Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 470139, 12 pages
http://dx.doi.org/10.1155/2013/470139

Research Article

Formal Modeling and Verification for MVB

Mo Xia, Kueiming Lo, Shuangjia Shao, and Mian Sun

School of Software, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China

Correspondence should be addressed to Mo Xia; tabris17th@gmail.com

Received 7 February 2013; Accepted 19 March 2013

Academic Editor: Xiaoyu Song

Multifunction Vehicle Bus (MVB) is a critical component in the Train Communication Network (TCN), which is widely used in most of the modern train techniques of the transportation system. How to ensure security of MVB has become an important issue. Traditional testing could not ensure the system correctness. The MVB system modeling and verification are concerned in this paper. Petri Net and model checking methods are used to verify the MVB system. A Hierarchy Colored Petri Net (HCPN) approach is presented to model and simulate the Master Transfer protocol of MVB. Synchronous and asynchronous methods are proposed to describe the entities and communication environment. Automata model of the Master Transfer protocol is designed. Based on our model checking platform M^3C, the Master Transfer protocol of the MVB is verified and some system logic critical errors are found. Experimental results show the efficiency of our methods.

1. Introduction

Multifunction Vehicle Bus (MVB) is a crucial component in the Train Communication Network (TCN) which is widely used in most of the modern train control techniques of the transportation system. How to ensure security of MVB has become an important issue.

The traditional method to verify MVB usually uses simulation technique or testing approach. A simulation tool based on the model of Slave devices and a Master Frames Generator is presented in [1]. It accomplishes the verification flow for interchanging message data among MVB devices. Using an error-tolerance decode algorithm, the transmitted data of the Bus are sampled in [2], where the compatibility and dependability of data transmission among devices are analyzed via protocol analysis application.

Model checking is a method for automatically verifying finite state systems [3, 4]. The procedure uses an exhaustive search of the state space of a system model to determine whether a specification is satisfied or violated. Although model checking has been applied in many fields, such as circuit and critical software verification in [5–7], it is hardly used for verification of the MVB or TCN. Hierarchy Colored Petri Net (HCPN) is a behavioral modeling language [8]. It can be used for modeling the validation of concurrent systems. This paper presents a modeling approach for MVB based on HCPN. Two modeling approaches based on model checking method for the MVB are given in this paper too.

The remainder of this paper is organized as follows. The following section gives the details of the MVB and Master Transfer. Section 3 proposes a modeling method with HCPN and shows the modeling of the Master Transfer. Section 4 shows two modeling methods based on model checking and also explains how to model the Master Transfer. Section 5 introduces the tools we use for verification and reports the experimental results. Section 6 concludes the paper.

2. MVB and Master Transfer

In this section, first we introduce the MVB and then present a protocol Master Transfer of MVB and its token passing algorithm.

2.1. MVB. The on-board train communication system has been widely demanded for modern railways [9, 10]. The MVB is a component of the TCN which is used in most of the modern train control systems. The TCN has been defined by the IEC (International Electrotechnical Commission) [11, 12]; it is the Vehicle Bus specified to connect standard equipment.

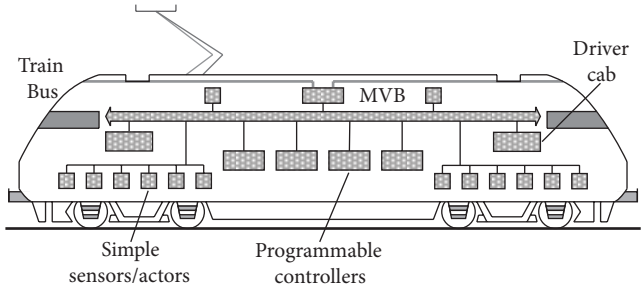

Figure 1: Multifunction Vehicle Bus in a locomotive.

It provides both the interconnection of programmable equipment pieces amongst themselves and the interconnection of this equipment with its sensors and actors. It can also be used as a Train Bus in trains which are not separated during normal operation. An application of the MVB in a locomotive is shown in Figure 1.

The MVB defines two types of devices: Master and Slave. Each Vehicle Bus and Train Bus has one Master node and several Slave ones. The Master sends information, the Master_Frame, to a number of Slave devices. The Slave receives information from the Bus and sends information, the Slave_Frame, in response to the Master. A Master_Frame and the corresponding Slave_Frame form a telegram, which is shown in Figure 2. All devices decode the Master_Frame. The addressed source device then replies with its Slave_Frame, which may be received by several other devices.

2.2. Master Transfer. Since a single Master presents a single point of failure, mastership may be assumed by several Bus Administrators (BAs), one at a time. To increase availability, mastership can be shared by two or more BAs, which both exercise mastership for the duration of a turn. Mastership is transferred from BusAdmin to BusAdmin within a few milliseconds in case of failure. To exercise redundancy, mastership is transferred every few seconds by a token frame. To this effect, all BAs are organized in a logical ring as shown in Figure 3. A token passing mechanism ensures that only one BusAdmin becomes Master.

In the IEC 61375-1 international standard [11], Mastership Transfer states is depicted in a SDL graph as shown in Figure 4.

Mastership Transfer describes the protocol which selects a Master from one of several BAs and ensures Mastership Transfer at the end of a turn or upon the occurrence of a failure. A token passing algorithm is defined in the IEC standard [11] to ensure a round-robin access of all BAs to the Bus:

(i) after the loss of the Master, staggering of the time-outs ensures that only one of the BAs becomes Master;

(ii) a Master exercises mastership for the duration of one turn;

(iii) After its turn, the Master looks for the next BusAdmin and reads its Device Status, which indicates if this device is a configured BusAdmin;

(iv) a Master may only pass mastership to a configured and actualized BusAdmin;

(v) if the device is not a configured and actualized BusAdmin, the Master looks for the next BusAdmin after the next turn;

(vi) if the device is a configured and actualized BuA, the Master offers mastership to it by sending a Mastership Transfer Request;

(vii) if the device accepts mastership in its Mastership Transfer Response, or if no answer comes, the Master retires to become a standby Master and monitors the Bus traffic for mastership offer or Bus silence;

(viii) if the other device rejects mastership, the current Master retains mastership for one more turn, after which the Master tries the next device in its BAs list;

(ix) a standby BusAdmin becomes Master if it accepts a Mastership Transfer Request or if it detects no Bus activity during a time greater than a defined time-out.

