Model Checking Coordination of CPS Using Timed Automata

Abstract—**The growing complexity of Cyber-Physical Systems (CPS) increasingly challenges the existing methods and techniques. The correctness of coordination between heterogeneous components of CPS is still a challenging problem. The coordination of CPS could be implemented with co-simulation technology, which uses Functional Mock-up Interface (FMI) techniques to generate simulations of heterogeneous components in CPS. However, the master algorithm for co-simulation may be livelock or deadlock. Moreover, the architecture modeling of CPS may also introduce an algebraic loop which is a feedback loop resulting in cyclic dependencies. To solve these problems, we propose a novel approach for model checking several properties of coordination such as deadlock, liveness and reachability. We model the architecture of CPS with SysML Block Definition Diagrams (BDDs) and Internal Block Diagrams (IBDs), which capture the dependence of Functional Mock-up Units (FMUs) and the orchestration of the master algorithm. According to BDD models, the coordination between components is implemented with the master algorithm. We model three various master algorithms with Timed Automata (TA). Besides, we encode FMU components with TA to bridge the semantics gap between FMU and TA. With the help of the model checker UPPAAL, we can analyse the correctness of the master algorithms and detect whether there is an algebraic loop in the architecture. By this way, the coordination of CPS is verified with model checking.**

To illustrate the feasibility of our approach, the case study water tank is presented. The experiment results show that our approach facilitates model checking coordination of CPS. The novelty of our work is that our approach provides a feasible framework for model checking coordination of CPS with TA.

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

I. Introduction

Cyber-physical systems (CPS) 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 CPS. The components of CPS are of various types, requiring interfacing and interoperability across multiple platforms and different models of computation. Verifying coordination of heterogeneous CPS is a challenging problem. The coordination between heterogeneous components of CPS could be implemented with Functional Mock-up Interface (FMI) based co-simulation technology. 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 [2]. FMI supports simulation of complex systems composed of heterogeneous components, by coupling different models with their own solvers in a co-simulation environment.

In this paper, we focus on verifying the coordination of CPS which implemented with FMI 2.0 [3] based co-simulation. The key point is Master Algorithm (MA) [4] and connector configuration between Functional Mock-up Units (FMUs) [5], which specifies the orchestration and the exchange of data among FMUs during the whole coordination process. To ensure the correctness of coordination, we need to verify MA and connector configuration with model checking. However, MA is not a part of FMI standard. This implies that the user or tool vendor needs to develop a sophisticated orchestration algorithm for the problem at hand. There are three versions of MA [6]: fixed step algorithm, rollback algorithm and predictable step size algorithm. Rollback and predictable step size algorithms are based on the extension of FMI 2.0, which supports the rollback and a predict function. P.G. Larsen et al. [7] formally analysed the fixed step and rollback algorithms with the FDR3 refinement checker. However, there still lacks effective approach to verify the whole FMI-based coordination process. Based on our previous work [8], we found that the simulation process of coordination is time-intensive. Therefore, it is reasonable to formalize the coordination with Timed Automaton (TA) [9], which is the classic formalism for modeling real-time system. In this paper, we propose to model the MA with TA and verify the correctness of MA. Furthermore, we also attempt to encode the components of CPS with TA and verify the architecture of whole system with the model checker UPPAAL [9].

In summary, our main contributions are as follows:

- We propose a novel approach to verify the coordination of CPS with model checking. To bridge the gap between FMU and the model checker, we propose some encoding rules to encode FMU as TA.
- We model and verify three various MAs to ensure the correctness of the coordination. With the help of UPPAAL, we analyse the reachability, livelock and deadlock of three versions of MA.
- The prototype for model checking coordination of CPS is under developing, which is integrated in our Modana platform [10]. We have implemented the *SysML modeling environment* and the *co-simulator* to simulate CPS in Modana (https://github.com/ECNU-MODANA/AL-Modana.git) [11].

The main novelty of our work, compared with the previous work, is that we propose to verify coordination of CPS with TA-based model checking. As far as we know, there is few existing approaches supporting TA-based model checking for the coordination of CPS.

The remainder of this paper is organized as follows. In Section II, we briefly review the technical background including FMI, FMU and TA. Then, we present the technical road map of our approach and discuss how to encode FMU as TA with the help of their semantic mapping rules in Section III. In Section IV, we model three versions of MA with TA and verify their properties such as the livelock and deadlock. Section V presents a case study to demonstrate the feasibility of our approach. We model the architecture of water tank system with SysML Block Definition Diagram (BDD) [12], and then obtain the FMU component of each block and the connection between FMUs. We encode the FMUs of water tank with TA and verify the correctness of the coordination between components of water tank system with UPPAAL. Finally, we position our work with respect to related work before concluding and discussing possible future extensions.

II. Background

In this section, we brifely recall the syntax and semantics of FMU and TA. The semantic gap is the main problem when we apply TA-based model checking technology to verify coordination between FMUs. To bridge the semantic gap between FMU and TA, we propose encoding rules to specify FMU with TA. The detailed encoding process will be discuss in Section III.

A. FMU

An FMU is a component which implements the methods defined in the FMI API [5]. We present the syntax and semantics of FMU. The aim is to encode FMU as TA based on their semantics.

