Modeling Distributed Real-time Elevator System by Three Model Checkers

https://doi.org/10.3991/ijoe.v14i04.8383

Qian Zhongsheng(✉), Li Xin, and Wang Xiaojin
Jiangxi University of Finance and Economics, Nanchang, China
changesme@163.com

Abstract—The characteristics of three popular model checking tools called SPIN, UPPAAL and NuSMV respectively, are compared and analyzed to determine which type of systems is propitious to be described. And a distributed elevator system model is built, whose related properties are verified and compared by these three model checking tools. To begin with, SPIN, UPPAAL and NuSMV, whose modeling language features are compared, are employed to construct an elevator system model respectively. Then, the three validation tools are used to verify several important properties of the elevator model, and the result is analyzed and their own characteristics are summarized. Finally, the experimental results show that SPIN and NuSMV are more suitable for verifying distributed systems while UPPAAL is better for verifying real-time systems.

Keywords—Distributed Elevator System, SPIN, UPPAAL, NuSMV, Linear Temporal Logic, Model Checking

1 Introduction

Since it was put forward by Clarke and Quielle respectively in 1981, model checking has been having a unique advantage of verifying security as stated in Ref. [1], communication protocol and security protocol of control system due to being capable of automatically detecting all possible states in system. Then, model checker was developed to validate the theory of model checking as stated in [2]. Presently, the most popular model checkers include UPPAAL, SPIN and NuSMV as stated in [3], [4] and [5].

SPIN, which is a model checking tool based on LTL (Linear Temporal Logic)[6], uses PROMELA[7] as programming language at the moment of building a model while LTL statement is input at the moment of verifying properties. There are many researches with respect to model checking using SPIN. For example, Nagafuji et al. built a verification system for a mathematical elevator model called S-ring using SPIN as stated in Ref. [8].

UPPAAL, which is a model checking tool based on timed automata, supports the property description language of CTL (Computation Tree Logic)[2] that is a kind of temporal propositional logic. Of course, there are also a lot of studies concerning building model using UPPAAL. For instance, Dai Sheng-Xin proposed a template that can be used to analyze and verify the schedulability of multiprocessor real-time systems in UPPAAL as stated in Ref. [9].

NuSMV, which is more widely used, supports not only CTL and LTL but also PSL (Property Specification Language) [10]. In order to verify Petri nets, Szpyrka et al. who successfully employed two temporal logics called CTL and LTL for verification, devised a tool to convert Petri net into NuSMV language as stated in Ref. [11].

These three tools that use different languages also support various temporal logics. Therefore, the properties verified by them are also different. And it is necessary that the properties and merits of three tools are compared and analyzed. Of course, we also need to compare and analyze which type of systems is suitable for being verified by them.

The main work of this paper is to use UPPAAL, SPIN and NuSMV to verify the properties of the same distributed real-time system, and to determine which properties and systems are appropriate for verifying by them.

2 Related work

There are a lot of studies about comparison concerning model checker and which type of system more appropriate to be verified by which model checkers. Samat et al. [12] compared are analyzed the input language of SMV, SPIN, UPPAAL and PRISM, and they described the limitations and differences of the input language of the four tools through modeling and verifying a traffic light system. And they got a conclusion that PROMELA, the input language of SPIN, is more appropriate for the description of traffic lights system. By comparing the verification time of a reachability formula, Daw et al. contrasted the verification performance using NuSMV, SPIN and UPPAAL as stated in [13]. And the UML activity diagram of an infusion pump is converted into the front-end language. The results show that the best performance converting the UML activity diagram into front-end language is UPPAAL, the second is SPIN, and the worst is NuSMV. Frappier et al. [14] compared and analyzed the validation of information systems using Alloy, cadp, fdr2, NuSMV, ProB and Spin. They verified a number of properties of behavior, attributes and entity instances of a library management system, whose results proved that ProB is the most suitable for verifying the information system. In order to verify models with a large number of state variables, Choi Y et al. compared the validation effectiveness of the flight guidance system as stated in [15]. And the results show SPIN was much fitter for verifying the flight guidance system because SPIN is better able to avoid explosion state. Morimoto et al. [16] converted the BPEL of describing business processes into timed automata to simulate and verify the business process model of the enterprise. And these technologies can be used to detect and correct the error of model as early as possible before implementation. Aydal E G et al. compared four tools, respectively called USE, Alloy Analyzer, ZLive and ProZ as stated in [17]. And how to effectively verify system using different tools was proved by modeling and verifying the same system.