LTL (Linear-time Temporal Logic) is a temporal logic, it is suitable for presentation of some temporal properties. The definition of LTL is given as follows.

Definition 1 (Linear Temporal Logic). Linear Temporal Logic (LTL [13]) has the following syntax given in Backus Naur form:

$$\begin{aligned}\phi ::= {} & \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \longrightarrow \phi) \\ & \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi U \phi) \mid (\phi W \phi) \mid (\phi R \phi),\end{aligned} \tag{1}$$

where p is any propositional atom from some set of atoms. X means next state, F means some future state, and G means all future states. The next three, U, R, and W, are called Until, Release, and Weak-until, respectively.

According to the requirement specified in the standard, the Mastership Transfer must satisfy the following properties (suppose that there are n BAs altogether):

(1) there cannot be more than one Master at one time; it is written in the LTL as shown in

$$\mathrm{G}\neg\left(\mathrm{BA\_Master}_i \wedge \mathrm{BA\_Master}_j\right), \quad i, j = 1 \cdots n,\ i \neq j, \tag{2}$$

(2) there cannot be no Master at one time; it is written in the LTL as shown in

$$\mathrm{G}\neg\left(\bigwedge_{i=1}^{n} \mathrm{BA\_Standby}_i\right). \tag{3}$$

3. HCPN Modeling

3.1. HCPN. The Colored Petri Net (CPN [14]) preserves useful properties of Petri Nets and it has better structuring facilities such as types and modules. The CPN can be defined as follows.

Definition 2 (Colored Petri Net). A Colored Petri Net CPN is a tuple $(P, T, F, \Sigma, C, E, G, I)$, where

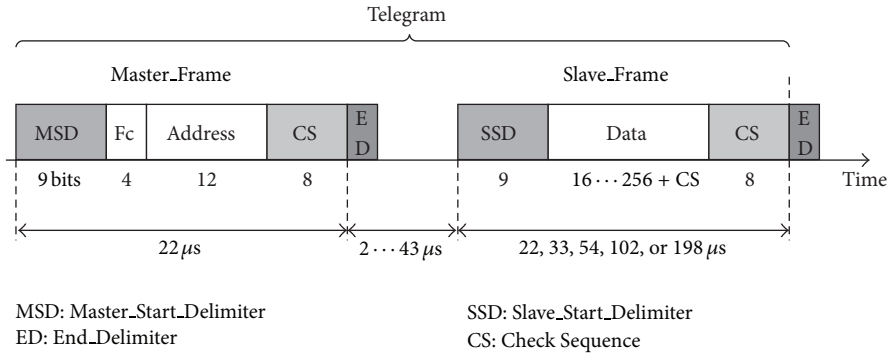

Figure 2: Telegram of MVB.

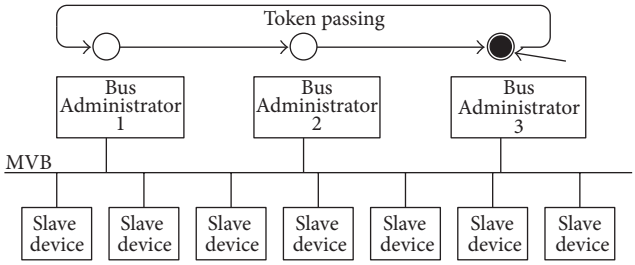

Figure 3: Mastership Transfer with multiple Masters.

Table 1: Color sets.

Color set	Declaration	Description
FID	colset FID = with MTRQ|MTRP|DSRQ|DSRP|TM	Signal types
ADDR	colset ADDR = int with $0\cdots1$	Device address
DATA	colset DATA = int with $0\cdots2$	Frame data
FRAME	colset FRAME = product FID × ADDR × DATA	Frame
TINT	colset TINT = int timed	Timed int type
TBOOL	colset TBOOL = bool timed	Timed bool type

(1) P is a finite set of places;

(2) T is a finite set of transitions, and $P \cap T = \varnothing$, $P \cup T \neq \varnothing$;

(3) $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs;

(4) Σ is a set of color sets defined within the CPN model; this set contains all possible colors, operations, and functions used within CPN;

(5) $C : P \rightarrow \Sigma$ is a color function; it maps places into colors in Σ;

(6) $E : F \rightarrow \Sigma$ is an arc expression function; it maps each arc $f \in F$ into an expression $e \in \Sigma$;

(7) $G : T \rightarrow \Sigma$ is a guard function; it maps each transition $t \in T$ into a guard expression $g \in \Sigma$;

(8) I is an initialization function.

The HCPN extends CPN, and it can simplify the structure of CPN and is easier for modification. The definition of HCPN is given as follows.

Definition 3 (Hierarchy Colored Petri Net). Hierarchy Colored Petri Net is extended from CPN, which adds two types of transitions and places. An HCPN is a tuple (N, IO, S), where

(1) N is a Colored Petri Net;

(2) $IO \subseteq P$, where for all $i \in IO$, i is attached with a type of IN or OUT;

(3) $S \subseteq T$, where for all $s \in S$, s is a substation transition, which is substituted by a page.

The distinction between tokens can strongly express data transferred in a communication channel. In the following part, we use HCPN to model all the components shown in Figure 4.

3.2. Color Sets. In the CPN model, color sets are used to distinguish tokens. There are six kinds of color sets defined in our HCPN model, whose name, declaration, and description are demonstrated in Table 1. Color set FID is used to present signal types, with any value in the finite set MTRQ, MTRP, DSRQ, TM. And color set FRAME is a compound color set, that is, an ordered triple, that stands for the frame transferred in the communication channel.