Definition 1. FMU syntax

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 variables of F. Note that an element $u \in U$ is a variable which ranges over a set of values $\mathbb{V}$.
- Y denotes the set of output variables of F. Each $y \in Y$ ranges over the 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.
- $s_0 \in S$ denotes the initial state of F.
- $set : S \times U \times \mathbb{V} \to S$ denotes that the function 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 a new state obtained by setting u to v.
- $get : S \times Y \to \mathbb{V}$ denotes that the function 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 that the function implements one simulation step. Given current state s, and a non-negative real $h \in \mathbb{R}_{\geqslant 0}$, $doStep(s, h)$ 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 $0 \leqslant h' < h$, it means that F rejects the time step h, but makes partial progress up to h', and reaches the new state 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 function $doStep$, which is a function of a Timed Input Sequence (TIS). 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 = $(t_0, s_0, v_0, v_0'), (t_1, s_1, v_1, v_1'), (t_2, s_2, v_2, v_2'), ..., (t_i, s_i, v_i, v_i'), (t_{i+1}, s_{i+1}, v_{i+1}, v_{i+1}'), ...$ 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}.

Therefore, the semantics of an FMU can be defined by a labelled transition system.

B. Timed Automaton

TA [9] is a classic theory to model the behavior of real-time system. It provides a powerful way to annotate state-transition graphs with many real-valued clocks. In this subsection, we recall the syntax and semantics of TA.

Definition 3. TA syntax

A TA over a finite set of clocks X and a finite set of actions Act is a quadruple $A = (L, l_0, E, I)$, where:

- L is a finite set of locations which ranges over by l;
- $l_0 \in L$ is the initial location;
- 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 $\bowtie \in \{<, \leqslant, \geqslant, >, =\}$.
- $E \subseteq L \times G(X) \times Act \times 2^X \times L$ is a set of edges labelled with guards and a set of clocks, where $Act = Act_i \cup Act_o$. Act_i is a set of input actions and Act_o is a set of output actions.
- $I : L \to G(X)$ assigns invariants to locations.

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 $x \in Y$, $v[Y = 0](x) = 0$.

Definition 4. TA semantics

The semantics of a TA $A = (L, l_0, E, I)$ is defined by a labelled transition system $L_A = (Proc, Lab, \{\xrightarrow{\alpha} \mid \alpha \in Lab\})$, where:

- $Proc = \{(l, v) \mid (l, v) \in L \times (X \to \mathbb{R}_{\geqslant 0})$ and $v \models I(l)\}$, i.e., states are of the form (l, v), where l is the location of the TA and v is a valuation that satisfies the invariant of $I(l)$;
- $Lab = Act \cup \mathbb{R}_{\geqslant 0}$ is the set of labels; and
- the transition relation is defined by $(l, v) \xrightarrow{\alpha} (l', v')$ if there is an edge $(l \xrightarrow{g,\alpha,r} l') \in E$, such that $v \models g$, $v' = v[r]$ and $v' \models I(l')$
$(l, v) \xrightarrow{d} (l, v+d)$ for all $d \in \mathbb{R}_{\geqslant 0}$, such that $v \models I(l)$ and $v + d \models I(l)$

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 TA. For universality, the definition uses arbitrary sets of clock valuations.

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

$$(l, t) \xrightarrow{x_1} (l, t + x_1) \xrightarrow{x_2} (l, t + x_1 + x_2) \xrightarrow{x_3} (l, t + x_1 + x_2 + x_3) \xrightarrow{x_4} ... \xrightarrow{x_i} (l, t + x_1 + x_2 + x_3 + ... + x_i) \xrightarrow{x_{i+1}} ...$$

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

III. Our Approach

In this section, we present a novel approach to verify the properties of coordination with TA. Therefore, we need first to bridge the semantic gap between FMUs and TA. We propose encoding rules to encode FMUs as TA. In Section V, we will apply these encoding rules to encode FMUs in our case study with TA, so that we can verify the coordination of the case study with the model checker UPPAAL.

A. Framework of our approach

The schematic view of our approach is shown in Fig. 1. We model the architecture of CPS with SysML BDD and SysML Internal Block Diagram (IBD) at the **modeling phase**. Each block of BDD represents a component of CPS and the connection between components is modeled with IBD. To simulate the whole system with co-simulation techniques, each block is modeled with an FMU and the IBD is described with the connector configuration in the **achitecture design phase**. We design MA to accomplish the communication between FMUs, which orchestrates coordination between components. Before simulating the system, we need to ensure the coordination of system is correct in **verification phase** which is the main contribution of our work. To verify the correctness of the coordination, firstly, we proposed encoding rules to encode FMU as TA, and model connector configuration as the channels between TA. Furthermore, we model MA with TA and verify the correctness of MA. By this way, we can obtain the network of TA composed with TA and channels. With the help of the model checker UPPAAL [9], some properties (e.g., livelock or deadlock) can be verified. Once the correctness of coordination is ensured, we can use co-simulation engine to simulate the whole system and generate the simulation traces. In this paper, we focus on how to verify the correctness of MA and coordination. In the following sections, we explain the main contributions of our approach in details.