Different from the existing work, the elevator system is proved to be concurrent and real-time in this paper. And the modeling language of UPPAAL, SPIN and NuSMV is compared. Besides, several properties are analyzed by comparing the verification effect of elevator system.

3 Constructing system model

The difference of distributed real-time elevator control system model and common elevator control system model is that the former considers the elevator user's behavior into model. And there is not only an elevator control process in the model, but also a process of description for user's behavior. There is the clock constraint in the entire process of using the elevator, which can display the property of real-time. The reason for choosing the distributed real-time elevator system model is that it has more properties to be verified than single real-time system or single distributed system.

3.1 Distributed real-time elevator system

The elevator model used in this paper is a general elevator with a single compartment in the dormitory. We assume that there are eight floors in total and a single compartment. The operation flow of elevator system is shown in Fig. 1.

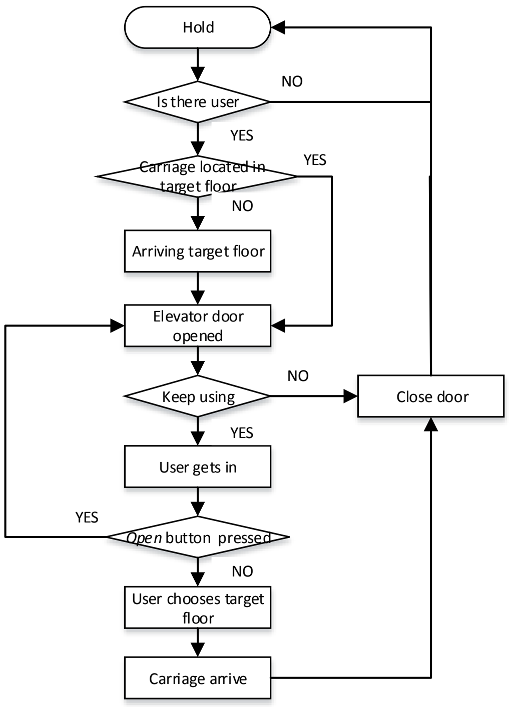


Fig. 1. Operation flow of elevator system

In the elevator model, each operation has time constraints. In other words, each operation will be finished within a certain period of time. And the carriage of the elevator will automatically enter the next state if there is no other operation to be done during this period.

According to the operation flow of the above elevator model, we can get the following important parameters as shown in table 1.

Table 1. The parameters of elevator system

Parameter	Significance	Data type
User_floor	User's initial floor	int
Car_floor	Elevator car floor	int
Aim_floor	User's destination floor	int
car	Elevator car status (0 for the stop, the move for the 1)	bool
door	Elevator door status (0 for close, 1 for open)	bool
user	User status (0 in the elevator, 1 out the elevator)	bool
Clk_ele	Elevator process clock	clock
Clk_user	User process clock	clock
mt	Maximum travel time of elevator carriage	int
dt	Elevator door closing and opening time	int

3.2 Elevator system model specified by SPIN, UPPAAL and NuSMV

Three model checking tools, called SPIN, UPPAAL and NuSMV, respectively have their own modeling language and different modeling style. In order to compare the characteristics of the three modeling languages, three tools are used to model the elevator system, respectively. We will compare and analyze whether the three tools can be used to describe the elevator system in detail and whether there are some problems cannot be well expressed in the process of modeling.

Elevator system model specified by SPIN. According to the elevator system model, when modeling elevator system by SPIN, the whole model needs to be divided into two processes: MAN process and ELE process. The role of the keyword "atomic" is to ensure that the process of atomicity. In other words, each process is not affected each other. Process declarations are as follows.

```
init
{
  atomic{run MAN();}
  atomic{run ELE ();}
}
```

Channels are used to communicate between processes in SPIN. In order to reflect the real-time response of the elevator, the channel is not able to store messages, so the message must be sent immediately when it arrives.

