Figure 8(a) shows the template for controller which executes random step size. It synchronizes with valve by signal v and transfers to *Rollback* state, and then waits for a signal from the master algorithm. Until the controller receives the *continue* signal, it does data exchange with other FMUs, and returns to *start* state. Otherwise it receives *rollback* signal, once it obtains the minimize step size of all FMUs, it transfers to *Rollback* state. The states and transitions of valve and tank1 template are similar with the template of controller. Figure 8(d) shows the template for the master algorithm. Firstly, the master algorithm initializes the parameters, and then it gets minimize step size of FMUs until all FMUs visit *dostep*. Next, the master algorithm decides which signal should be sent according to the guard. If the step sizes of all FMUs are equal, the master algorithm will send *continue* signal, otherwise, send *rollback* signal.

Figure 9 is the execution fragment of the co-simulation in UPPAAL, we can find that valve sends a w signal to perform data exchange with tank1. After that, tank1 moves to *dostep* state. The master algorithm sends a *rollback* signal to all templates, which leads to all of them arrive at *reset* state. Finally, the master algorithm sends a *continue* signal to all FMUs. All templates return to *start* state, and then do next step. The execution process shows that our model performs correctly.

In order to compare the behavior of three connection cases of water tank system presented in Section 4.2, we also model the other two connection cases in UPPAAL. We add a channel s on the templates for controller and tank1 of connection case 1 to obtain the model of connection case 2 as shown in Fig.10. We add a template (tank2) and channel $w2$ to obtain the model of connection case 3 as shown in Fig.11. In next subsection, we verify some properties of various connection cases to analyse the correctness of the architecture.



(a) Timed automata for FMU controller



(b) Timed automata for FMU tank1

Fig. 10. Network of TA for connection case 2: *TA _controller* || *TA _valve* || *TA _tank1* || *TA _ma*.

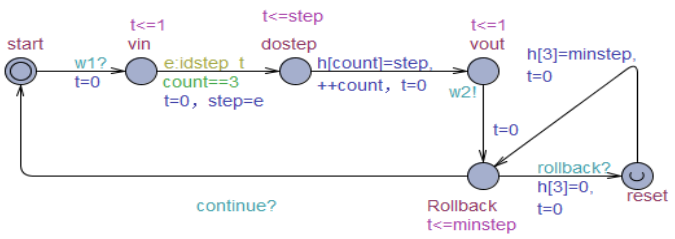


Fig. 11. Network of TA for connection case 3: *TA _controller* || *TA _valve* || *TA _tank1* || *TA _ma*|| *TA _tank2*.

TABLE II. EXPERIMENTAL RESULTS

Connection case	Property	Result
Case 1	$E\langle\rangle\ WT1.Rollback$	True
	$E\langle\rangle\ master.Continue$	True
	$master.start \rightarrow master.Continue$	True
	$A[]\ not\ deadlock$	True
Case 2	$E\langle\rangle\ WT1.Rollback$	True
	$E\langle\rangle\ master.Continue$	False
	$master.start \rightarrow master.Continue$	False
	$A[]\ not\ deadlock$	True
Case 3	$E\langle\rangle\ WT1.Rollback$	True
	$E\langle\rangle\ master.Continue$	True
	$master.start \rightarrow master.Continue$	True
	$A[]\ not\ deadlock$	True

UPPAAL uses a simplified version of TCTL [16] to express the requirement specification. We verify the following properties of each connection case:

- $E\langle\rangle\ WT1.Rollback$ and $E\langle\rangle\ master.Continue$ are reachability properties checking whether FMU tank1 can reach $Rollback$ state and whether the master algorithm can reach $Continue$ state respectively.
- $master.start \rightarrow master.Continue$ are liveness property. If the master algorithm arrive at $start$ state, it eventually reaches $Continue$ state.
- $A[]\ not\ deadlock$ is safety property checking whether the model will be deadlock.

The verification results are shown in Table II. We can find that all properties of connection case 1 and 3 are satisfied. It proves that our master algorithm is correct and the composition of FMUs is determinate. However, the liveness and reachability properties of connection case 2 are not satisfied. We find that there is a algebraic loop which may be introduced with the I/O dependency in this connection case. The experimental results show that our approach is feasible and useful for model checking the FMI co-simulation. The approach facilitates the verification of the CPSs architecture models.

V. RELATED WORK

For simulating CPSs [17], distinct simulation domains need to be integrated for a comprehensive analysis of the interdependent subsystems. Co-simulation [18] can maintain all system models within their specialized simulators and synchronizes them in order to coherently integrate the simulation domains. FMI [2] [11] is an industry standard which enables co-simulation of complex heterogeneous systems using multiple simulation engines.