3.3. Translation Rules. The Mastership Transfer state shown in Figure 4 is translated into a CPN model according to the rules as follows:

(1) use a place element to represent a state, such as STANDBY_MASTER and FIND_NEXT;

(2) use a place element to represent a process, ignoring the execution time of the process;

(3) use a transition element to represent a task;

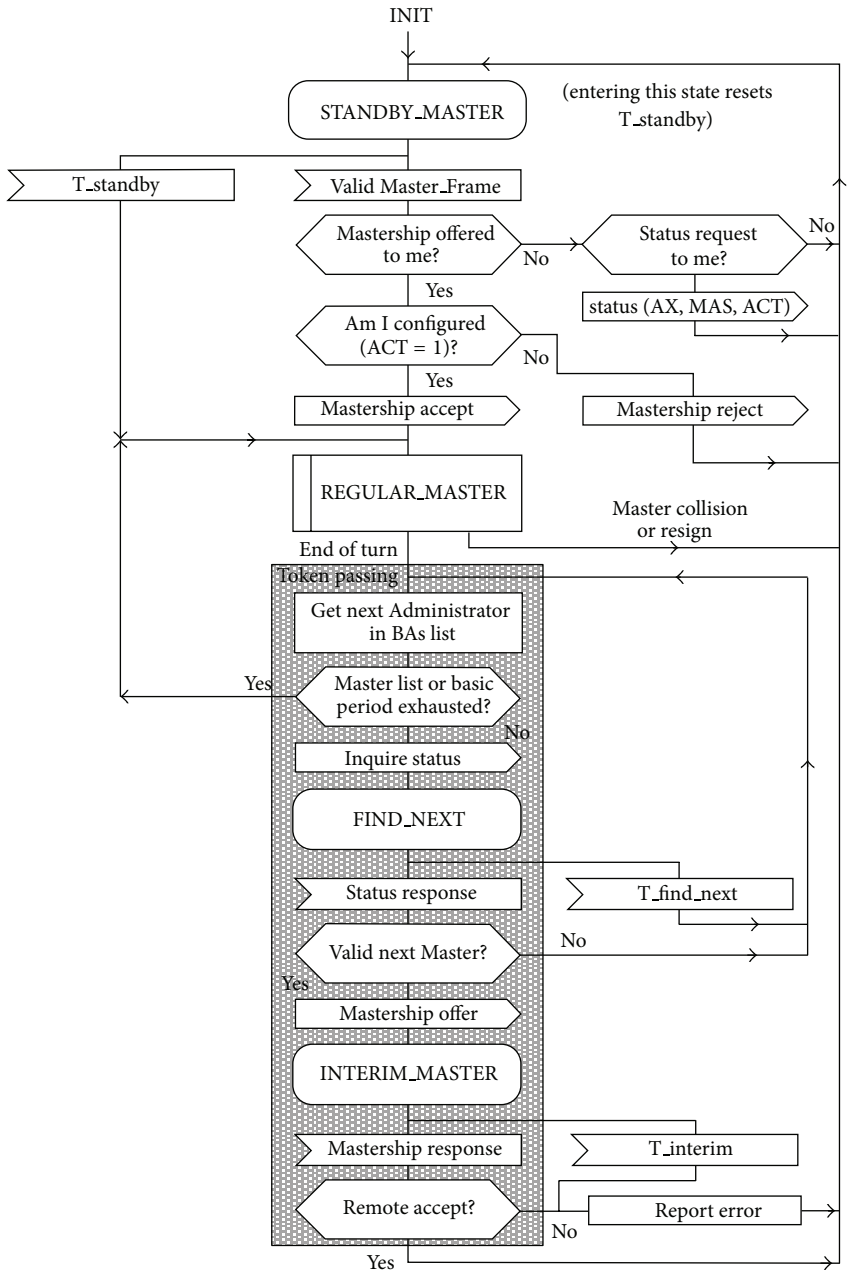

Figure 4: Mastership Transfer states.

(4) use an arc expression to represent a decision;

(5) input and output signals are divided into three types: Mastership Transfer telegram, Device Status telegram, and Time-out signals. These signals are distinguished by the FID field as shown in Table 2.

3.4. HCPN Model. The HCPN model of the Mastership Transfer consists of four modules (assume there are two BAs), and the details of them are given as follows.

3.4.1. Top Module. Assume there are two BAs, that is, BusAdmin1 and BusAdmin2, as shown in Figure 5. They share one Bus and exercise mastership for the duration of a turn. In addition, transitions *BusAdmin1*, *BusAdmin2,* and *Channel* are substitution transitions, which will be substituted by subpages *BusAdmin1*, *BusAdmin2,* and *Channel,* respectively. Places in the Top module are described in Table 3.

3.4.2. Channel Module. Data collected from the output buffers of BAs will be sent to the communication channel.

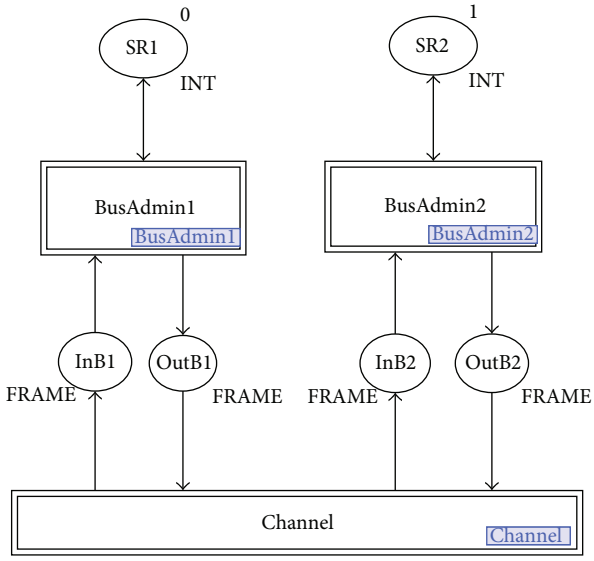

Figure 5: Top module.

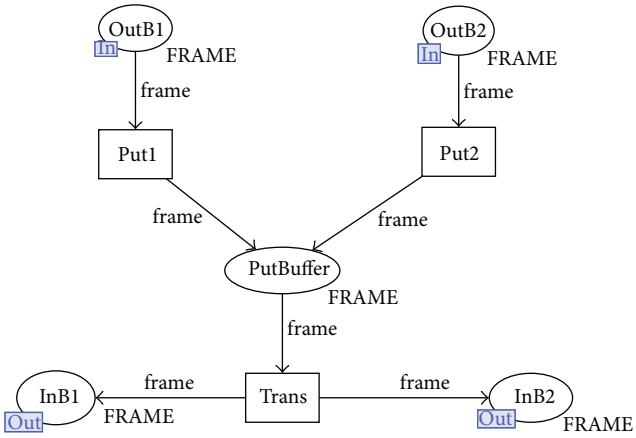

Figure 6: Channel module.

