 CMC, vol.58, no.1, pp.287-306, 2019

A Co-Verification Interface Design for High-Assurance CPS

Yu Zhang[1, 2, *], Mengxing Huang[1, 2, *], Hao Wang[3], Wenlong Feng[1, 2], Jieren Cheng[1, 2] and Hui Zhou[1, 2]

Abstract: Cyber-Physical Systems (CPS) tightly integrate cyber and physical components and transcend traditional control systems and embedded system. Such systems are often mission-critical; therefore, they must be high-assurance. High-assurance CPS require co-verification which takes a comprehensive view of the whole system to verify the correctness of a cyber and physical components together. Lack of strict multiple semantic definition for interaction between the two domains has been considered as an obstacle to the CPS co-verification. A Cyber/Physical interface model for hierarchical a verification of CPS is proposed. First, we studied the interaction mechanism between computation and physical processes. We further classify the interaction mechanism into two levels: logic interaction level and physical interaction level. We define different types of interface model according to combinatorial relationships of the A/D (Analog to Digital) and D/A (Digital to Analog) conversion periodical instants. This interface model has formal semantics, and is efficient for simulation and formal verification. The experiment results show that our approach has major potential in verifying system level properties of complex CPS, therefore improving the high-assurance of CPS.

Keywords: CPS, interface, co-verification, co-simulation, high-assurance.

1 Introduction

Cyber-Physical Systems (CPS) tightly integrates cyber and physical components, thereby creating opportunities for more direct integration of the physical world into the cyber world [Lee (2010)]. CPS has always been focused on integration of cyber and physical components. CPS is often mission-critical and usually subject to stringent safety and reliability requirements. CPS applications are many, including avionics, personalized health-care, intelligent transportation, smart grid and robotics as representative examples

[1] State Key Laboratory of Marine Resource Utilization in South China Sea, Hainan University, Haikou, 570228, China.

[2] College of Information Science and Technology, Hainan University, Haikou, 570228, China.

[3] Big Data Lab, Department of ICT and Natural Sciences, Norwegian University of Science and Technology, Postboks 1517, N-6025 Aalesund, Norway.

[*] Co-Corresponding Author: Mengxing Huang. Email: huangmx09@163.com;
Yu Zhang. Email: yuzhang2015@hainu.edu.cn.

where embedded cyber components tightly interact with physical components via sensor/actuator networks to ensure the delivery of the desired behaviors. Due to the inherent and ever-growing complexities, uncertain delay and the requirement of precise control, Cyber/Physical co-verification becomes more complicated.

Lack of strict multiple semantic definition for interaction between the two domains has been considered as an obstacle to the CPS co-verification. In previous work Yu et al. [Yu, Dong and Fei (2014); Yu, Fei, Dong et al. (2013)], we propose an automata-theoretic approach and CPS virtualization to check the properties of the system. However, there is no unifying formal model for representing the implementation semantic of Cyber/Physical interface accurately. In this paper, we propose an approach for delimiting the cyber/physical interface. First, we studied the interaction mechanism between computation and physical processes. We further classify the interaction mechanism into two levels: logic interaction level and physical interaction level. The interaction between the physical system being controlled and the software implementation of control algorithms forms the logic level interaction. Accordingly, the interaction, which application software interact with physical system through execution platform, forms the physical level interaction. Secondly, a Cyber/Physical interface model for hierarchical a verification of CPS is proposed. We define different types of interface model according to combinatorial relationships of the A/D (Analog to Digital) and D/A (Digital to Analog) conversion periodical instants. We advocate the use of Cyber/Physical interface model for bridging multiple semantic gap between cyber and physical components. This interface model has formal semantics which cover all Cyber/Physical interaction behaviors, and is efficient for simulation and formal verification. Finally, we have realized this approach and applied it to a real-world control system.

The rest of this paper is organized as follows. Section 2 elaborates the interaction mechanism in CPS. Section 3 presents the co-verification model. 4 evaluate our approach by some case studies. Section 5 reviews the related research works. Section 6 summarizes this paper and forecasts the direction of research work in the future.

2 Analysis of Co-Verification component in CPS

As illustrated in Fig. 1, there are three co-verification components in CPS: the cyber component, physical component and cyber/physical interface component. Cyber component monitors and controls the physical component, usually with feedback loops through cyber/physical interface component where physical component affect Cyber component and vice versa. The CPS cyber component refers to the abstract model of the control application. The CPS physical component is an abstract model for describing the physical system, can be further elaboration for the controlled object (plant) and the physical environment model.

In implementation level, cyber/physical interface model is divided into two types: logical interaction model and physical interaction model. Control software monitor and control physical model through logical interaction model, which is a logical coupling; Physical interaction model is based on the execution platform by A/D and D/A conversion to monitor and control physical model, which is physical coupling. Therefore, the interaction between the cyber model and physical model needs to describe the following

aspects: 1) the relationship between physical model and cyber model, which includes information flow and control flow; 2) sampling and control cycle time; 3) hardware platform specifications, operating system configuration, etc.