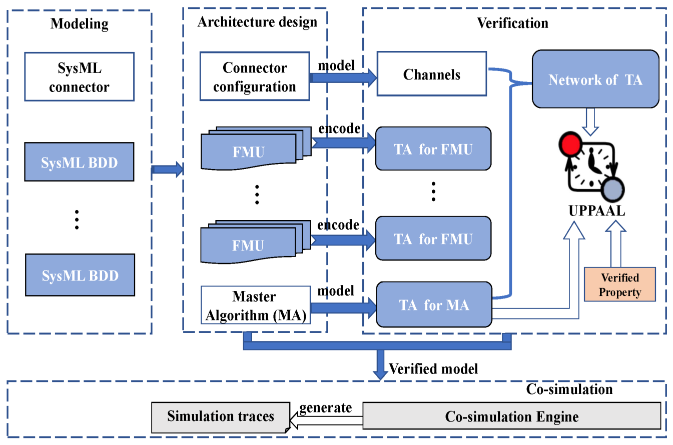


Fig. 1. A schematic view of our approach.

B. Encoding FMUs as TA

We find that there is a semantic gap between FMUs and TA. The former focuses on the execution sequence of FMU, which specifies the state change process with time elapsing. Essentially, the execution trace of TA is semantic equivalence to the execution sequence of FMU. Naturally, we propose to encode FMU as TA to analyse the behavior of FMUs without exploring its internal structure. Given an FMU $F = (S, U, Y, D, s_0, set, get, doStep)$, we encode the FMU as TA $A = (L, l_0, E, I)$, the congruent relationships between them are as following:

- L is a set of finite locations. Note that the state of the transition system L_A can be seen as the state of F, i.e., $(l, v) \Rightarrow s$.
- The initial state of the transition system L_A can be seen as the initial state of F, i.e., $(l_0, v_0) \Rightarrow s_0$.
- Each input variable $u \in U$ ranges over $Act_i \cup \{absent\}$.
- Each output variable $y \in Y$ ranges over $Act_o \cup \{absent\}$.
- An input action $e \in Act_i$ is such that the function set of F sets the input variable u with a given value.
- An output action $e \in Act_o$ indicates that the function get of F gets the output variable y. The set of values in the Act_i can be seen as Y of F.
- The communication between TA is the same as the I/O dependencies information between the FMUs. $(u, y) \in D$ denotes that output y depend on input u. The output actions also depend on the input actions in TA.
- For any $e \in Act$ of A, there is a transition $s \xrightarrow{e} s'$, which may be found after the function $doStep$ execute. 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, the time step h may be rejected, if there is a rollback behaviour happens, the transition in TA could be an edge $l' \xrightarrow{e} l$, which denotes that a location travels to the former location.

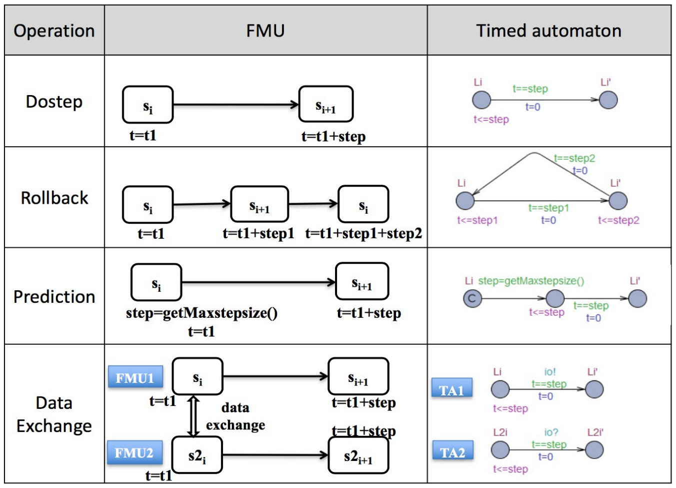


Fig. 2. Encoding rules from FMU to TA.

It is not easy to map FMUs to TA directly. S. Tripakis encodes TA as FMUs in [5]. Inspired by their work, we propose some encoding rules to encode FMUs as TA according to the congruent relationships. 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 as 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 s_i 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 units and reach a new location L_i' after a transition, and then returns to the former location L_i.

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

For $DataExchange$ 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 exchanging.

To bridge the semantic gaps between FMU and TA, we provide proper encoding rules to formalize FMUs as TA. It lays the foundation to analyse coordination of CPS with TA-based model checking. As for the correctness of encoding rules, we analyse the equivalence of execution fragments as following:

- For the encoding rule of $Dostep$ operation, the execution fragments of FMU and TA are (s_i, t_1), (s_{i+1}, $t_1 + step$) and (l_i, t), (l_i', $t + step$). It means that TA and FMU execute $step$ time units, and jump to a new state or location.
- For encoding rule of $Rollback$ operation, the execution fragments of FMU and TA are (s_i, t_1), (s_{i+1}, $t_1 + step1$), (s_i, $t_1 + step1 + step2$) and (l_i, t), (l_i', $t + step1$), (l_i, $t + step1 + step2$). It means that TA and FMU execute $step1$ time units, and jump to a new state or location, and then execute $step2$ time units and return to the previous state or location.
- For the encoding rule of $Prediction$, the execution fragments of FMU and TA are (s_i, t_1), (s_{i+1}, t_1+$step$) and (l_i, t), (l_i', $t + step$). It means that TA and FMU predict the step size $step$, and then execute $step$ time units.
- For the encoding rule of $DataExchange$ operation, the execution fragments of FMU1 and TA1 are (s_i, t_1), (s_{i+1}, $t_1 + step$) and (l_i, t), (l_i', $t + step$). The execution fragments of FMU2 and TA2 are ($s2_i$, t_1), ($s2_{i+1}$, $t_1 + step$) and ($l2_i$, t), ($l2_i'$, $t + step$). It means that FMUs or TA exchange data, and then execute $step$ time units.