Then the channel broadcasts the data to inform each device on the network, as shown in Figure 6.

3.4.3. BusAdmin1 Module. This module implements the token passing algorithm mentioned in Section 2. The details of this module are given in Figure 7.

3.4.4. BusAdmin2 Module. The BusAdmin2 module is similar to BusAdmin1.

4. Synchronous and Asynchronous Modeling

Model checking is a formalization verification method for automatically verifying finite state systems. The procedure uses an exhaustive search of the state space of a system model to determine whether a specification is satisfied or violated.

Table 2: Input and output signals corresponding with its FID.

Signal	FID	Signal type
Mastership offer	MTRQ	Mastership transfer telegram
Mastership response	MTRP	Mastership transfer telegram
Inquire status	DSRQ	Device status telegram
Status response	DSRP	Device status telegram
T_standby, T_find_next, T_interim	TM	Time-out signals

Table 3: Places and their descriptions in the top module.

Place ID	Description
SR1	Address of BusAdmin1, default 0
SR2	Address of BusAdmin2, default 1
InB1	Input buffer of BusAdmin1
InB2	Input buffer of BusAdmin2
OutB1	Output buffer of BusAdmin1
OutB2	Output buffer of BusAdmin2

The process of model checking mainly includes three parts: modeling, specification, and verification. Modeling is to establish a model, which must essentially describe the behavior of the system. Requirement specification, which is usually given in some logical formalism, such as temporal logic, is to specify the properties that the model must satisfy. Verification is completely automated using model checking tools. Given a model and a specification, a model checking tool can determine whether the model satisfies the specification. If the model does not satisfy the specification, a model checker will give a counterexample execution of the model which demonstrates how the specification is violated.

A modeling and verification method for modeling external environments and temporal features is proposed in [15] for PLC (Programmable Logic Controller [16]) system. The idea is also suitable for MVB, but the method cannot be applied in MVB due to the differences between the PLC and MVB. In this section, we first propose a model of the BusAdmin and the Bus and then present two modeling methods, the synchronous and asynchronous modeling, including how to translate in the PROMELA codes of the model checker SPIN [17].

4.1. Bus Administrator, Bus, and Communication Model. A BusAdmin is a device that could be a Master device to control the Slave devices. There are several BAs in the MVB usually, but only one can be the Master at one time. The definition of a BusAdmin is given as follows.

Definition 4 (Bus Administrator). A BusAdmin A is a tuple (S, t, R, i, o), where

(1) S is a finite set of states;

(2) t is an initial state, $t \in S$;

(3) R is a transition relation, $R \subseteq S \times S$ such that R is left total, that is, for all $s \in S$, $\exists s' \in S$ such that $(s, s') \in R$;

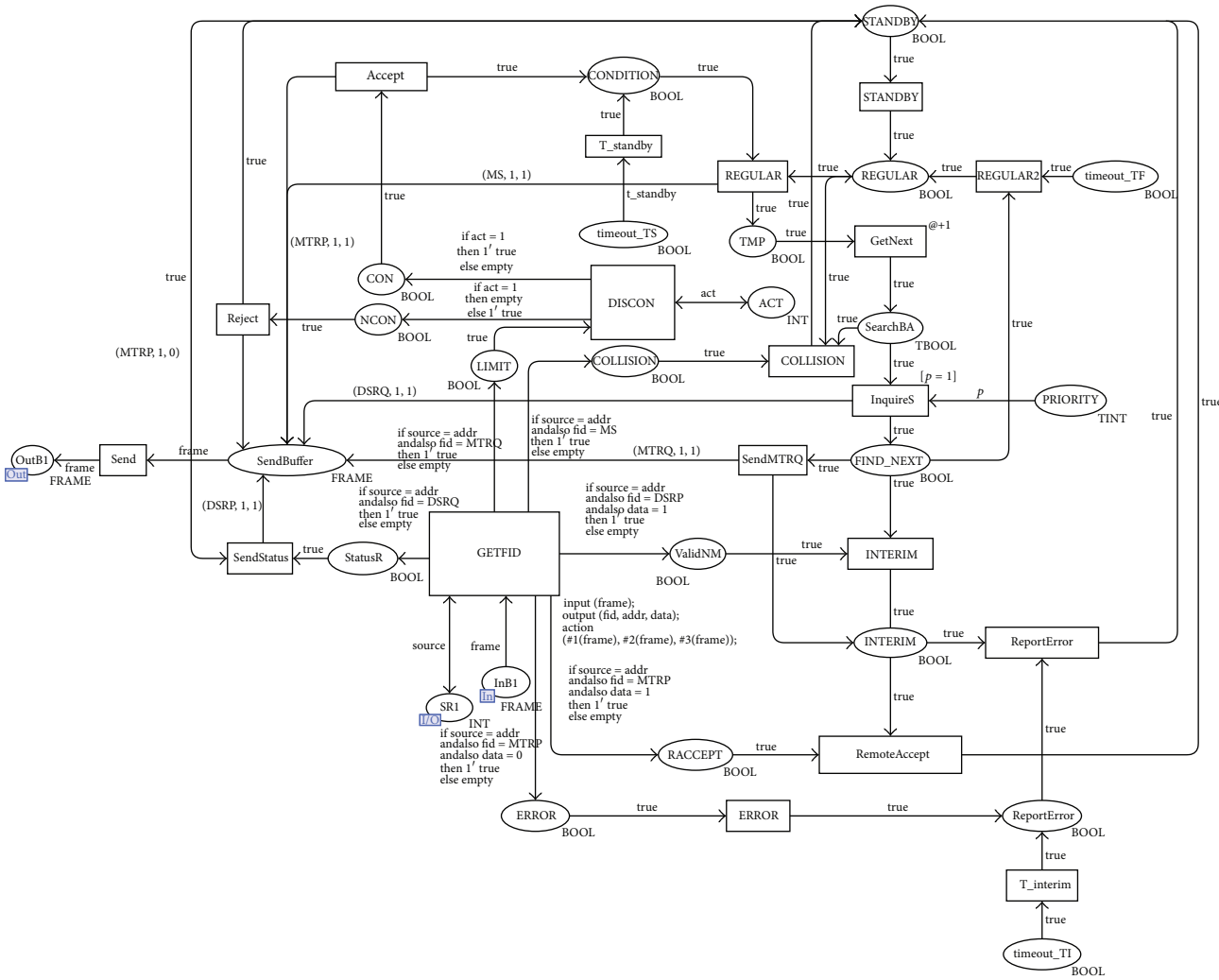

Figure 7: BusAdmin1 module.