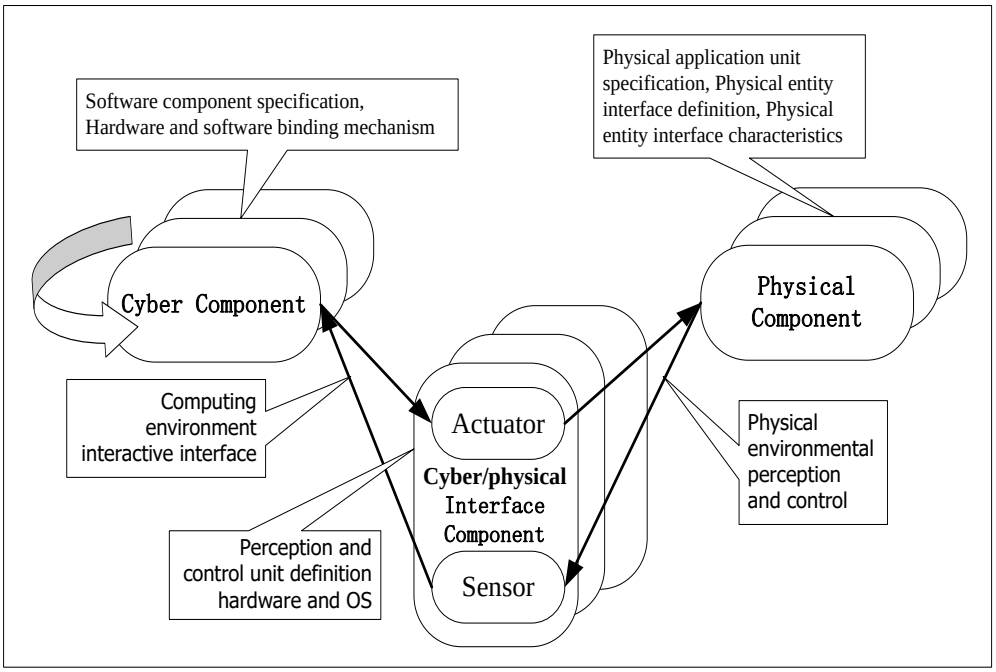

Figure 1: The co-verification components

Therefore, the interaction between the cyber model and physical model is mainly specified through the cyber/physical interface model. TableSat [Vess (2005)] is an interactive platform from the University of Michigan, which emulates in 1-degree-of-freedom the dynamics, sensing, and actuation capabilities required for satellite attitude control. NASA uses the simulation experiment platform to simulate the satellite attitude changes, sampling and control process. TableSat basic structure is shown in Fig. 2. TableSat cbyer model include Controller module and communication module. Control module is applied to implement the TableSat control algorithm, communication module realized communications functions with experimental machine (to simulate the ground satellite receiving station, ground station). TableSat execution platform, which including Athena II SBC, Network Device, A/D conversion, D/A conversion and Debian operating system. abstracted by interface model. (Light Sources and the Magnetic Fields is a physical environment model, TableSat rotary movement is plant model. Interface model realize the interaction between cyber model and the physical model. The control process is: based on the angular velocity measured by a high-precision rate gyro, the controller monitor and control physical model to stabilize the TableSat motion.

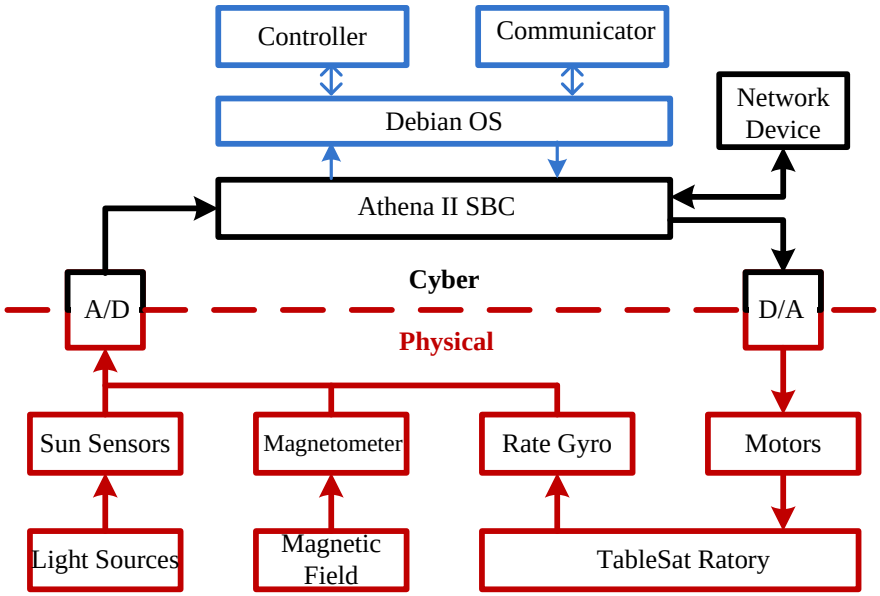

Figure 2: The TableSat

3 Cyber/physical co-verification model

For heterogeneous modeling, the difficulty lies in the interaction between cyber model and physical model. Cyber model is essentially a discrete event model, and its operational semantics refers to the execution sequence with a time stamp; Physical model is essentially a continuous time model, the model is formulated for differential equation. For integration of discrete event model and continuous time model, its execution sequence in the operational semantics is interaction protocols between the two heterogeneous models. Therefore, how to effectively model the interaction strategies to effectively deal with the discrete process and continuous process of different semantics is an important problem. Following are the formal description of object model under the co-verification framework.

As illustrated in Fig. 3, there are three types of primitive components in the co-verification: cyber component, physical component and cyber/physical interface component model. Cyber component includes control software and its running platform. There are two types of interaction between cyber and physical worlds: one is cyber/physical logical interaction between control software and plant, another is cyber/physical physical interaction between hardware and plant. Cyber/physical interface model bridges these two semantics gap between cyber and plant model by propagating events across Cyber/physical boundaries. As shown in Fig. 3, according to the hierarchical division on the logical interaction and physical interaction, the co-verification is divided into two levels of feedback loop: one is logic layer feedback loop which composed of control software and the physical system; another is the physical layer of the feedback loop which composed of the control software, execution platform and physical.

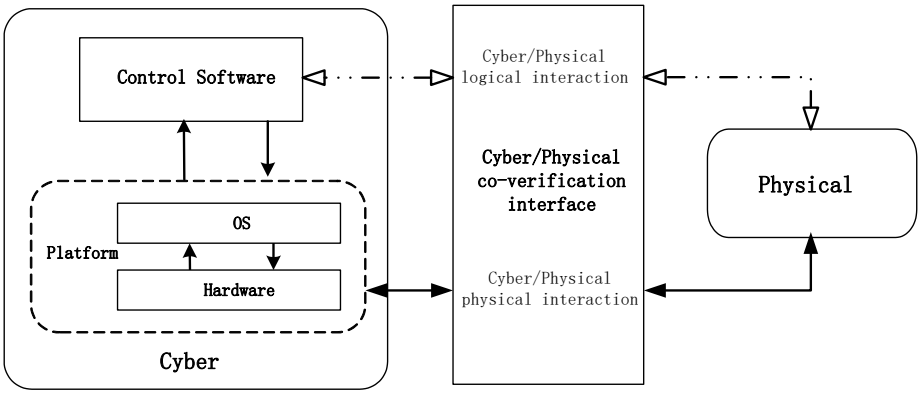

Figure 3: Formal framework for co-verification

Under this framework, the software components of an embedded system execute on generic hardware platform while the plant components are implemented as differential equations. The software components and plant components interact through an embedded OS that also schedules the execution of the software components. Software schedulers are not explicitly represented in this model. Instead, the timing parameters of a control task are integrated into the component model as assumptions of the components. The cyber/physical interface components and the timing parameter constraints together abstract the embedded OS by providing necessary information about timing parameters of control tasks.

Below we characterize the dynamic of the main components in detail and discuss how their integration is handled.