We have analysed the whole execution trace of FMU and TA for these encoding rules. According to the encoding rules, we find that the execution traces of FMUs and TA are equivalent which means the encoding rules work well. In section V, we apply these encoding rules to the water tank system. According to the simulation fragments of the case study, we also find that the encoding rules work well.

IV. Modeling and Analysis of Master Algorithm

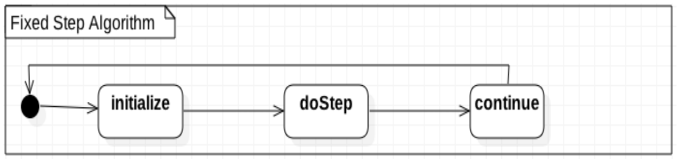
The MA provides the orchestration of FMUs, which denotes the co-simulation of various FMUs. To ensure the correctness of coordination, it is necessary to verify certain properties of the MA. In this section, we model three versions of MA with TA and verify some expected properties of MA such as deadlock, livelock and reachability with UPPAAL.

A. I/O Dependency Information

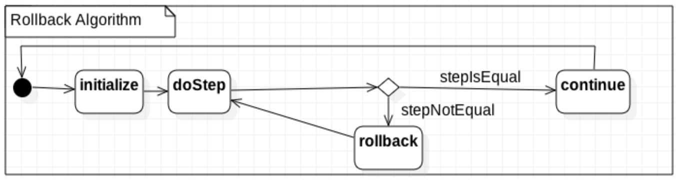
When it comes to co-simulation, I/O dependency information [6] is inevitably required to be well considered. The MA calls function Set to provide input value to an FMU and the function Get to obtain an output value. It is of vital importance to know the dependence between input and output of FMUs. The 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 obtained with the element $ModelStructure$ [13]. However, sometime there may be an algebraic loop generated by the sequence of function call, which may not converge. In section V, we present water tank system to show how to detect algebraic loops in the architecture.

B. Master Algorithm

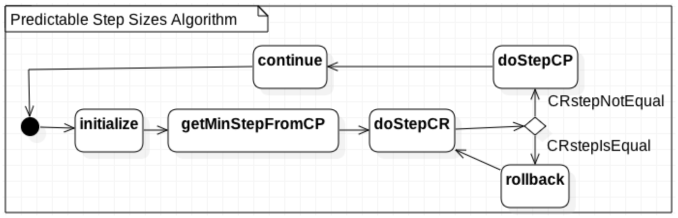
The MA is used to orchestrate the execution of different components or subsystems. Each subsystem serves as an FMU component whose simulation is triggered by a particular MA. FMU can be seen as a black box which can be simulated independently until it exchanges data or synchronizes. There are three versions of MA, which are 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 MAs.

1) Fixed Step Algorithm: For fixed step algorithm, all FMUs have the same step size. When MA 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 its 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 the next simulation step. The activity diagram for 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 [6], the co-simulation process should be reliable, when all FMUs are reliable. When some error happens during a simulation step, the co-simulation will be affected due to 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, FMU_1 accepts the request and FMU_2 rejects it. If we save the state of FMU_1 and FMU_2 at the communication point, we can restore the scene after FMU_2 rejects $doStep$. The activity diagram of rollback algorithm [6] 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 could return to the previous state if the step sizes of all FMUs simulation are not equal.

3) Predictable Step Size Algorithm: To improve the efficiency of MA, it is important to predict the step size. So, predictable step size algorithm was proposed [6]. 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. When $discard$ and $error$ occur, the FMU needs to rollback to the previous saved state. Whether an FMU is predictable or not, it 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 can get the maximum step size of the FMU using $GetMaxStepSize$ function.

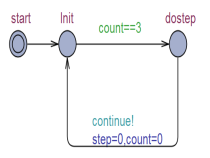
In predictable step size algorithm, MA chooses the maximum step size of all predictable FMUs and finds the smallest communication step size h which ensure all predictable step size can be accepted. Then, the states of all FMUs are saved. MA calls $doStep(h)$ function of FMUs which support rollback. The function $doStep()$ will return the real performed step size. If all performed step sizes are equal to h, MA will call $doStep(h)$ for FMUs. Otherwise, MA will find the smallest performed step h_{min}, then all FMUs will restore the saved state. 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 chooses the smallest one of them.

C. Modeling and Verification of MA

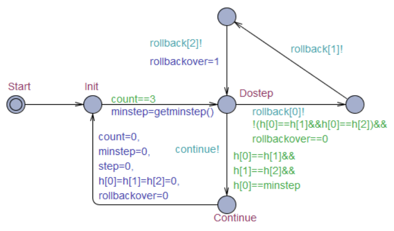
UPPAAL is a toolset for verification of real-time systems modeled by a network of TA which is extended with integer variables, structured data types, and channel synchronization. The Fig.4 shows the TA models of three MAs, respectively. Fixed step algorithm has $Init$, $doStep$ states and synchronize 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 $step2$ of FMUs supporting $GetMaxStepSize$ function and the maximal step size $step1$ 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 three various MAs including reachability, liveness and deadlock. Experimental results are shown in Table I.