(4) i is an input channel, which is used to receive messages from the Bus;

(5) o is an output channel, which is used to send messages to the Bus.

Each BusAdmin has a unique time-out T_standby, address, and rank. The behavior of a BusAdmin is mainly decided by the topological relation of its states and transition relation as shown in Figure 8. And part of the PROMELA codes of a BusAdmin is given in Listing 1.

The Bus is the medium through which a BusAdmin can send and receive frames with the others. The definition of a Bus is given as follows.

Definition 5 (Bus). A Bus B is a tuple (S, t, R, I, O), where

(1) S is a finite set of states;

(2) t is an initial state, $t \in S$;

(3) R is a transition relation, $R \subseteq S \times S$ such that R is left-total, that is, for all $s \in S$, $\exists s' \in S$ such that $(s, s') \in R$;

(4) i is a set of input channels, which is used to receive messages from the BAs;

(5) o is a set of output channels, which is used to send messages to the BAs.

The definition of the Bus is almost the same as the BusAdmin. The main difference between them is their behaviors: the behavior of the Bus is mainly broadcasting the frame sent by a BusAdmin to all the other BAs, and the frame in the Bus could be lost or be changed.

The communication model is the model of the MVB we want to verify. The definition of a communication model is given as follows.

Definition 6 (communication model). A communication model C is a tuple (A, b), where

(1) A is a set of BAs;

(2) b is a Bus, for all $a \in A$, b is connected with a.

```
active proctype Bus_Administrator_1 ( ) {
   ...
   BA_1_STANDBY_MASTER:   /*State Standby Master*/
   atomic {
        BA_1. curState = STANDBY_MASTER;
        BA_1. in_count > 0;   /*wait until BA_1. in_count>0*/
        BA_1. input ? frame;   /*receive a frame*/
        BA_1. in_count−−;
        if
        :: frame.type == LOST->
             ...  /*frame loss*/
        :: (frame. type == MASTERSHIP_OFFERED && frame.recv == BA_1. rank) ->
             ...  /*Mastership offered to me*/
        :: (frame. type == STATUS_REQUEST && frame.recv == BA_1. rank) ->
             ...  /*request status of me*/
        ::  (frame. type == REGULAR) ->
             ...  /*Master Frame*/
        :: else -> ...
        fi;
   }
   ...
}
```

Listing 1: PROMELA codes of a BA.

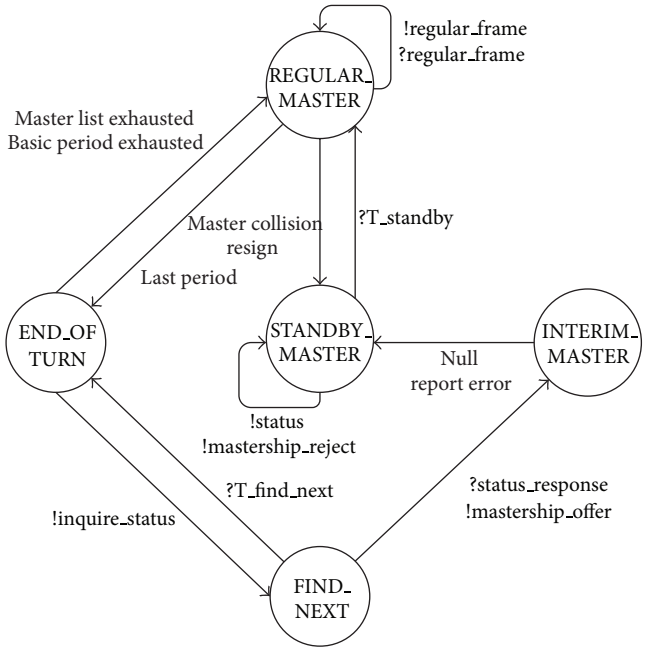

Figure 8: States and transitions of a BA.

Every BusAdmin is connected to the Bus, they do not send frames to the others directly, and all the frames are transmitted through the Bus. If a BusAdmin sends a frame, the Bus will receive the frame and send it to all the other BAs. So the number of the input channels of the Bus is the sum of all the BAs, and it is similar for the output channels.

4.2. Synchronous Modeling. The communication process of MVB depends on the Bus, by which all the devices exchange their messages periodically. To model periodical message transformation, an independent process is used to represent the Bus; it is defined as a finite state entity with input and output channels. Every BusAdmin in the communication model is defined as a state machine, while every state of each BusAdmin as an atomic step. The transition of BusAdmin will fire on the moment when the communication is taking place via the Bus. If there is no information exchange, the state of the BusAdmin will not detect the changes of itself or other BAs. Thus the state is a basic unit, within every basic period of the Bus communication, it experiences one and only one state.

The message transformation is taken place periodically with a transition fires of each BusAdmin synchronized. We cannot separate the communication process from time elapsing. So the communication phase and the synchronized states of each BusAdmin are activated alternately. So the basic period is viewed as a tick to model time. Time is split into end-to-end time zones, and the split lines are the same for different BAs. In every time zone, all the BAs stay in one and only one state. When it enters the next time zone, the state can change, and every BusAdmin can feel other BAs' changes expressed by the message transferred via the Bus. The communication mechanism is shown in Figure 9.