In the elevator system model, 5 channels are declared, namely, Opress, OPpress, Nopress and Ipress. Channels declarations are as follows.

```
chan get=[0] of {byte}; chan Opress=[0] of {byte};
chan Ipress=[0] of {byte}; chan OPpress=[0] of {bool};
chan Nopress=[0] of {bool};
```

After the arrival of the carriage of the elevator to the target floor, the "get" channel is used to send a message to the user process, and the type of message is byte and the content is the number of floor to be reached. The message that the user outside the elevator sends to the elevator process in Opress channel expresses the number of the target floor where the user stands, and the type of message is byte. However, the message that the user in the carriage of the elevator sends to the elevator process in Ipress channel expresses the number of the target floor where the user wants to reach, and the type of message is byte type. The message in OPpress channel that expresses the meaning of opening the door is sent to the ELE process. And the message type is bool type, and the number "1" means opening the door of the elevator. The Nopress channel is used to send a message to the ELE process when the user stand outside the carriage and the elevator is in the "Owait" state without further action. And the message type is bool type, and the number "1" means that the user does not enter any command and leave the elevator. Specific PROMELA code is as follows.

```
proctype MAN()
{
  outside:
  {
    clk_user=0;user=0;Opress!user_floor;gotoOwait;
  }
  Owait: ...
  inside: ...
  Iwait: ...
}
proctype ELE()
{
  hold:
  {
    clk_ele=0;door=0;car=0;Opress?user_floor;
    if
      ::(car_floor==user_floor)->goto open;
      ::(car_floor!=user_floor)->goto move;
    fi
  }
  move: ...
  open: ...
  close: ...
  }
```

There are 4 states, respectively called outside, Owait, inside, Iwait, in the MAN process. The meaning of the 4 states is as follows.

- outside: the user is ready to use the elevator outside the elevator.
- Owait: the user presses the button to use the elevator, waiting for the arrival of the carriage of the elevator.
- inside: the user is ready to enter the destination floor in the elevator.
- Iwait: the user waits for the elevator to reach the destination floor and open the elevator door.

Corresponding to the MAN process, the ELE process also has 4 states, namely, hold, move, open, and close.

- hold: the elevator is waiting for use.
- move: the elevator car is in operation and stops after arrival.
- open: the door of the elevator is opened, waiting for the user to come in or out.
- close: the door of the elevator is closed.

Fig. 2 is the result of running data in Random mode in Simulate interface, and Fig. 3 is the communication details between the ELE process and the MAN process.

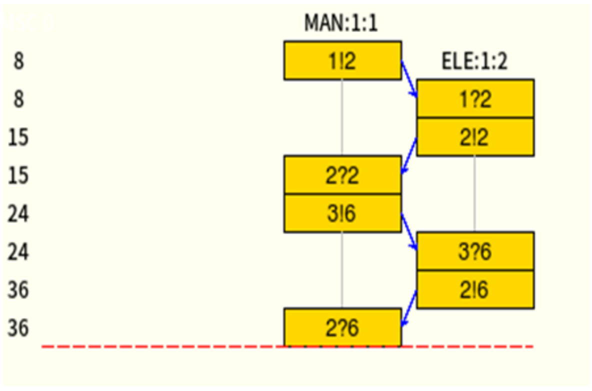
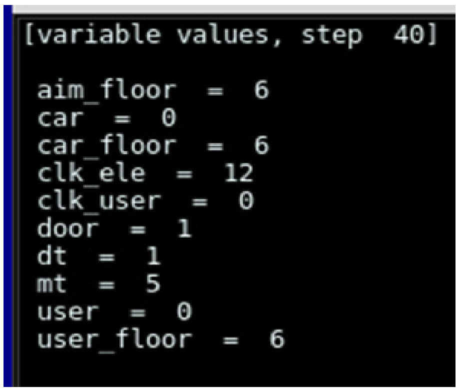


Fig. 2. Communication between processes



```
[variable values, step  40]

 aim_floor  =  6
 car  =  0
 car_floor  =  6
 clk_ele  =  12
 clk_user  =  0
 door  =  1
 dt  =  1
 mt  =  5
 user  =  0
 user_floor  =  6
```