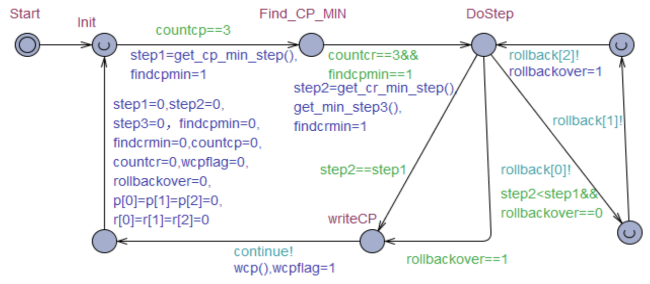
- $E\langle\rangle$ $master.dostep$, $E\langle\rangle$ $master.Continue$ and $E\langle\rangle$ $master.writeCP$ are reachability properties checking whether the system can reach these states;
- $master.Init \rightarrow master.dostep$, $master.Init \rightarrow master.Continue$ and $master.Init \rightarrow master.Continue$ are liveness properties. If the MA arrives at the former state, it eventually reaches the latter state;



(a) TA for fixed step algorithm



(b) TA for rollback algorithm



(c) TA for predictable step size algorithm

Fig. 4. TA for three versions of MAs.

TABLE I. Experimental results for verifying MA

MA	Verified 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

- $A[]\ not\ deadlock$ is safety property, which means whether the MA will be deadlock.

Table I shows that the properties such as deadlock, liveness and reachability are satisfied, which ensures that the correctness of MA. For example, The property $A[]\ not\ deadlock$ is satisfied, which means the MA is deadlock free. The property $master.Init \rightarrow master.doStep$ is satisfied, which means if the model reached the former state $Init$, it will eventually reach the state $doStep$. The property $E\langle\rangle\ master.doStep$ is satisfied, which means there exists a reachable state $doStep$. In short, the coordination of CPS involves architecture and MA. In this section, we have verified the correctness of various MAs, which will be applied to the case study.

V. Case Study

To illustrate our approach, we take an example water tank as the case study inspired by [14]. According to the I/O dependency information between FMUs, the architectural model for water tank is constructed using SysML BDD, which helps to model the components and their connections.

The water tank system is shown in Fig. 5. 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 depending on the water level. In this paper, we illustrate three various version of water tank systems to show the scability of our approach. The difference between various water tank cases lies in various connection way between the controller, valve and tank.

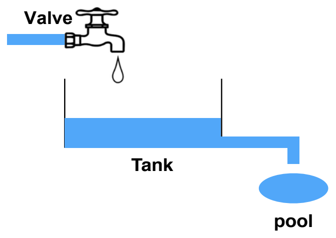


Fig. 5. Water tank system.

A. Architecture Modeling with SysML

SysML is a general purpose Domain-Specific Language (DSL) [12] for Model-Based Systems Engineering (MBSE) [15], which is originated as an initiative of the International Council on Systems Engineering (INCOSE) [16] in January 2001. The SysML BDD describes the structure of the system with blocks. The IBD describes the internal structure and connections of blocks. The ports of blocks are connected by the connector. The I/O dependence of blocks describes the communication between blocks. SysML BDD is usually used to model the architecture of system.

Fig. 6 shows the SysML BDD for the water tank system, which consists of three blocks: $Valve$, $Tank$ and $Controller$. $Valve$ 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 $OpenClosed$ signal.

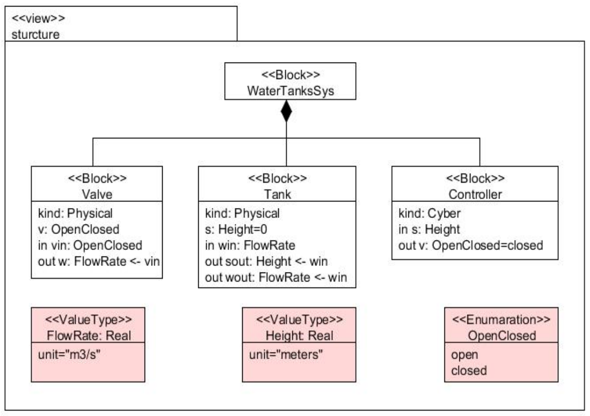
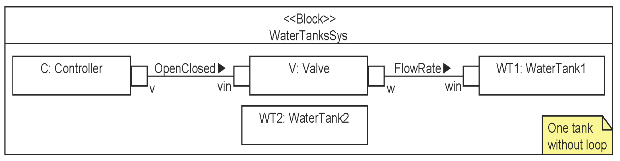
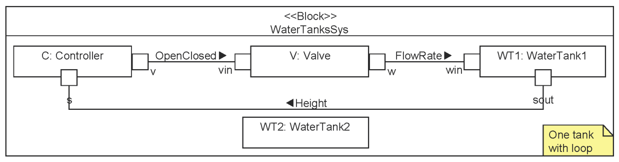


Fig. 6. SysML BDD for water tank.

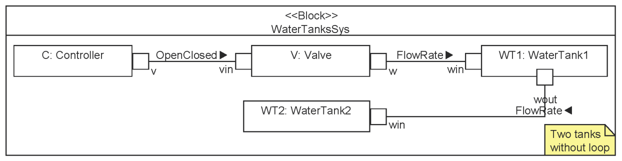
Fig. 7 shows the connection diagram for the system. There are three cases for connections. The first case is that the system has one valve, one controller and one tank. The controller sends stochastic signals to control the valve on/off leading to various rates of water flow. The second case is that the signal from the controller is affected by the water level of the tank. The last case is modified based on the first case and added another $waterTank2$ which is affected by the flow rate of the $waterTank1$.