In the synchronous approach all the states of BAs are represented as automata transfer synchronously; that is, if a transition of some BusAdmin fires, it needs to wait for entering the next state until all the transitions of other BAs take place. So synchronous barriers are set where the transition lays between two states. The mechanism of synchronous

```
active proctype Synchronous_Bus ( ) {
atomic {
      if
      /*BA_0 as the Regular Master*/
      :: (BA_0. cur == REGULAR_MASTER && BA_1. cur != REGULAR_MASTER) ->
           BA_0. output ? frame; BA_0. input ! frame;
           master_type = frame. type;
           /*Master Frame from BA_0 to BA_1*/
           if
                :: true -> frame. type = LOST;
                :: true -> frame. type = master_type;
           fi;
           BA_1. input ! frame;
           /*Slave Frame from BA_1 to BA_0*/
           if
                :: frame. recv == 1 -> ... /*Answer the Master Frame*/
                :: else -> skip;
           fi;
      ... /* BA_1 as the Regular Master */
      :: (BA_0. cur!=REGULAR_MASTER && BA_1. cur!=REGULAR_MASTER) ->
           frame. type = LOST;  /*No Regular Master*/
           BA_0. input ! frame; BA_1. input ! frame;
      :: else -> ... /*Receive data from output channel of BAs in Regular Mastership*/
      fi;
      ...
}
```

Listing 2: PROMELA codes of the synchronous bus.

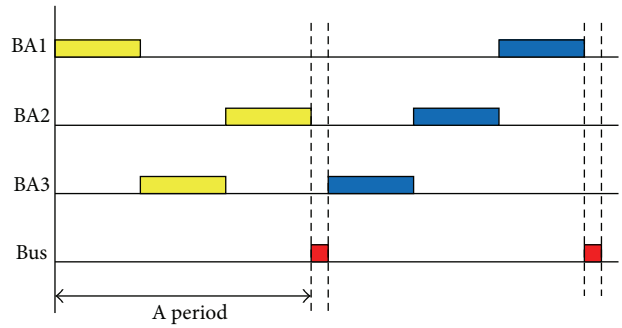

Figure 9: Timing diagram of the synchronous model.

barriers is that the channel of size zero in PROMELA is used, as shown in Listing 2. Channels are established between BAs and the Bus, and the communication is immediate without any buffer. When a BusAdmin is leaving an old state, it performs as the sender and the Bus performs as the receiver; conversely before it enters a new state, the Bus performs as the sender and it performs as the receiver. Two-way information exchanges between BAs and the Bus take place before and after the transitions take place synchronously.

As the communication through Bus performs periodically, the synchronous transitions are used to model the time elapse. Time in the synchronous approach is divided into small pieces, called the time spans, and the time spans appear one after another. For some BAs, a time appearance is associated with one and only one state. In the time span, no information exchange happens, so all the BAs do not know any state changes of the other BAs. When a time span is exhausted, the BAs leave the old state and tell all the other BAs through the Bus what changes the BAs which have left the old states has made. When entering the next time span, all the BAs get the information through the Bus, and get to know what they should do by the received messages, so certain transitions fire leading to proper state to work for certain purpose. If some time span is trivial, states also transfer along with its elapsing, but the pack in the channel transmission is label with Type SKIP, which implies the time does not elapse at this moment. The role of the Bus is not only to forward information from one end to another, but also to be potential to change the content of the information. For instance, the Bus can simulate the information losing though changing the information type to LOSE.

4.3. Asynchronous Modeling. The structures of the asynchronous and the synchronous models are alike. The main difference is that a BusAdmin can send a frame freely without waiting for the other BAs in the asynchronous model. So the sequence of the BAs' actions is not synchronous; it could be random as shown in Figure 10. And the channels are different too; each BusAdmin uses two channels whose size is not ZERO to communicate with the other BAs through the Bus, and the Bus broadcasts the frames sent by one BusAdmin to the other BAs with the channels.

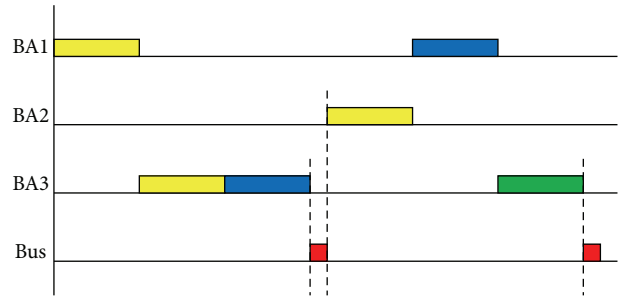

Figure 10: Timing diagram of the asynchronous model.

1	0	T_standby @ (1:BusAdmin2)
2	0	STANDBY @ (1:BusAdmin2)
3	0	REGULAR @ (1:BusAdmin2)
4	0	GetNext @ (1:BusAdmin2)
5	0	REGULAR @ (1:BusAdmin1)
6	0	Send @ (1:BusAdmin2)
7	0	Put2 @ (1:Channel)
8	0	Send @ (1:BusAdmin1)
9	0	Trans @ (1:Channel)
10	0	GETFID @ (1:BusAdmin2)
11	0	GetNext @ (1:BusAdmin1)
12	0	GETFID @ (1:BusAdmin1)
13	0	Put1 @ (1:Channel)
14	0	Trans @ (1:Channel)
15	0	GETFID @ (1:BusAdmin2)
16	0	GETFID @ (1:BusAdmin1)
17	1	COLLISION @ (1:BusAdmin1)
18	1	COLLISION @ (1:BusAdmin2)
19	1	STANDBY @ (1:BusAdmin2)
20	1	STANDBY @ (1:BusAdmin1)

Figure 11: Simulation report for HCPN model.

Every BusAdmin uses two variables, an input_count and an output_ count, to count the number of elements in the stack of the input and output channel separately. If a BusAdmin sends a frame, the variable output_count of it will be increased by one, and if a BusAdmin receives a frame, the variable input_count of it will be decreased by one. If any BusAdmin sends some frames, the Bus will receive the frame by checking whether the value of the output_count of the BusAdmin is zero and will send the received frames to the other BAs. When the Bus receives a frame from a BusAdmin, the variable output_count of the BusAdmin will be decreased by one, and when the Bus sends a frame to a BusAdmin, the variable input_count of the BusAdmin will be increased by one similarly.

The behavior of the Bus is mainly to receive and send frames; PROMELA codes of the Bus by the asynchronous method is shown in Listing 3. Firstly, the Bus checks the value of the variable output_count of each BusAdmin to decide whether to receive frames; if all the values are zero, the Bus will do nothing but waiting for the next frame or send a LOST frame to all the BAs to simulate the frame loss. If the value of the variable output_count of a BusAdmin is greater than zero, then the Bus will receive a frame from its output channel and send the frame to the other BAs. To simulate the real environment, any frame transmitted on the Bus could be changed or lost, so the frame send by the Bus could be changed or replaced by the LOST frame. When the Bus sends the frames, it will return to the beginning to receive and send frames.