Definition1. A cyber model is a tuple $c = (S_{cyber}, TA)$, where S_{cyber} is static structure of cyber model, that is, source code. TA is a time automata, which is used to specified the dynamic behavior of cyber model. The set of all cyber model in a CPS is denoted as $C = \{c_1, c_2, \cdots c_n\}$

Definition2. A physical model is a tuple $p = (S_{physical}, HA)$, where $S_{physical} = (u, x, y, dom)$ is static structure of physical model, U, x and y are input, state, and output of physical model respectively , $dom : (u, x, y) \rightarrow DataType$, HA is a hybrid automata, which is used to specified the dynamic behavior of physical model. The set of all physical model in a CPS is denoted as $P = \{p_1, p_2, \cdots p_m\}$

Definition3. A Cyber/Physical Interface Model is a tuple $(Int, State, Event, Platform, Time)$, where Int is the name of the interface model, State are the state variables provided either by program or plant and accessible by both, Event is the access and modification to the State , Platform is the specification of running platform hardware, Time is the interaction time type of the cyber/physical interface. The set of all interface models in a CPS is denoted as $I = \{i_1, i_2, \cdots i_k\}$.

- In the cyber/physical logical interaction, Platform could be set to null;
- In the cyber/physical physical interaction, Platform must specify the running environment.

Interface events have two types: cyber or physical. When cyber updates the physical interface states, a cyber interface event occurs, and vice versa. For example, when the cyber writes a command to the physical, the cyber/physical interface will set the related actuator accordingly. The cyber/physical interface model also describes the behaviors of physical dynamic when it interacts asynchronously with cyber, i.e. when there is no D/A conversion. The cyber/physical interface is defined by modeling it using hybrid automata. There are a finite set of continuous variables whose values are described by plant models. Consider the example of TableSat. The equations of TableSat motion are: $I\dot{\omega} = 4lK_{\omega f}\nu - f_{TS}(\omega)$ and $\dot{\nu} = -\alpha\nu + K_{\nu\dot{\omega}}(V - f_{fan}(\nu))$, where I is the TableSat moment of inertia, ω is the TableSat angular velocity, ν is the speed of the fan, l is the fan moment arm, f_{TS} is the TableSat friction and is a function of ω, $K_{\nu\dot{\omega}}$ is the fan speed to force constant, V is the voltage applied to the fan, α is the fan time constant, $K_{\nu\dot{\omega}}$ is the fan voltage to change in speed constant, and f_{fan} is the frictions in the fans and are function of ν.

A configuration of Platform specify the following the running environment configuration information (hardware, operating system, A/D conversion and D/A conversion, etc.). As shown in Fig. 4, configuration information includes operating system (Debian), hardware (i386, isa devices), and A/D conversion and D/A transformation configuration (resolution and transformation of time). In addition, A/D and D/A conversion need to describe two things: 1) resolution; 2) conversion time.

A single iteration in system execution begins when the plant's state is sensed and ends after the plant evolves for one sampling period based on the controller's actions on the sensed data. Different execution conditions and different timing parameters of control tasks require different time types of cyber/physical interfaces.

Definition 4. Time of cyber/physical interface is a tuple $(\mathrm{T}, \mathrm{t}_i^k, \mathrm{t}_o^k)$, where T is a period of time, t_i^k ($k = 0,1,2,...$) and t_o^k ($k = 0,1,2,...$) refer to the A/D and D/A conversion periodical instants, respectively. The cyber/physical interface states that the inputs to the interface are sampled at t_i^k ($k = 0,1,2,...$) and the outputs are written at t_o^k ($k = 0,1,2,...$).

In the following, we formulate some popular design approaches as different types of Cyber/Physical interface. Many variations of the following described Cyber/Physical interfaces are possible. Our goal is to illustrate the concept of a Cyber/Physical interface concretely. It is briefly discussed, for each Cyber/Physical interface, how it can be derived and implemented.

InterfacePlatform{

I_p ={var= $\{u(t_i + d), y(t_i)\}$ }

I_c ={var= $\{dscadscan.sample\_values, pos\_counts\}$ }

/*cyber variable to physical input vectors mapping*/

(pos_counts, $\{u(t_i + d)\}$)

/*physical output vectors to cyber variable mapping*/

($y(t_i)$, dscadscan.sample_values)

/*A/D conversion*/

AD={Resolution, convert_time };

/*D/A conversion*/

DA={Resolution, convert_time};

/*OS version*/

OS = {Debian}

/*CPU info*/

CPU = {i386}

/*Communication */

Commu = {isa}

}

Figure 4: A Platform configuration of interface model

- **Zero Computation Time.** A Zero Computation Time (ZCT) type of cyber/physical interface is specified as a tuple (T, t_i^k, t_o^k), where $t_i^k = t_o^k = t_k$. The cyber/physical interface states that, at every instants $t_k = T \cdot k$ ($k = 0,1,2,...$), the A/D conversion to the controller are sampled, the outputs are computed and complete D/A conversion (i.e. $L_s^k = L_{io}^k = 0$, where L_s denoted as the sampling latency and L_{io} denoted as input-output latency). A typical control design process naturally results in a ZCT type and control engineers can use standard results
- **Bounded DA Conversion Time.** A Bounded DA Conversion Time (BDACT) type of cyber/physical interface is specified by a tuple (T, t_i^k, t_o^k), where $t_i^k = t_k$ and $t_o^k \in L_{io}^k$. The A/D conversion is sampled at times t_k, the D/A conversion are written at admissible variations of period. The BDACT type constitutes enforcing that the outputs are written at any point within the interval of a period, instead of precisely at same points.
- **Fixed Computation Time type.** A Fixed Execution Time (FET) type of cyber/physical interface is specified as a tuple (T, t_i^k, t_o^k), where $(t_i^k - t_o^k) > 0$ is positive number constant. This cyber/physical interface requires that the interval

from A/D conversion instant t_i to D/A conversion instant t_o is fixed.

- **Variable Computation Time.** A Variable Execution Time (VET) type of Cyber/Physical interface is specified by a tuple (T, t_i^k, t_o^k), where $t_i^k \in L_s^k$ and $t_o^k \in L_{io}^k$, and L_s^k and L_{io}^k are bounds on admissible variations of period.

Definition 5. A CPS model is a tuple $S = (S_{cps}, HA_{cps})$, where $S_{cps} = \bigcup_{k=1}^{n} S_{cyber} + \bigcup_{k=1}^{k} S_{interface} + \bigcup_{k=1}^{m} S_{physical}$ is static structure of CPS model, $HA_{cps} = TA_1 \| TA_2 \| \cdots \| TA_n \| HA_1 \| HA_2 \| \cdots \| HA_m$ is a cartesian product of automata, The set of all physical model in a CPS is denoted as $P = \{p_1, p_2, \cdots p_m\}$.

Definition 5. A state of CPS model is a tuple $s = (s_{cyber}, s_{interface}, s_{physical})$, where s_{cyber} is sa set of cyber model, $s_{physical}$ is sa set of physical model, $s_{interface}$ is sa set of interface model.