(a) Connection case 1



(b) Connection case 2



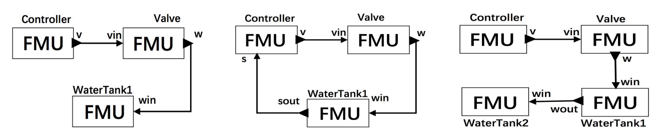
(c) Connection case 3

Fig. 7. SysML IBD for water tank.

The SysML BDD shows the blocks of system and SysML IBD shows the connection between blocks. In next section, we abstract each block as an FMU, and model SysML IBD as the connector configuration.

B. FMUs Connection of Water Tank

Fig. 8 is the FMUs and FMUs connection of water tank system. There are three connection cases between the FMUs according to the SysML IBD shown in Fig. 7. The first case contains three FMU components $Controller$, $Valve$ and $WaterTank1$ and two channels v_vin, w_win as shown in Fig.8(a). The $Controller$ and $Valve$ are connected with channel v_vin. The $Valve$ and $WaterTank1$ are connected with channel w_win. The second case is shown in Fig.8(b), there could be a channel $sout\_s$ between $WaterTank1$ and $Controller$, which means the water level of $WaterTank1$ affects the control strategy of the **controller**. Fig. 8(c) shows the third case, there could be another $WaterTank2$, the $WaterTank1$ and $WaterTank2$ are connected by the channel w_out.



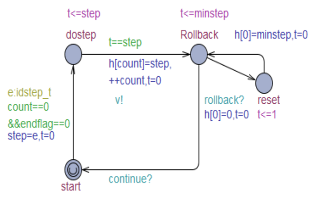
(a) Case 1 (b) Case 2 (c) Case 3

Fig. 8. FMUs connections of three water tank cases.

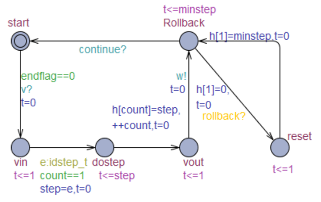
Until now, we design the architecture and the connections of the case study. Before the system is co-simulated in the engine, how can we assure the correctness of the architecture models? To solve this problem, we use model checking to verify it. This is one of our main contributions. More details of verification process can be found in the next section.

C. Verification and Analysis with UPPAAL

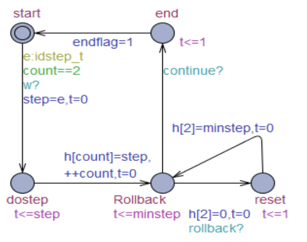
This section performs a formal analysis of the architectures of water tank. First of all, we encode FMUs of the water tank and model the MA with TA which composes a network of TA. Next, the models are verified with the model checker UPPAAL. The execution of FMU and co-simulation is time-related. We abstract the execution of FMUs for the water tank and encode it with the locations and transitions of TA according to the encoding rules proposed in Section III. Besides, we also model the MA as a TA to coordinate the execution between several FMUs. The TA templates for FMUs and MA are shown in Fig.9. Here, we adopt the rollback MA to coordinate the FMUs. The other two MAs can be analysed with the similar way. We do not present the details of them due to the space limits of this paper.



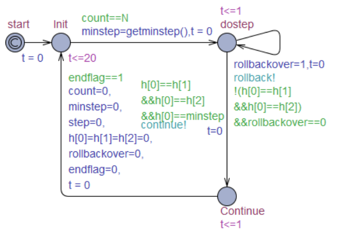
(a) TA for FMU_controller



(b) TA for FMU_valve

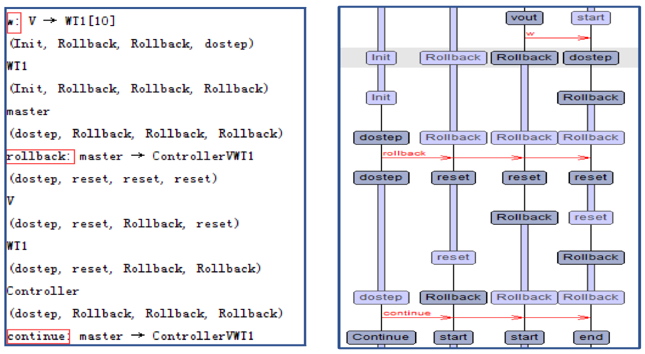


(c) TA for FMU_WaterTank1



(d) TA for rollback MA

Fig. 9. TA Network for connection case 1: $controller$ || $valve$ || $WaterTank1$ || MA.



```
w: V → WT1[10]
(Init, Rollback, Rollback, dostep)
WT1
(Init, Rollback, Rollback, Rollback)
master
(dostep, Rollback, Rollback, Rollback)
rollback: master → ControllerVWT1
(dostep, reset, reset, reset)
V
(dostep, reset, Rollback, reset)
WT1
(dostep, reset, Rollback, Rollback)
Controller
(dostep, Rollback, Rollback, Rollback)
continue: master → ControllerVWT1
```

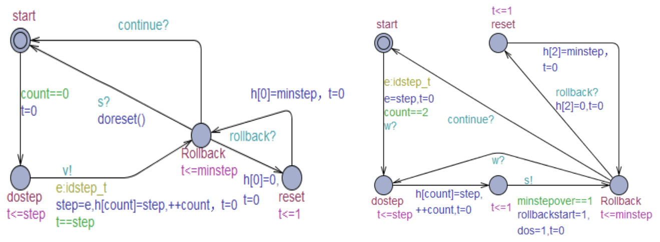
(a) Execution trace (b) Execution sequence diagram

Fig. 10. The execution fragment of the coordination in UPPAAL.