Jens Bastian et al. adopts fixed step size master algorithm to simulate heterogeneous systems in [19]. David Broman et al. discussed the determinate composition of FMUs for co-simulation. To do that, they extended the FMI standard to designs FMUs that enables deterministic execution for a broader class of models. Besides, rollback and predictable step size master algorithms are proposed in their work. In [20], Fabio Cremona et al. presents FIDE, an Integrated Development Environment (IDE) for building applications using FMUs. In our recent work, we have implemented the prototype *co-simulator* for continuous-time Markov chains (CTMCs) [21], discrete-time Markov chains (DTMCs) [22] and Modelica models in [23]. We also proposed an improved co-simulation framework that focuses on the capture of nearest future event to reduce the number of running steps and the frequency of data exchange between models. In short, the existing work focus on how to achieve deterministic execution of FMUs and improve the efficiency of master algorithms, however, there is few work to analyse the correctness of master algorithms. PG Larsen et al. [24] presented formal semantics of the FMI described in the formal specification language CSP. They formally analyse the CSP model with the FDR3 refinement checker. Nuno Amalio et al. [12] presented an approach to verify both healthiness and well-formedness of an architecture design modeled with SysML. They attempt to check the conformity of component connectors and the absence of algebraic loops to ensure the co-simulation convergence.

In [25], Mladen Skelin et al. reports on the translation of the FSM-SADF formalism to UPPAAL timed automata that enables a more general verification than currently supported by existing tools. Stavros Tripakis [10] discussed the principles for encoding different modeling formalisms, including state machines (both untimed and timed), discrete-event systems, and synchronous dataflow, as FMUs. In this paper, our work focuses on the model and analyse I/O dependency information and master algorithms for FMI co-simulation. Compared with the existing work, the novelty of our approach is that it models the FMI co-simulation with timed automata. By this way, the existing model checker can be used to analyse and verify the co-simulation of CPSs. Moreover, we model and analyse three versions master algorithm to ensure the correctness of the co-simulation mechanism.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a novel approach to model check the FMI co-simulation, which facilitates the formal verification of CPSs. It involves model checking the reachability, livelock and deadlock of three various master algorithms. Besides, the correctness of the system architecture is also analysed. We encode the FMU component and master algorithms with timed automata, so that properties of the co-simulation can be verified with UPPAAL. To illustrate the feasibility of our approach, the example water tank is discussed. Its requirement is specified with SysML block diagrams, from which the relevant FMUs are derived to co-simulate the system behavior. With the help of encoding, the network of timed automata for the water tank system is built and verified with UPPAAL. The results show that the co-simulation behavior of CPSs can be analysed effectively with model checking technology.

An interesting direction of future work is to analyse and compare the performance of various master algorithms. Besides, some industrial case studies will be conducted to check the scalability of our approach. The tool implement of co-simulation should also be improved further.

ACKNOWLEDGEMENT

This work was supported by NSFC (Grant No.61472140, 61202104).

REFERENCES

[1] S. Zanero, "Cyber-physical systems," *IEEE Computer*, vol. 50, no. 4, pp. 14–16, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.105

[2] T. Blochwitz, "The functional mockup interface for tool independent exchange of simulation models," no. 2011-03-22, pp. 105–114, 2011.

[3] B. V. Acker, J. Denil, H. Vangheluwe, and P. D. Meulenaere, "Generation of an optimised master algorithm for FMI co-simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, part of the 2015 Spring Simulation Multiconference, SpringSim '15, Alexandria, VA, USA, April 12-15, 2015*, 2015, pp. 205–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=2872993

[4] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(94)90010-8

[5] M. Rahim, A. Hammad, and M. Ioualalen, "A methodology for verifying sysml requirements using activity diagrams," *ISSE*, vol. 13, no. 1, pp. 19–33, 2017. [Online]. Available: http://dx.doi.org/10.1007/s11334-016-0281-y

[6] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, "UPPAAL 4.0," in *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, 2006, pp. 125–126. [Online]. Available: http://dx.doi.org/10.1109/QEST.2006.59

[7] B. Cheng, X. Wang, J. Liu, and D. Du, "Modana: An integrated framework for modeling and analysis of energy-aware cpss," in *IEEE Computer Software and Applications Conference*, 2015, pp. 127–136.

[8] P. Fritzson and V. Engelson, "Modelica - a unified object-oriented language for system modelling and simulation," *Lecture Notes in Computer Science*, vol. 1445, no. 1445, pp. 67–90, 1998.