The transaction condition of CPS is denoted as $r = \varphi \cup t$, where φ is a set of events, t is a set of clock. r can be expressed either event trigger or time trigger. A trace $s_0, s_1, s_2, \cdots, s_{k-1}, s_k, \cdots, s_n$ can be denoted as, $\pi = s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} s_2 \cdots \xrightarrow{r_{k-2}} s_{k-1} \xrightarrow{r_{k-1}} s_k \cdots \xrightarrow{r_{n-1}} s_n$. From the view of the CPS system, the CPS model is consisted of a series of discrete states, and each discrete state itself may be a continuous time model.

During the symbolic execution, we only explore finite traces. In this case, however, the observed finite traces are not necessarily proper prefixes of the original program traces, and our approach can produce false results, as the symbolic execution can continue past unsatisfied loop termination conditions. We use the infinite extension semantics to resolve ugly prefixes into presumably good or presumably bad. We characterize the truth value in $\mathbf{B}_4$ of a LTL formula φ with respect to a single finite trace s.

Lemma 1

1. $[S \vDash \varphi]_B = \mathrm{T}$ iff $\nexists s \in S, \omega \in \Sigma^{\omega}, [s\omega \vDash \neg\varphi]_{\omega} = \mathrm{T}$;
2. $[S \vDash \varphi]_B \supseteq \mathrm{T}^P$ iff $\nexists s \in S, [ss_{n-1}^{\omega} \vDash \neg\varphi]_{\omega} = \mathrm{T}$;
3. $[S \vDash \varphi]_B = \perp$ iff $\exists s \in S, \forall \omega \in \Sigma^{\omega}, [s\omega \vDash \neg\varphi]_{\omega} = \mathrm{T}$;
4. $[S \vDash \varphi]_B \supseteq \perp^P$ iff $\exists s \in S, [ss_{n-1}^{\omega} \vDash \neg\varphi]_{\omega} = \mathrm{T}$;

Proof. (1) Since $\nexists s \in S, \omega \in \Sigma^{\omega}, [s\omega \vDash \neg\varphi]_{\omega} = \mathrm{T}$ is equivalent to $\forall s \in S, [s\omega \vDash \neg\varphi]_{\omega} = \perp$ and $\forall s \in S, [s\omega \vDash \varphi]_{\omega} = \mathrm{T}$, thus by Definition 2,

$\left[s \vDash \varphi\right]_B = \mathrm{T}$; therefore, $\bigcap_{s\in S}\left[s \vDash \varphi\right]_B = \mathrm{T} \Rightarrow \left[S \vDash \varphi\right]_B = \mathrm{T}$;

(2) Since $\nexists s \in S, \left[ss_{n-1}^{\omega} \vDash \neg\varphi\right]_\omega = \mathrm{T}$ is equivalent to $\forall s \in S, \left[ss_{n-1}^{\omega} \vDash \varphi\right]_\omega = \mathrm{T}$, thus by Definition 2, $\forall s \in S, \left[ss_{n-1}^{\omega} \vDash \varphi\right]_B = \mathrm{T} \supseteq \left\{\left[ss_{n-1}^{\omega} \vDash \varphi\right]_\omega = \mathrm{T} \wedge \exists \omega \in \Sigma^{\omega}\right\}$, therefore, $\bigcap_{s\in S}\left[s \vDash \varphi\right]_B = \mathrm{T}^P \Rightarrow \left[S \vDash \varphi\right]_B = \mathrm{T}^P$;

(3) Since $\left[s\omega \vDash \neg\varphi\right]_\omega = \mathrm{T}$ is equivalent to $\left[s\omega \vDash \varphi\right]_\omega = \perp$, by Definition 2, $\exists s \in S, \forall \omega \in \Sigma^{\omega}, \left[s\omega \vDash \varphi\right]_\omega = \perp \Rightarrow \left[s\omega \vDash \varphi\right]_B = \perp$, therefore, $\left[S \vDash \varphi\right]_B = \perp$;

(4) Similarly, since $\left[ss_{n-1}^{\omega} \vDash \neg\varphi\right]_\omega = \mathrm{T}$ is equivalent to $\left[ss_{n-1}^{\omega} \vDash \varphi\right]_\omega = \perp$, by Definition 2, $\left[ss_{n-1}^{\omega} \vDash \varphi\right]_\omega = \perp \supseteq \left\{\left[uu_{n-1}^{\omega} \vDash \varphi\right]_\omega = \perp \wedge \exists \omega \in \Sigma^{\omega}, \left[u\omega_{n-1}^{\omega} \vDash \varphi\right]_\omega\right\}$, therefore, $\left[S \vDash \varphi\right]_B \supseteq \perp^P$.

6 Evaluation

6.1 Co-simulation

In this section, we improve on the simulation tool [Yu, Fei, Dong et al. (2013)] that we previously built by improving its shortcomings to provide different time types of cyber/physical interface for high-assurance CPS.

As shown in Fig. 5, a co-simulation environment is developed for TableSat. An X86 processor model is utilized to emulate the Athena II SBC in QEMU. The embedded control program is written in C language the plant components are modeled mathematically according to respective physical characteristics in Matlab/Simulink.

We conducted this experiment with different time configuration of Cyber/Physical interface. We set the step input of expected angular velocity with 30 deg/sec. Experimental datasets were used to compare accuracy of these time types of Cyber/Physical interface. For the zero computation time of Cyber/Physical interface, we set the fixed sampling interval T=0.4 s in the virtual TableSat. The experimental results are shown in Fig. 6. For the fixed computation time of Cyber/Physical interface, we set the fixed sampling interval T=0.4 s and the fixed interval from A/D conversion to D/A conversion d=0.2 s. The experimental results are shown in Fig. 7. For the bounded DA conversion time, we set the fixed sampling interval T=0.4 s and the input-output jitter $\mathrm{t}_o^k \in [0, 0.1]$ in the virtual TableSat. The experimental results are shown in Fig. 8. For the variable computation time, we set the fixed sampling interval T=0.4 s the sampling jitter $\mathrm{t}_i^k \in [0, 0.1]$and the input-output jitter $\mathrm{t}_o^k \in [0, 0.1]$. The experimental results are shown in Fig. 9.