Fig. 9(a), 9(b), 9(c) are the templates for $controller$, $valve$ and $WaterTank1$ respectively, which model FMUs of the water tank. These FMUs have four key states: $start$, *dostep*, *Rollback* and *reset*. Fig. 9(a) shows the template for *controller* which executes with the random step size. It synchronizes with *valve* by signal v and transfers to *Rollback* state, and then waits for a signal from the MA. Until the *controller* receives the *continue* signal, it does data exchange with other FMUs, and returns to *start* state. Otherwise, it receives *rollback* signal, when it obtains the minimize step size of all FMUs, it transfers to *Rollback* state. The states and transitions of *valve* and $WaterTank1$ template are similar with the template of *controller*. Fig. 9(d) shows the template for the MA. Firstly, the MA initializes the parameters, and then it gets minimize step size of FMUs until all FMUs visit *dostep*. Next, the MA decides which signal should be sent according to the guard. If the step sizes of all FMUs are equal, the MA will send *continue* signal, otherwise, send *rollback* signal.

Fig. 10 is the execution fragment of the coordination in UPPAAL, we can find that *valve* sends a w signal to perform data exchange with $WaterTank1$. After that, $WaterTank1$ moves to *dostep* state. The MA broadcasts a *rollback* signal to all templates, which leads to all of them arrive at *reset* state. Finally, the MA sends a *continue* signal to all FMUs. All templates return to *start* state, and then do the next step. The execution fragments show that our models are correct.

In order to compare the behavior of three connection cases of water tank system, we also model the other two connection cases in UPPAAL. For connection case 2, we add channel s in the templates for *controller* and $WaterTank1$ as shown in Fig.11. For connection case 3, we create template for $WaterTank2$ and channel $w2$ as shown in Fig.12. The other models are the same as models of connection case1. Limited to the length of this paper, we only show the templates for *controller* and $WaterTank1$ of connection case 2 and template for $WaterTank2$ of connection case 3. In the next subsection, we verify some properties of various connection cases to detect whether there is an algebraic loop in the architecture.



(a) TA for FMU_controller (b) TA for FMU_WaterTank1

Fig. 11. TA for connection case 2.

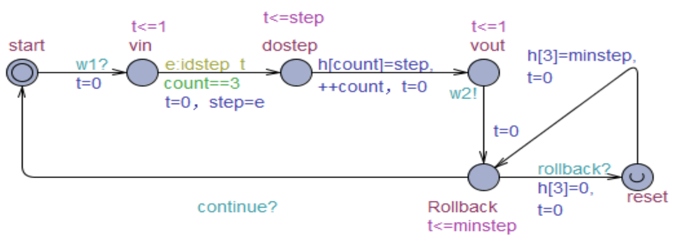


Fig. 12. TA for FMU_WaterTank2 of connection case 3.

TABLE II. EXPERIMENTAL RESULTS FOR VARIOUS CONNECTION CASE

Connection case	Verified 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 supports a simplified version of TCTL [17] to specify the property. We verify the following properties of each connection case:

- $E\langle\rangle\ WT1.Rollback$ and $E\langle\rangle\ master.Continue$ specify reachability properties. It means the FMU of $WaterTank1$ will *Rollback* and the MA will reach *Continue* state.
- $master.start \rightarrow master.Continue$ specifies liveness property. It means once the MA start, it will continue eventually.
- $A[]\ not\ deadlock$ specifies safety property. It means the execution of the system will not be deadlock.

The verification results are listed in Table II. We can find that all properties of connection case 1 and case 3 are satisfied. It shows that our MA works well and the composition of FMUs is determinate. However, the liveness and reachability properties of connection case 2 are not satisfied. It means there is an algebraic loop which may be introduced with the I/O dependency in the architecture. The experimental results show that our approach is feasible and useful for model checking the coordination of CPS. Here, we only focus on the detection of algebraic loop and the correctness of coordination. In the future work, we will consider how to eliminate the algebraic loop.

VI. Related Work

For simulating CPS, H. Georg et al. proposed to integrate some distinct simulation domains for a comprehensive analysis of the interdependent subsystems in [18]. As a promising technique, co-simulation [19] can maintain all system models within their specialized simulators and synchronizes them in order to coherently integrate the simulation domains. FMI [20] [13] is an industry standard which enables co-simulation of complex heterogeneous systems using multiple simulation engines. It has been adopted by the industry and academic. For example, J. Bastian et al. adopted fixed step size MA to simulate heterogeneous systems in [21]. D. Broman et al. discussed the determinate composition of FMUs for co-simulation in [6]. They extended the FMI standard to design FMUs that enables deterministic execution for a broader class of models. Besides, rollback and predictable step size MAs are proposed in their work. In [22], F. Cremona et al. presented FIDE, which is 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)

[23], discrete-time Markov chains (DTMCs) [24] and Modelica models in [8]. We also proposed an improved co-simulation framework that focuses on the capture of the 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 MAs, however, there is few work to analyse MAs with formal methods.