Fig. 3. Result of communication

Elevator system model specified by UPPAAL. Due to the definition of parameters and the establishment of the model are separated on the editor interface, the parameters of elevator system should be declared when the elevator system model is built in UPPAAL [18]. The two processes ELE and MAN in SPIN are represented as two processes in UPPAAL, namely elevator and man. The channels of passing messages between processes in the UPPAAL statement are as follows.

```
  chan get; // The get channel is used to transmit the
message of arrival.
  chan Opress[8];// Opress[8] is a set of channel groups
that are used to transmit messages to elevator when user
presses button on all floors to use elevator.
  chan Ipress[8]; //Ipress[8] is a set of channel groups
that are used to transmit messages to elevator when user
presses button on all floors to choose destination floor.
  chan OPprrss;// The OPprrss channel is used to transmit
the message of open door.
  chan Nopress;// The Nopress channel is used to transmit
the message of the user giving up the use of elevator
messages.
```

The difference between UPPAAL channel and SPIN channel is that the former can’t store a number of messages in a channel or define the type of message.

The four states {hold, move, open, close} of process ELE in SPIN are represented as four location {hold, move, open, close} of elevator model.

The elevator system modeled by UPPAAL is shown in Fig. 4. It is different from SPIN that the hold location has an initial clock constraint: Clk_ele<=0. The hold location is different from the other locations as the initial state, but there is no initial state in the PROMELA. The edge from the hold location to the move location, the green font (Car_floor! =User_floor) is a guard, equivalent to the case of select statement::Car_floor!=User_floor. UPPAAL does not has select statement, only has judgment statement. The blue font (Car_floor=User_floor) is the update of the data that is executed in the move location, and the update of the data in the SPIN is assigned to the data at the beginning of the state.

There is a clock constraint Clk_ele<=mt at move location, and mt is maximum travel time of elevator car. In SPIN there is no clock constraint.

The four states {outside, Owait, inside, Iwait} of process ELE in SPIN are represented as four states {outside, Owait, inside, Iwait} of elevator model.

The elevator model modeled by UPPAAL is shown in Fig. 4 and Fig. 5. The two processes have their own initial state and clock. In man process, there is a clock Clk_user which is independent of Clk_ele.

The communication details between the models are shown as Fig. 6, and Fig. 7 is the result of date running in simulator.