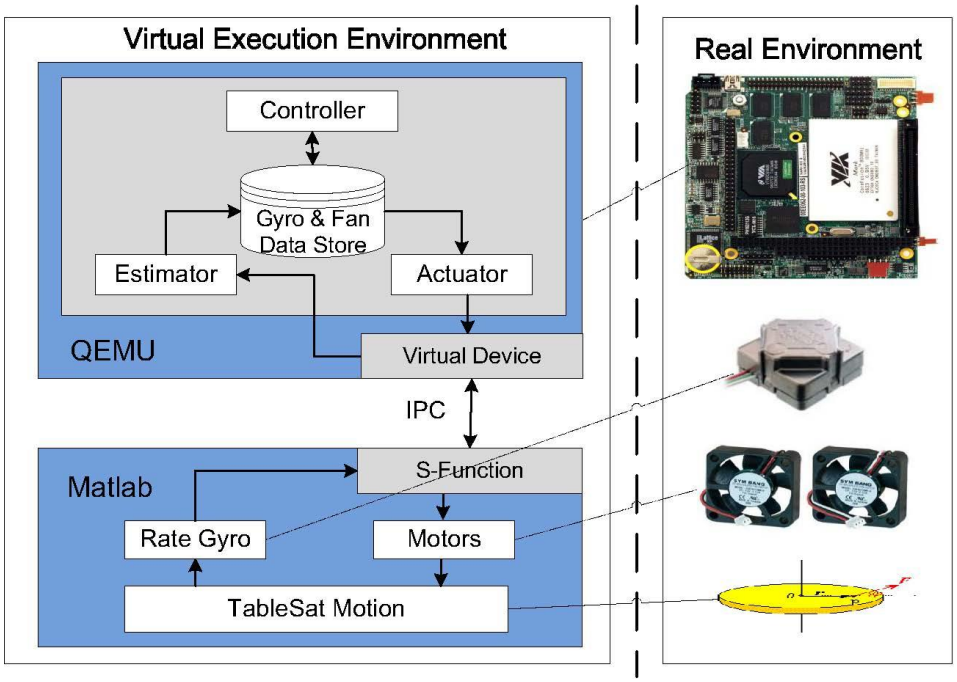

Figure 5: Co-simulation environment for TableSat

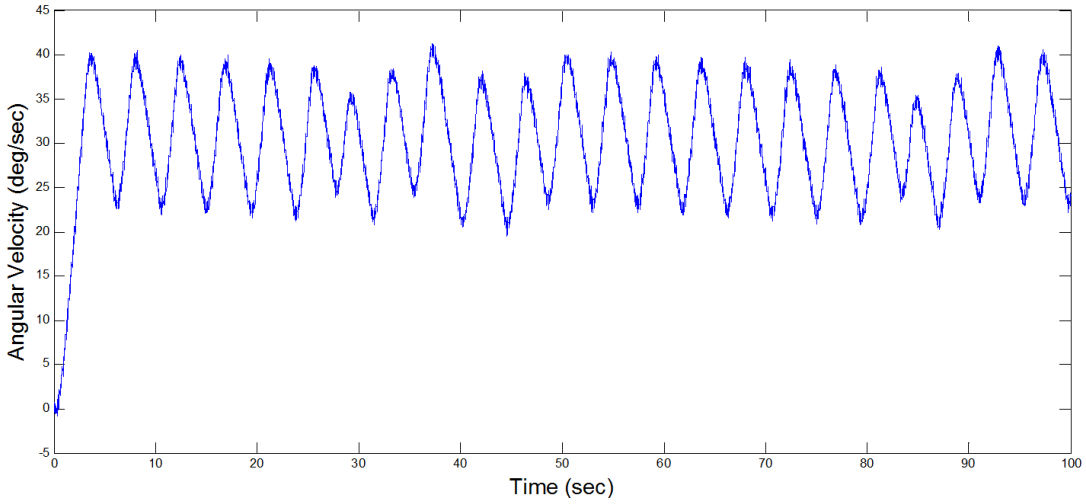

Figure 6: The experimental results of zero computation time

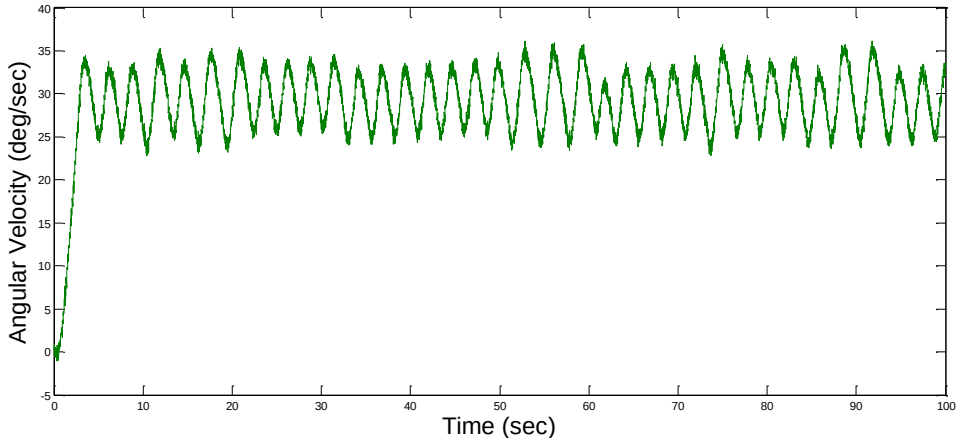

Figure 7: The experimental results of fixed computation time

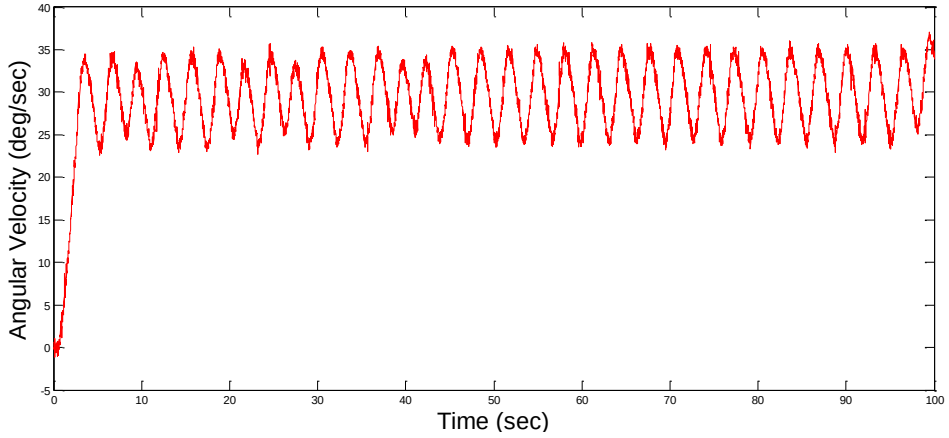

Figure 8: The experimental results of bounded DA conversion time

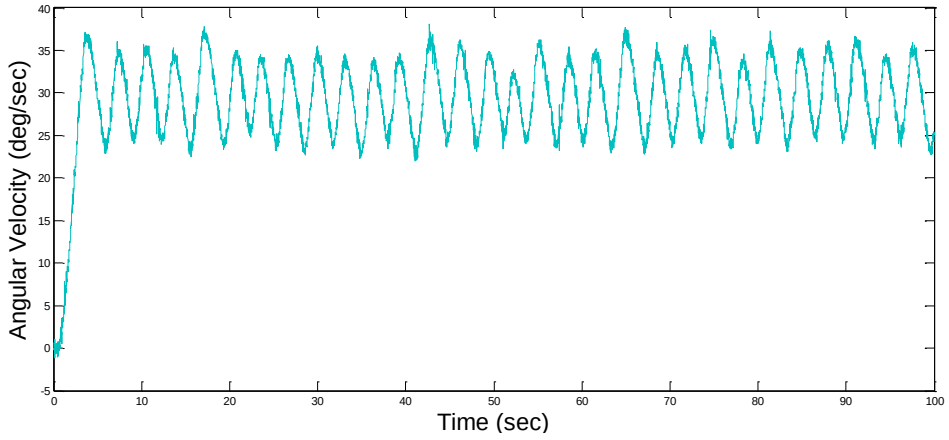

Figure 9: The experimental results of variable computation time

The experiment results show that the controller can meet the requirements of system rapidity and control accuracy, satisfy the Bounded Input Bounded Output (BIBO) stability. In order to quantify the divergence between the real environment and co-simulation environment, we define the absolute divergence. This evaluation metric is the difference between the actual velocity and virtual velocity in different time of Cyber/Physical interface, i.e. $e = v_{actual} - v_{virtual}$. Tab. 1 shows comparisons statistics of absolute divergence over eight runs. Each column in the table shows statistics of a system run with different time of Cyber/Physial interface. We recorded the angular velocity at every 0.5 s. Through comparing three experiments, the results indicate divergence between the real TableSat and its virtualization reduces sequentially, which shows the type of Cyber/Physical interface could improve the accuracy. The average absolute divergence over all time instant is relatively low and below 1.772 deg/sec. All the maximum absolute divergence values occur in the first two 2 s.

Table 1: Summary of absolute divergence

Test	Statistics	Interface Type			
		zero computation time	fixed computation time	bounded DA conversion time	variable computation time
Run 30	max	18.02	16.38	16.19	16.01
	min	-4.591	-2.702	-2.733	-2.609
	mean	1.772	1.429	1.064	1.26
	std	3.697	2.514	2.269	2.377

The experiment shows that our approach can simulate the real system with reasonable accuracy. This can enable early development and verification of the synergy between cyber and physical components

6.2 Co-verification

To evaluate the proposed approach, we have applied the approach to real-world control systems. In all experiments, we want to check whether the system meet these constrains or not with slight perturbations in the inputs and outputs to the system.

6.2.1 TableSat co-verification

We use the same embedded control program as in co-simulation. First, we constructed the program (as shown in Fig. 10) and physical model based on the cyber/physical interface. Then we formulated these constrains of the system with LTL, and conducted bounded model checking. We chose the fixed computation time type of cyber/physical interface in this experiment. We set the following initial set of parameters in the experiment: the sampling interval is 2 s, the A/D conversion instant is 0.4 s, the D/A conversion instant is 1.6 s and the target rotary velocity is 30 deg/s. the initial value of angular velocity is used as a symbolic variable ([0, 40]). Tab. 2 summarized the results. The verification result shows that the TableSat satisfies the last two LTL constrains. For

the first LTL property, bounded model checker pointed out a simple bug of the cyber component that: If the initial value of angular velocity is 39.960621 deg/s, then the rotary velocity will reach 63.649414 deg/s at 2.324336 s, which led to above a threshold (60 deg/s) The running time largely depends on the backend SMT solver.