Before the FMI-based co-simulating of the heterogeneous system, we should verify the coordination of the components in the system. There are some related works ahout coordination and the verification of FMI. F. Arbab et al. proposed a channel-based coordination model for component composition in [25]. S. Dziwok et al proposed specification and verification for real-time coordination protocols of cyber-physical systems in [26]. P.G. Larsen et al. [7] presented formal semantics of the FMI described in the formal specification language CSP. They formally analysed the CSP model with the FDR3 refinement checker. N. Amalio et al. [14] presented an approach to verify both healthiness and well-formedness of an architecture design modeled with SysML. They attempted to check the conformity of component connectors and the absence of algebraic loops to ensure the co-simulation convergence. In [27], M. Skelin et al. reported on the translation of the FSM-SADF formalism to TA that enables a more general verification. S. Tripakis [5] discussed the principles for encoding different modeling formalisms, including state machines (both untimed and timed), discrete-event systems, and synchronous data flow, as FMUs. **Compared to the existing work**, the novelty of our approach is that it verifies the coordination of CPS with TA. The execution of FMU and co-simulation is time related. It is natural to use TA by means of its powerful ability of specifying time and extensive tool support.

VII. Conclusion and Future Work

In this paper, we present a novel approach to model check the coordination of CPS. We model the architecture of CPS with SysML BDD and IBD, which capture the architecture of the system and the connection of blocks. According to BDD and IBD, each block is implemented as an FMU. The coordination between blocks are implemented with MA and connector configuration. To verify coordination, we proposed encoding rules to encode FMUs as TA. Besides, we model three various MAs with TA, which orchestrate the coordination between FMUs. With the help of the model checker UPPAAL, we analyse the correctness of MAs and detect whether there is an algebraic loop in the architecture. By this way, the coordination of CPS is verified. To illustrate the feasibility of our approach, the case study water tank is presented. The experiment results show that our approach is feasible and effective.

An interesting direction of future work is to analyse and compare the performance of various MAs. How to eliminate algebraic loop of the architecture is also an interesting problem. Besides, some industrial case studies will be conducted to check the scalability of our approach.

Acknowledgement

This work was supported by NSFC (Grant No.61472140, 61202104, China HGJ project No.2017ZX01038102-002).

References

[1] W. Wolf, “Cyber-physical systems,” pp. 8–9, 2009.

[2] C. Clau, “Modelisar: From system modeling to s/w running on the vehicle,” *Co-Simulation*.

[3] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, “Automatic generation of master algorithms for fmi 2.0 co-simulation,” 2006.

[4] B. V. Acker, J. Denil, H. Vangheluwe, and P. D. Meulenaere, “Generation of an optimised master algorithm for fmi co-simulation,” in *Symposium on Theory of Modeling and Simulation: Devs Integrative M and s Symposium*, 2015, pp. 205–212.

[5] S. Tripakis, “Bridging the semantic gap between heterogeneous modeling formalisms and fmi,” in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2015, pp. 60–69.

[6] D. Broman, C. 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*, 2013, pp. 1–12.

[7] 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.

[8]

[9] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, Y. Wang, and M. Hendriks, “Uppaal 4.0,” in *International Conference on the Quantitative Evaluation of Systems*, 2006, pp. 125–126.

[10]

[11] 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.

[12] O. Semerth, gnes Barta, kos Horvth, Z. Szatmri, and D. Varr, “Formal validation of domain-specific languages with derived features and well-formedness constraints,” *Software and Systems Modeling*, vol. 16, no. 2, pp. 357–392, 2017.

[13] M. Otter and D. Zimmer, “Proceedings of the 9th international modelica conference,” in *International Modelica Conference*, 2012.

[14] N. Amlio, R. Payne, A. Cavalcanti, and J. Woodcock, “Checking sysml models for co-simulation,” in *International Conference on Formal Engineering Methods*, 2016, pp. 450–465.

[15] D. Dori, *Model-Based Systems Engineering with OPM and SysML*. Springer New York, 2016.

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

[17] H. Boucheneb, G. Gardey, and O. H. Roux, “Tctl model checking of time petri nets,” *Journal of Logic and Computation*, vol. 19, no. 19, pp. 1509–1540, 2009.

[18] H. Georg, S. C. Mller, C. Rehtanz, and C. Wietfeld, “Analyzing cyber-physical energy systems:the inspire cosimulation of power and ict systems using hla,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2364–2373, 2014.

[19] 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.

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

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

[22] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, “Fide:an fmi integrated development environment,” 2016, pp. 1759–1766.

[23] V. Danos, T. Heindel, I. Garnier, and J. G. Simonsen, “Computing continuous-time markov chains as transformers of unbounded observables,” 2017.

[24] M. A. Guerry, “On the embedding problem for discrete-time markov chains,” *Journal of Applied Probability*, vol. 50, no. 4, pp. pgs. 918–930, 2013.

[25] F. Arbab, “Reo: a channel-based coordination model for component composition,” *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2002.

[26] S. Dziwok, “Specification and verification for real-time coordination protocols of cyber-physical systems,” Ph.D. dissertation, University of Paderborn, Germany, 2017.

[27] M. Skelin, E. R. Wognsen, M. C. Olesen, and R. R. Hansen, “Model checking of finite-state machine-based scenario-aware dataflow using timed automata,” in *IEEE International Symposium on Industrial Embedded Systems*, 2015, pp. 1–10.