5. Verification

In this section, the simulation of HCPN is presented firstly, then a tool M^3C is introduced, at last, the experimental results of verification of the Master Transfer are given, and the counter-examples is analyzed.

5.1. Qualitative Analysis of HCPN. We use the CPN Tools to do qualitative analysis. CPN Tools is a tool for editing, simulating, and analyzing untimed and timed Hierarchical Colored Petri Nets. In Section 2, two properties that ought to be satisfied are specified:

(1) Property 1: there cannot be more than one Master at one time;

(2) Property 2: there cannot be no Master at one time.

In the HCPN model, these properties are verified by analyzing simulation reports. A simulation report generated by CPN Tools is a sequence of transitions. In the simulation report, if the COLLISION transition appears, Property 1 is violated; if the STANDBY transition of BusAdmin1 is followed by the STANDBY transition of BusAdmin2, or vice versa, Property 2 is violated. Different initial markings are assigned to simulate our HCPN model depicted in Section 3.

Violations of both Property 1 and Property 2 are found when the initial condition is that BusAdmin1 is at its REGULAR state, BusAdmin2 is at its STANDBY state, and T_standby time-out of BusAdmin2 occurs. Assume that when Property 2 is dissatisfied, simulation will stop. Simulation results show that simulation will stop at the 20th step. A Simulation report with the counterexample is shown in Figure 11.

5.2. M^3C. Module Modeling and Model Checking (M^3C) is a tool we developed, it integrates the modeling and model checking. The modeling process uses modules, so that users could use mouse to drag and drop modules, such as state, to establish a model, like SIMULINK or VISIO. When the modeling process is done, it will translate the model into the PROMELA codes which is either of the PROMELA models mentioned in Section 4. After the user inputs the properties to be checked, it could invoke SPIN to execute the model checking and get the results of it, and then it will show the

```
active proctype Asynchronous_Bus ( ) {
    atomic {
    if
        :: BA_0. out_count > 0 -> ... /*receive a frame from BA_0*/
        :: else ->skip;
    fi;
    if
        :: !recv0 && !recv1 && !recv2 ->skip;
        :: !recv0 && !recv1 && !recv2 -> ... /*frame loss*/
        :: else -> skip;
    fi;
    if
        :: recv0 == true; /*receive a frame from BA_0*/
            if
            :: true -> .../*change the frame*/
            :: true -> ... /*frame loss*/
            :: true -> skip;
            fi;
            ... /*broadcast the frame*/
    fi;
    ...
}
```

Listing 3: PROMELA codes of the asynchronous bus.

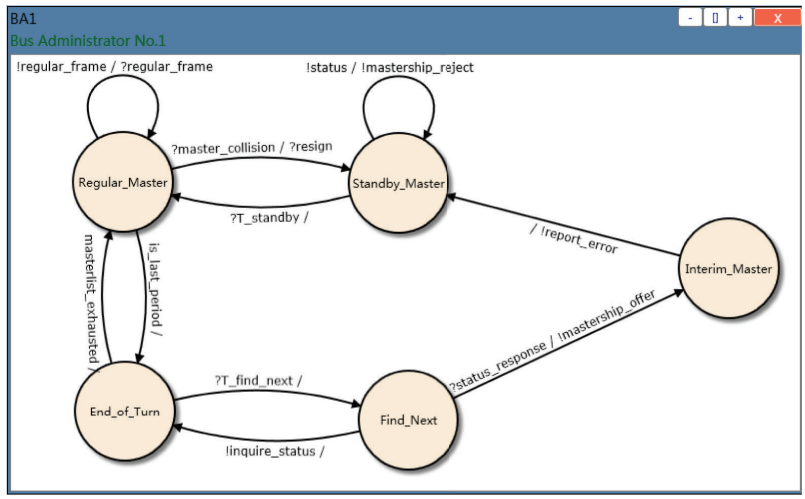

Figure 12: Model of a BusAdmin constructed in M^{3}C.

performance results and the counter-examples in a visual way. The model of a BusAdmin constructed in M^{3}C is shown in Figure 12.

5.3. Performance. The LTL formula (2) and formula (3) given in Section 2 are translated into PROMELA codes as follows (assume that there are three BAs and a Bus):

$p1$: [ ] ! is_master_collision

$p2$: [ ] ! (BA_0.curState == STANDBY_MASTER && BA_1.curState == STANDBY_MASTER && BA_2. curState == STANDBY_MASTER),

where the is_master_collision is a boolean variable that becomes true when a BusAdmin detects a Master_Frame that is not sent by itself. We configure the model checking process that it would stop at the first error and using both the depth-first and width-first search strategies. The experimental results of both properties by two models are given in Tables 4 and 5.

As shown in the results, violations of the both LTL formulae $p1$ and $p2$ can be found in both models. In the DFS mode, the asynchronous model's performance of states, transitions, time, and the counter-example path length is better than that of the synchronous model in most cases. When the number of BAs connected with the Bus gets to five,

Table 4: Experimental results of property $p1$.

Model	BA	DFS				BFS			
		States	Transitions	Time (sec.)	Path length (step)	States	Transitions	Time (sec.)	Path length (step)
Sync	2	121	121	0.001	280	1218	1728	0.003	159
	3	160	160	0.002	378	14369	24771	0.024	215
	4	199	199	0.002	476	192186	373760	0.438	271
	5	238	238	0.003	574	2764002	5787788	7.730	327
Async	2	32	47	0.001	132	22983	24373	0.029	72
	3	70	121	0.001	179	347201	375343	0.576	85
	4	168	379	0.003	231	OOM	OOM	NaN	NaN
	5	448	1313	0.004	354	OOM	OOM	NaN	NaN

Table 5: Experimental results of property $p2$.