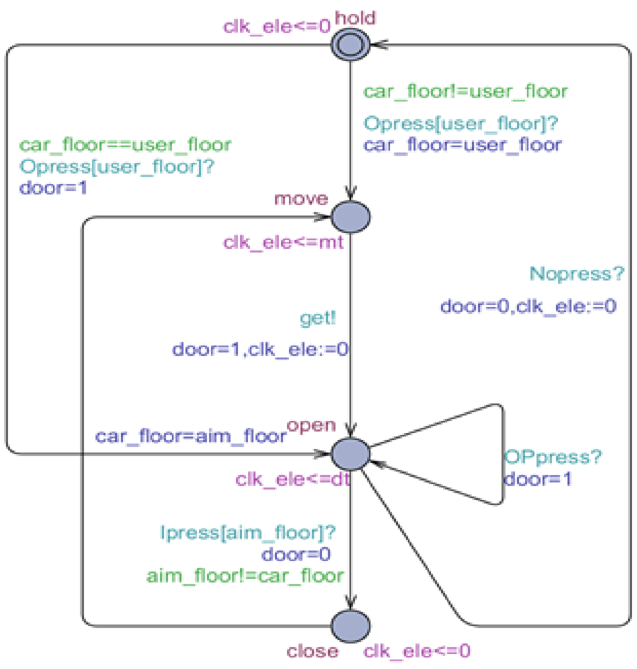


Fig. 4. Elevator template

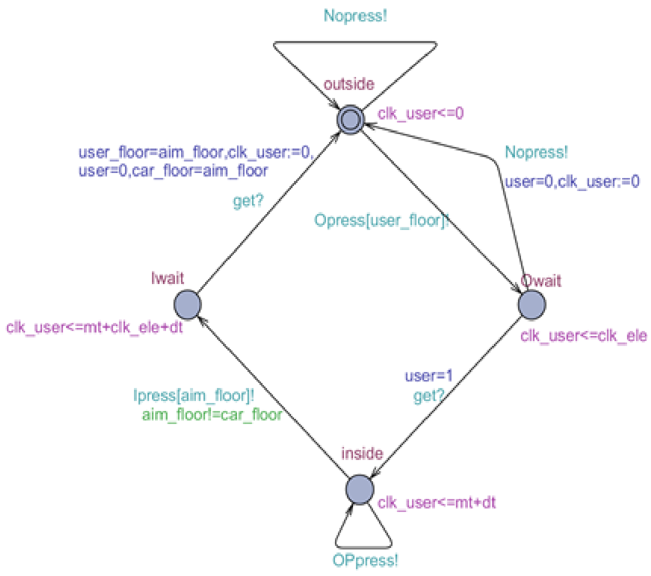


Fig. 5. Man template

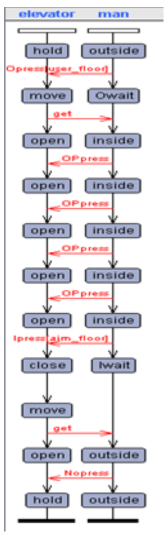
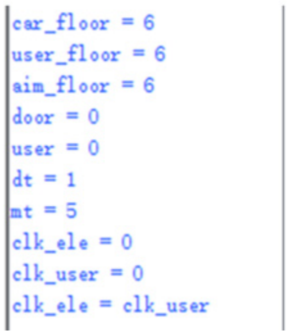


Fig. 6. Communication between templates



```
car_floor = 6
user_floor = 6
aim_floor = 6
door = 0
user = 0
dt = 1
mt = 5
clk_ele = 0
clk_user = 0
clk_ele = clk_user
```

Fig. 7. Result of data running

Elevator system model specified by NuSMV. Unlike SPIN and UPPAAL, the NuSMV system model specification is written in an elevator.smv file which is edited by using text editor. The definition and assignment of parameters are in the process of elevator and man in SPIN. In NuSMV, we need to define and give the initial value in main module. There is no channel in NuSMV, but it can also complete the communication between the processes through the global parameters. In the elevator system model we define a global parameter press, mpress and epress are defined in the man process and ele process to correspond. Press from 1 to 8 is equivalent to the channel Opress in SPIN, and Press from 9 to 18 is equivalent to the channel Ipress in SPIN, press 17 is equivalent to the channel get in SPIN. The data type of the parameters press is Boolean, the value is {TRUE, FALSE}. Specific code is as follows.

```
MODULE main
VAR
  pr1:process man(press);
  pr2:processele(press);
  press:array 1..17 of boolean;
  user:boolean; car_floor:1..8; user_floor:1..8;
  aim_floor:1..8; door:boolean; car:boolean;
ASSIGN
  init(press[1]):=FALSE; init(press[2]):=FALSE;
    ...
  init(user_floor):=1; init(aim_floor):=6;
  init(car_floor):=8; init(door):=FALSE;
  init(car):=FALSE;  init(user):=FALSE
SPEC A [(car=TRUE)U(door=FALSE)]
```

```
  SPEC EF (pr1.mstate=Iwait)
  SPEC AG(pr1.aim_floor=6)-> EF(pr1.user_floor=6)
  SPEC EF (pr2.car_floor=1)
  SPEC EF
((pr2.car=TRUE)&(pr1.user=FALSE)&(pr1.mstate=Iwait))
  SPEC EBF 0..10 pr1.user_floor=6.
```