6.2.2 Thermostat co-verification

The second experiment is the thermostat system Thermostat is a typical CPS system which utilize the temperature controller to ensure a particular space for expectations of intelligent system. In thermostat system, the environment temperature is physical process which is continuous change, and the controller is discrete cyber process: when the controller detects the temperature is higher than the preset temperature, cut off the heater power. When the test temperature is lower than the preset temperature, restart the heater to heat environment. thermostat program (line number is: 45) is shown in Fig. 11. When the temperature drops below to 19°C, control software gives control instruction on it, so as to open the heater; And when the system temperature is higher than 21°C, on the contrary, sends out control instructions off control applications, thus closing the heater. Automatic temperature control system to ensure that the environment temperature is between 18°C and 22°C.

Table 2: Design constraints for TableSat

No.	LTL Constraint	Result
1	$\mathrm{G}(Rotary.Velocity \leq VelocityUpBound)$: the controller never accelerates the TableSat over the rotary velocity limit VelocityUpBound.	$\perp$
2	$\mathrm{G}((Rotary.Velocity > 1.5 \times TargetVelocity) \rightarrow (Actuator.FanVoltage \equiv 0))$: When the rotary velocity below 1.5 times of its expected value, the controller will set the fans to 12 volts.	T^p
3	$\mathrm{G}((Rotary.Velocity > TargetVelocity) \rightarrow F(Actuator.FanVoltage = fullNeg))$: after the Rotary velocity below its bound, controller will set the motors to the full voltage.	T^p

```
01 int main(int argc, char **argv, char **envp){
02   ...
03   initial();
04   while(TSrunning){
05     ...
06     waitClock();
07     /* Read the raw sensor values */
08     if (!readSensors(&SensorReadings)) {
09         printf("A/D error in ReadSensors\n");
10         return;
11     }
12     calculateOutput();
13    /* Actually fire off the motors */
14     commandMotor(Actuator.FanVoltage);
15     updateState();
16   }
17   Return;
18 }
19 int ReadSensors(SensorReadings_t *sensors)
20 {  ...
21   /*associate with read sensor event $E_{state}$*/
22   if((result = dscADScan(dscb, &dscadscan, samples))!= DE_NONE){
23    ...
24   }
25 }
26 void commandMotor(double *v)
27 {  ...
28   /*associate with write command event $E_{comm}$*/
29   if((result = dscDAConvert(dscb, pos_channel, pos_counts)) != DE_NONE){
30     ...
31   }
32 }
```

Figure 10: TableSat program

```
01 int main(int argc, char **argv, char **envp)
02 {
03     ...
04     while(1){
05         ...
06         sensor_data = AD_Conversion();
07         if(sensor_data < 19){
08             v_command = 1;
09         }
10         if(sensor_data > 21){
11             v_command = 0;
12         }
13         DA_Conversion(v_command);
14         ...
15     }
16    ...
17 }
```

Figure 11: Thermostat program

We chose a zero computation time of cyber/physical interface in this experiment. The following initial set is used during this experiment: {T=0.3 s, t_i =0 s, t_o =0 s}. In control theory, control engineers always assume that A/D conversion periodically and D/A conversion instantaneously at the beginning of each period.