Model	BA	DFS				BFS			
		States	Transitions	Time (sec.)	Path length (step)	States	Transitions	Time (sec.)	Path length (step)
Sync	2	133	133	0.002	305	1297	1842	0.003	166
	3	180	180	0.002	421	15241	26048	0.026	222
	4	227	227	0.002	537	203129	392149	0.462	278
	5	274	274	0.003	653	2908091	6068997	8.130	334
Async	2	40	57	0.001	193	60960	65385	0.078	89
	3	84	136	0.001	276	1063505	1163843	1.790	102
	4	190	401	0.002	364	OOM	OOM	NaN	NaN
	5	495	1361	0.005	645	OOM	OOM	NaN	NaN

the performance of the asynchronous model becomes worse in states, transitions, and time than that of the synchronous one, but the impact of the state explosion problem is still not too much significant. While in the BFS mode, the space and time cost is much bigger than that in the DFS mode, but on the contrary the path length is the shortest. The synchronous model performs better than the asynchronous one except for the path length in this mode.

The asynchronous model uses two variables to count the number of elements in the channels for each BusAdmin, while the synchronous model uses the channel whose size is zero and no need to count its content. So along with the increase of BAs, the state space of the asynchronous model grows faster than the synchronous model, and it implies the cost of space is lower for complex systems in the synchronous model. But when no BusAdmin sends and receives frames in a period, the Bus of the synchronous model would send SKIP frames to all BAs. It only means no BusAdmin takes any actions in last period and all BAs can enter the next period. The asynchronous model does not need to keep the period with all devices, so the counter-example path is shorter, and it implies it is easier to locate the error in the asynchronous model.

5.4. Counterexample Analysis. Both model checking based models can find the same violations. We trailed the counterexamples generated from both models and discovered that the violations they found are similar. The difference between them is mainly the path length; the synchronous model did some useless actions and transmitted some SKIP frames only to skip to the next period. If we ignore the useless actions and SKIP frames, the counter-example paths are almost the same.

The counter-example paths show that the violations could happen in these conditions.

(1) There are more than one Master at one time: initially, assume there is only one BusAdmin in state "REGULAR MASTER," and the other BAs are in state "STANDBY MASTER". If a BusAdmin in state "STANDBY MASTER" receives no Master Frames during a time-out T_standby, maybe because the frames are lost, then it will go to state "REGULAR MASTER." Eventually, there will be two Masters at one time.

There is no Master at one time: initially, assume there is only one BusAdmin in state "REGULAR MASTER", and the other BAs are in state "STANDBY MASTER". If a BusAdmin in state "STANDBY MASTER" receives no Master Frames during a time-out T_standby, maybe because the frames are lost, then it will go to state "REGULAR MASTER". Now, there are two BAs in state "REGULAR MASTER". If they both receive Master Frames that are not sent by themselves, and they assume a master collision, then they both return to state "STANDBY MASTER". Eventually, there will be no Master at one time.

As the analysis of the counter-examples shows, although the probability of the violation is very low, it is critical for the security of the control system. The Master Transfer protocol has some solution of the accidents, but it does not prevent the happening of these accidents.

6. Conclusions

In this paper, we presented two systematic approaches to model and verify the MVB. We gave two methods based on model checking and a method based on HCPN. We use these methods to model and verify the Master Transfer protocol of MVB, and they can all work properly. The synchronous model needs less states and transitions, and it is suitable for the periodical system; the asynchronous model costs less verification time and the path of counter-example is shorter; it is suitable for the periodical system. We also integrated them into our Modeling and Model Checking tool M^3C, so that they can be easy to use. Experimental results showed the efficiency and power of our approaches. The impact of the state explosion problem is not as significant as the general method periodic model.

Acknowledgment

This work is supported by the Funds NSFC61171121, NSFC60973049, the Science Foundation of Chinese Ministry of Education—China Mobile 2012.

References

[1] J. Jiménez, I. Hoyos, C. Cuadrado, J. Andreu, and A. Zuloaga, "Simulation of message data in a testbench for the multifunction vehicle bus," in *Proceedings of the 32nd IEEE Annual Conference on Industrial Electronics (IECON '06)*, pp. 4666–4671, 2006.

[2] H. Zhiwu, Z. Sheng, G. Weihua, and L. Jianfeng, "Research and design of protocol analyzer for multifunction vehicle bus," in *Proceedings of the 7th IEEE World Congress on Intelligent Control and Automation (WCICA '08)*, pp. 8358–8361, 2008.

[3] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logics of Programs*, vol. 131, pp. 52–71, Springer, Berlin, Germany, 1982.

[4] J. P. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in *International Symposium on Programming*, vol. 137, pp. 337–351, Springer, Berlin, Germany, 1982.

[5] C. Baier and J.-P. Katoen, *Principles of Model Checking*, MIT Press, Cambridge, Mass, USA, 2008.

[6] R. Jhala and R. Majumdar, "Software model checking," *ACM Computing Surveys*, vol. 41, no. 4, p. 21, 2009.

[7] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, pp. 2431–2436, 2000.

[8] P. Huber, K. Jensen, and R. M. Shapiro, "Hierarchies in coloured Petri nets," in *Advances in Petri Nets 1990*, vol. 483 of *Lecture Notes in Computer Science*, pp. 313–341, Springer, Berlin, Germany, 1991.

[9] G. Fadin, H. Kirrmann, and P. Umiliacchi, "Rosin, railway open system interconnection network. Web technologies for railways," in *Proceedings of Automation in Transportation*, 1998.

[10] P. Umiliacchi, "The role of european research in the railways modernisation process: the rosin project," in *Proceedings of the World Congress on Railway Research*, pp. 63–68, 1997.

[11] I. E. Commission et al., "IEC 61375-1," TrainCommunication Network, 1999.

[12] H. Kirrmann and P. Zuber, "The IEC/IEEE train communication network," *IEEE Micro*, vol. 21, no. 2, pp. 81–92, 2001.

[13] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, vol. 2, Cambridge University Press, Cambridge, Mass, USA, 2nd edition, 2004.

[14] K. Jensen, "Coloured Petri nets," in *Petri Nets: Central Models and Their Properties*, vol. 254, pp. 248–299, Springer, Berlin, Germany, 1987.

[15] P. Liu, G. Luo, M. Xia, and M. He, "Automatic verification of event-driven control programs: a case study," in *Proceedings of the 4th IEEE International Workshop on Advanced Computational Intelligence (IWACI '11)*, pp. 249–256, 2011.

[16] K. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*, Springer, 2010.

[17] G. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.