The two processes in the SPIN contain four states, the two processes are divided into four small modules, and the parameters are changed in these four modules. But the NuSMV code is different, a process is divided into modules, respectively, the VAR parameter definition module and the ASSIGN parameter change module, all the parameters are a small change of the module. In upper code, mstate is the states of user, are equal to four locations in UPPAAL, namely, {outside, Owait, inside, Iwait}. All parameter updates need to be determined in the "next" function, if the case is correct then implement the statement behind the ":". Due to space limitations, here is only part of the key code.

```
  MODULE man(mpress)
  VAR
    mstate:{outside,Owait,inside,Iwait};
    ...
  COMPUTE MIN[mstate=outside,mstate=Iwait];
  ASSIGN
    init(mstate):=outside;...
    next(mstate):=
    case
      (mstate=outside)&(user=FALSE):Owait;
      (mstate=Owait)&mpress[17]:inside;
      (mstate=inside)&(aim_floor!=car_floor):Iwait;
      (mstate=Iwait)&mpress[17]:outside;
       TRUE:mstate;
    esac;...
```

Estate: {hold, move, open, close} is the four states of the elevator in ele process, and it is equivalent to the four states of ELE process in SPIN.

4 Verification, comparison and analysis

For the elevator system, the following properties need to be verified:

- Safety: it is the most important property. And there will not be any risk that can endanger the safety of users and any dangerous things in the entire running process of elevator system.
- No deadlock: there will be no deadlock when the elevator system is running, which can ensure that the user will not be unable to get out of the elevator.
- Activity: elevator will eventually reach the target floor.

- Reachability: there is always a way, by which the user can reach the destination floor.
- Real-time: the elevator can arrive within the specified time.
- Fault tolerance: when the user is wrong, the elevator is not affected to continue to run.
- Concurrency: two users use the elevator in the same time.

4.1 The result of verification and its analysis in SPIN

Properties in LTL for 7 properties above are shown in the table 2 as follows.

Table 2. Properties in LTL of SPIN

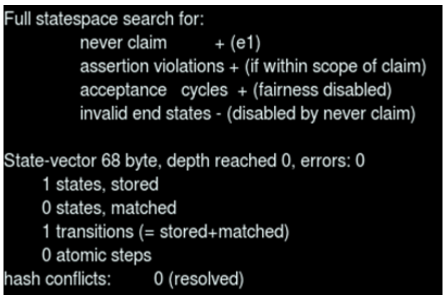
Properties	Properties in LTL	Result
Safety	ltl e1 {!([](c==1)&&(door==1))};	TRUE
No deadlock	ltl e3 {[]((User_floor==2)-><>(Car_floor==2))};	TRUE
Activity	ltl e2 {<>(Car_floor==6)};	TRUE
Reachability	ltl e3 {[]((Aim_floor==6)-><>(User_floor==6))};	TRUE
Real-time	ltl e4 {[](Clk_user<=13)};	FALSE
Fault tolerance	ltl e5{<>((Aim_floor!=User_floor)&&(car==1)&&(user==0)&& (Clk_user==6))};	FALSE
Concurrency	ltle6{[](((Aim_floor==3)&&(Aim_floorb==3))-> <>((User_floor==3)&&(User_floorb==3)))};	FALSE

Fig. 8 is the result of security verification in SPIN. As it is shown in the figure, there is no security error or warning in the elevator system. Fig. 9 is the result of fault tolerance verification in SPIN, and there is an obvious error in this figure. Due to the limited space, there are only two figures of verification result.

To verify the concurrency of the elevator system, we create another user process MAN_1, and define Opress and Ipress as channels, which can store 1 message in the ELE process.

```
chan Opress=[1] of {byte}; chan Ipress=[1] of {byte};
```

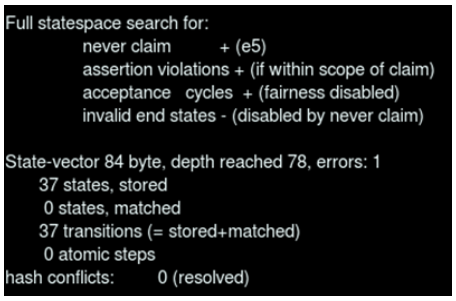
The concurrency in LTL is described that two users at different floor have the same destination floor. And the result is FALSE. The reason is that when the two user processes to send messages in the channel, the ELE process will only read one message and finish it while another process will permanently block in the elevator system model. It means another user will always stay out of the elevator, so the elevator system has no concurrency.



```
Full statespace search for:
          never claim         + (e1)
          assertion violations + (if within scope of claim)
          acceptance  cycles  + (fairness disabled)
          invalid end states - (disabled by never claim)

State-vector 68 byte, depth reached 0, errors: 0
     1 states, stored
     0 states, matched
     1 transitions (= stored+matched)
     0 atomic steps
hash conflicts:        0 (resolved)
```