[9] D. Broman, C. X. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter, "Determinate composition of fmus for co-simulation," in *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, 2013, pp. 2:1–2:12. [Online]. Available: http://dx.doi.org/10.1109/EMSOFT.2013.6658580

[10] S. Tripakis, "Bridging the semantic gap between heterogeneous modeling formalisms and FMI," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2015, Samos, Greece, July 19-23, 2015*, 2015, pp. 60–69. [Online]. Available: http://dx.doi.org/10.1109/SAMOS.2015.7363660

[11] T. Blochwitz, M. Otter, J. kesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Proceedings of the 9th International Modelica Conference*. The Modelica Association, 2012, pp. 173–184. [Online]. Available: http://dx.doi.org/10.3384/ecp12076173

[12] N. Amálio, R. Payne, A. Cavalcanti, and J. Woodcock, "Checking sysml models for co-simulation," in *Formal Methods and Software Engineering - 18th International Conference on Formal Engineering Methods, ICFEM 2016, Tokyo, Japan, November 14-18, 2016, Proceedings*, 2016, pp. 450–465. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47846-3_28

[13] O. Semeráth, Á. Barta, Á. Horváth, Z. Szatmári, and D. Varró, "Formal validation of domain-specific languages with derived features and well-formedness constraints," *Software and System Modeling*, vol. 16, no. 2, pp. 357–392, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10270-015-0485-x

[14] D. Dori, *Model-Based Systems Engineering with OPM and SysML*. Springer, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-1-4939-3295-5

[15] I. K. Pepper and R. Wolf, "International council on systems engineering," *Police Journal*, vol. 88, no. 3, pp. 7–7, 2015.

[16] H. Boucheneb, G. Gardey, and O. H. Roux, "TCTL model checking of time petri nets," *J. Log. Comput.*, vol. 19, no. 6, pp. 1509–1540, 2009. [Online]. Available: http://dx.doi.org/10.1093/logcom/exp036

[17] H. Georg, S. C. Müller, C. Rehtanz, and C. Wietfeld, "Analyzing cyber-physical energy systems: The INSPIRE cosimulation of power and ICT systems using HLA," *IEEE Trans. Industrial Informatics*, vol. 10, no. 4, pp. 2364–2373, 2014. [Online]. Available: http://dx.doi.org/10.1109/TII.2014.2332097

[18] S. Bogomolov, M. Greitschus, P. G. Jensen, K. G. Larsen, M. Mikucionis, A. Podelski, T. Strump, and S. Tripakis, "Co-simulation of hybrid systems with spaceex and uppaal," 2015.

[19] J. Bastian, C. Clau, S. Wolf, and P. Schneider, "Master for co-simulation using fmi," 2011.

[20] F. Cremona, M. Lohstroh, S. Tripakis, C. X. Brooks, and E. A. Lee, "FIDE: an FMI integrated development environment," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, 2016, pp. 1759–1766. [Online]. Available: http://doi.acm.org/10.1145/2851613.2851677

[21] V. Danos, T. Heindel, I. Garnier, and J. G. Simonsen, "Computing continuous-time markov chains as transformers of unbounded observables," in *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, 2017, pp. 338–354. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-54458-7_20

[22] M. Guerry, "On the embedding problem for discrete-time markov chains," *J. Applied Probability*, vol. 50, no. 4, pp. 918–930, 2013. [Online]. Available: http://dx.doi.org/10.1017/S002190020001370X

[23] J. Liu, K. Jiang, X. Wang, B. Cheng, and D. Du, "Improved co-simulation with event detection for stochastic behaviors of cpss," in *40th IEEE Annual Computer Software and Applications Conference, COMPSAC 2016, Atlanta, GA, USA, June 10-14, 2016*, 2016, pp. 209–214. [Online]. Available: http://dx.doi.org/10.1109/COMPSAC.2016.133

[24] P. G. Larsen, J. Fitzgerald, J. Woodcock, and P. Fritzson, "Integrated tool chain for model-based design of cyber-physical systems: The into-cps project," in *International Workshop on Modelling, Analysis, and Control of Complex Cps*, 2016, pp. 1–6.

[25] M. Skelin, E. R. Wognsen, M. C. Olesen, R. R. Hansen, and K. G. Larsen, "Model checking of finite-state machine-based scenario-aware dataflow using timed automata," in *10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015, Siegen, Germany, June 8-10, 2015*, 2015, pp. 235–244. [Online]. Available: http://dx.doi.org/10.1109/SIES.2015.7185065