Fig. 12 shows the cyber/physical interface model of thermostat by hybrid automata. The model consists of 4 discrete locations corresponding to each node, 3-dimensional continuous states $\tilde{x} = \{x, t\}$, and 6 discrete state transitions corresponding to the edges. Let t represent the internal timer. There are 4 discrete locations in the interface model (Turn On: On_AD and On_DA; Turn Off: Off_AD and Off_DA). Each discrete transition is enabled by its guard condition. For example, a discrete transition d from On_AD to Off_DA has a guard condition t>T ∩ On. When the controller sends command (i.e. when Boolean variable, On or Off, is set to true), the motion of thermostat switches to the corresponding law. An edge entering Off AD represents the initial constraint.

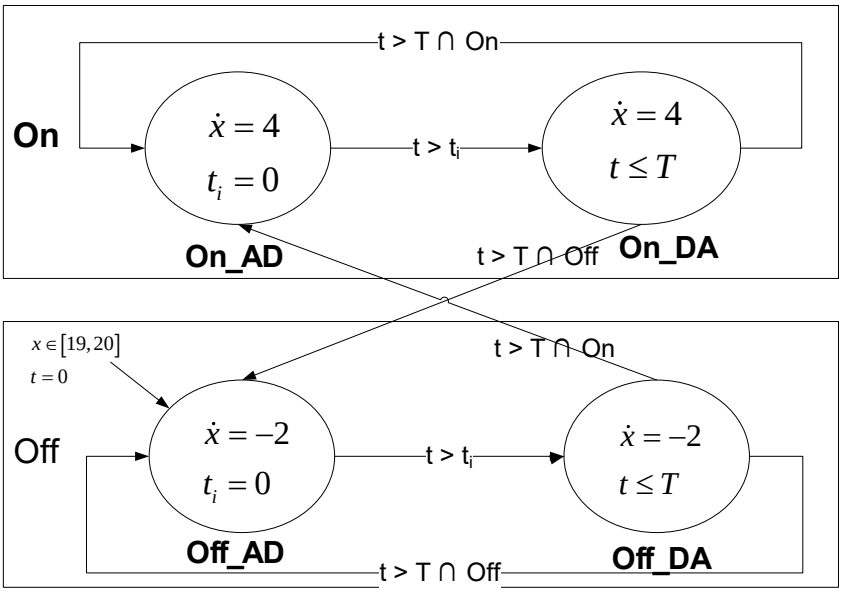

Figure 12: Cyber/physical interface of Thermostat

We applied our co-verification approach to the thermostat system with the same process as TableSat. As shown in Tab. 3, the system satisfies the last three constraints. Bounded model checking thus revealed a simple bug of the controller that was, however, subtle enough not to be detected when designing the model: when the room temperature near the temperature limit (22°C), instead of applying the off, the program still turns the heater on, allowing the temperature to exceed the temperature limit. This happens since the program re-computes the thermostat setting only every 0.3 s.

7 Related work

Many scholars have done many work and gained their research results on cyber system and physical system verification respectively. And considerable effort and tools have been put into figuring out how to verified these two separate systems. In physical systems research, they focus on physical system and tend to model cyber system as a equipment which strictly implement control algorithm based on the assumptions, such as network latency, sampling time, etc. And these assumptions are just a few exceptions (like the worst-case execution time), which is difficult to meet. In cyber systems research, they improve the level of abstraction and specify characteristics and demand of physical environment as non-functional properties. This leads to lack of attention on the cyber/physical multiple semantic interaction.

Various formal verification methods have been proposed for specifying hybrid systems [Chan, Ricketts, Lerner et al. (2016); Kaur and Kaur (2017); Bersani and Garcia-Valls (2016); Cimatti, Mover and Tonetta (2012)]. Well-known tools for verifying hybrid systems include HyTech [Henzinger, Ho and WongToi (1997)] and Uppaal [Larsen, Pettersson and Yi (1997)]. There has been much research on abstracting hybrid systems,

largely categorized into sufficient abstraction and equivalent abstraction (surveyed in [Alur, Henzinger, Lafferriere et al. (2000))]. In Goubault et al. [Goubault, Putot, Baufreton et al. (2008)], they applied affine arithmetic to reason about the precision of floating point C program. In Herrmann et al. [Herrmann, Blech, Han et al. (2016)] and Shan et al. [Shan, Zhou, Wang et al. (2015)] they propose an approach to formally analyzing such control software using model checking of UPPAAL. In Eggers et al. [Eggers, Ramdani, Nedialkov et al. (2011)], they used an interval-based SMT solver for ODEs. In Bae et al. [Bae, Ölveczky, Kong et al. (2016)], they proved that the decision problem for bounded logic formulas over the real numbers with general nonlinear functions are decidable.

Table 3: Design constraints for thermostat system

No.	LTL Constraint	Result
1	$\mathbf{G}\ (Temper \le TemperUpLimit)$: the thermostat controller will never heat over the temperature limit.	$\perp$
2	$\mathbf{G}\,(Temper \ge TemperDownLimit)$: the thermostat controller will never heat below the temperature limit.	T^p
3	$\mathbf{G}\ (\ (Temper \ge up\_th) \rightarrow \mathbf{F}\ (Off)\ )$: after the temperature is above the up_th, then the controller will be sent Off command.	T^p
4	$\mathbf{G}\ (\ (Temper < down\_th) \rightarrow \mathbf{F}\ (On)\ )$: after the temperature is lower than down_th, then the controller will be sent On command.	T^p

Due to the scalability of formal verification is not high, simulation is a low-cost and efficient method in detecting shallow bugs. There has been much research [Eker, Janneck, Lee et al. (2003); Hoffmann, Kogel and Meyr (2001); Semeria and Ghosh (2000); Passerone, Lavagno and Chiodo (1997); Cong, Lei, Yang et al. (2015)] on co-simulation that has led to industrial tools such as Matlab/Simulink, Mathematica and Modelica. In Mueller et al. [Mueller, Becker, Elfeky et al. (2012)], they proposed a methodology and toolset for the CPS virtual prototyping. In Al-Hammouri [Al-Hammouri (2012)], they presented a comprehensive co-simulation platform for CPS, which is built on Modelica and ns-2 tools. In Zhenkai et al. [Zhenkai, Emeka, Xenofon et al. (2014)], a CPS co-simulation method based on time trigger was proposed, which integrate SystemC and CarSim. In Davide et al. [Davide, Riccardo, Roberto et al. (2012)], they presented a co-simulation tool which integrate SystemC/SCNSL with MATLAB/Simulink. These methods combine different simulation tools for CPS co-simulation, however, they did not consider the different types of interaction between cyber component and physical component.

8 Conclusions

An approach has been presented to componentized the interface and abstracts the interaction by Cyber/Physical interface components. We classify the interaction

mechanism into two levels: logic interaction level and physical interaction level. We designed a co-verification interface model to capture the interaction between computation and physical processes for hierarchical a verification of CPS. We define different types of interface model according to combinatorial relationships of the A/D (Analog to Digital) and D/A (Digital to Analog) conversion periodical instants. We advocate the use of Cyber/Physical interface model for bridging multiple semantic gap between the two domains. This interface model has formal semantics, and is efficient for simulation and formal verification. Thirdly, an approach is presented to Cyber/Physical co-verification using co-simulation in physical level and formal co-verification in logic level. The approach is illustrated through realistic examples. The evaluation has demonstrated the effectiveness of this approach. Our research to develop better abstraction/refinement to reduce verification complexity associated with certain algorithms is ongoing.

Acknowledgement: This research received financial support from Natural Science Foundation of Hainan province (Grant Nos. 617062, 2018CXTD333 and 617048), the National Natural Science Foundation of China (Grant Nos. 61462022, 61762033 and 61662019), Major Science and Technology Project of Hainan province (Grant No. ZDKJ2016015), Scientific Research Staring Foundation of Hainan University (Grant No. kyqd1610).

References

Al-Hammouri, A. T. (2012): A comprehensive co-simulation platform for cyber-physical systems. *Computer Communications*, vol. 36, no. 1, pp. 8-19.

Alur, R.; Henzinger, T. A.; Lafferriere, G.; Pappas, G. J. (2000): Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971-984.

Årzén, K. E.; Cervin, A.; Henriksson, D. (2005): *Implementation-Aware Embedded Control Systems.* Birkhäuser Boston, USA.

Bae, K.; Ölveczky, P. C.; Kong, S.; Gao, S.; Clarke, E. M. (2016): Smt-based analysis of virtually synchronous distributed hybrid systems. *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pp. 145-154.

Bersani, M. M.; Garcia-Valls, M. (2016): The cost of formal verification in adaptive cps. an example of a virtualized server node. *Proceedings of the 2016 IEEE 17th International Symposium on High Assurance Systems Engineering*, pp. 39-46.

Chan, M.; Ricketts, D.; Lerner, S.; Malecha, G. (2016): Formal verification of stability properties of cyber-physical systems. *Proceedings of the CoqPL'16*, pp. 39-40.

Cimatti, A.; Mover, S.; Tonetta, S. (2012): Smt-based verification of hybrid systems. *Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 5072-5077.

Clarke, E.; Kroening, D.; Lerda, F. (2004): A tool for checking ansi-c programs. *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 168-176.

Cong, K.; Lei, L.; Yang, Z.; Xie, F. (2015): Automatic fault injection for driver robustness testing. *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pp. 361-372.

Davide, Q.; Riccardo, M.; Roberto, B.; Paolo, F. (2012)**:** A SystemC/Matlab co-simulation tool for networked control systems. *Simulation Modelling Practice and Theory*, vol. 23, pp. 71-86.

Eggers, A.; Ramdani, N.; Nedialkov, N.; Fränzle, M. (2011): Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In: Barthe, G.; Pardo, A.; Schneider, G. (Eds.): *Software Engineering and Formal Methods*, pp. 172-187, Springer Berlin Heidelberg.

Eker, J.; Janneck, J. W.; Lee, E. A.; Liu, J.; Liu, X. et al. (2003): Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127-144.

Gastin, P.; Oddoux, D. (2001): Fast LTL to Büchi automata translation. In Berry, G.; Comon, H.; Finkel, A. (Eds.): *Computer Aided Verification*, pp. 53-65, Springer Berlin Heidelberg.

Goubault, E.; Putot, S.; Baufreton, P.; Gassino, J. (2008): Static analysis of the accuracy in control systems: principles and experiments. In Leue, S.; Merino, P. (Eds.): *Formal Methods for Industrial Critical Systems*, pp. 3-20, Springer Berlin Heidelberg.

Henzinger, T.; Ho, P.; Wong-Toi, H. (1997): *Hytech:* a model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1, pp. 110-122.

Herrmann, P.; Blech, J. O.; Han, F.; Schmidt, H. (2016): A model-based toolchain to verify spatial behavior of cyber-physical systems. *International Journal of Web Services Research*, vol. 13, no. 1, pp. 40-52.

Hoffmann, A.; Kogel, T.; Meyr, H. (2001): A framework for fast hardware-software co-simulation. *Proceedings of 2001 Design, Automation and Test in Europe*, pp. 760-764.

Kaur, J.; Kaur, K. (2017): A fuzzy approach for an iot-based automated employee performance appraisal. *Computers Materials & Continua*, vol. 53, no. 1, pp. 23-36.

Larsen, K.; Pettersson, P.; Yi, W. (1997): Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1, pp. 134-152.

Lee, E. A. (2010): Cps foundations. *Proceedings of the 2010 Design Automation Conference*, pp. 737-742.

Mueller, W.; Becker, M.; Elfeky, A.; DiPasquale, A. (2012): Virtual prototyping of cyber-physical systems. *17th Asia and South Pacific Design Automation Conference*, pp. 219-226.

Passerone, C.; Lavagno, L.; Chiodo, M. (1997): Fast hardware/software co-simulation for virtual prototyping and trade-off analysis. *Proceedings of the 34th annual Design Automation Conference*, pp. 389-394.

Semeria, L.; Ghosh, A. (2000): Methodology for hardware/software co-verification in c/c++. *Proceedings of 2000 ASP-DAC Conference*, pp. 405-408.

Shan, L. J.; Zhou, X. S.; Wang, Y. Y.; Zhao, L.; Wan, L. J. et al. (2015): Statistical model checking of cyber-physical systems control software. *Journal of Software*, vol. 26, no. 2, pp. 380-389.

Vess, M. F. (2005): *System Modeling and Controller Design for a Single Degree of Freedom Spacecraft Simulator (Ph.D. Thesis)*. University of Maryland.

Yu, Z.; Fei, X.; Dong, Y. W.; Yang, G.; Zhou, X. (2013): High fidelity virtualization of cyber-physical systems. *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 4, no. 2.

Yu, Z.; Dong, Y. W.; Fei, X. (2014): Bounded model checking of hybrid automata pushdown system. *Proceedings of the 14th International Conference on Quality Software*, pp. 190-195.

Zhenkai, Z.; Emeka, E.; Xenofon, K.; Joseph, P.; Gabor, K. et al. (2014)**:** A co-simulation framework for design of time-triggered automotive cyber physical systems, *Simulation Modelling Practice and Theory*, vol. 43, pp. 16-33.