Fig. 8. Security verification



```
Full statespace search for:
          never claim         + (e5)
          assertion violations + (if within scope of claim)
          acceptance  cycles  + (fairness disabled)
          invalid end states - (disabled by never claim)

State-vector 84 byte, depth reached 78, errors: 1
     37 states, stored
      0 states, matched
     37 transitions (= stored+matched)
      0 atomic steps
hash conflicts:        0 (resolved)
```

Fig. 9. Fault tolerance verification

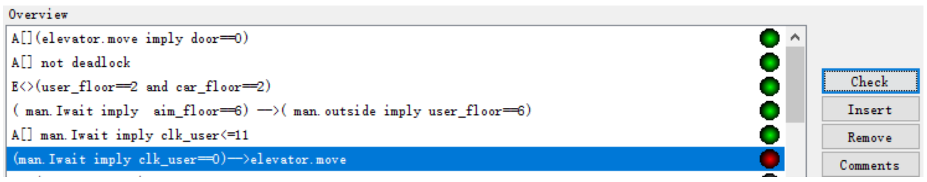
4.2 The result of verification and its analysis in UPPAAL

Properties in CTL for 7 properties above are shown in the table 3 as follows.

Table 3. Properties in CTL of UPPAAL

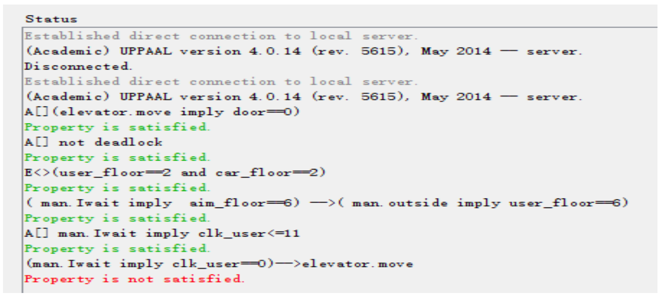
Properties	**Properties in CTL**	**Result**
Safety	A[](elevator.move imply door==0)	Satisfied
No deadlock	A[] not deadlock	Satisfied
Activity	E<>(User_floor==2 and Car_floor==2)	Satisfied
Reachability	(man.Iwait imply Aim_floor==6) -->(man.outside imply User_floor==6)	Satisfied
Real-time	A[] man.Iwait imply Clk_user<=11	Satisfied
Fault tolerance	(man.Iwait imply user==0)-->elevator.move	Not satisfied
Concurrency	This property cannot be expressed	Not satisfied

Fig. 10 shows description in CTL of UPPAAL for the first 6 properties, and the green dot behind properties indicates that property is satisfied, the red dot behind properties indicates that property is not satisfied. Fig. 11 shows the final verification results.



```
A[](elevator.move imply door==0)                                   ●
A[] not deadlock                                                   ●
E<>(user_floor==2 and car_floor==2)                                ●
( man.Iwait imply  aim_floor==6) -->( man.outside imply user_floor==6)  ●
A[] man.Iwait imply clk_user<=11                                   ●
(man.Iwait imply clk_user==0)-->elevator.move                      ●
```

Fig. 10. Properties in CTL of UPPAAL



```
Status
Established direct connection to local server.
(Academic) UPPAAL version 4.0.14 (rev. 5615), May 2014 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.14 (rev. 5615), May 2014 -- server.
A[](elevator.move imply door==0)
Property is satisfied.
A[] not deadlock
Property is satisfied.
E<>(user_floor==2 and car_floor==2)
Property is satisfied.
( man.Iwait imply  aim_floor==6) -->( man.outside imply user_floor==6)
Property is satisfied.
A[] man.Iwait imply clk_user<=11
Property is satisfied.
(man.Iwait imply clk_user==0)-->elevator.move
Property is not satisfied.
```

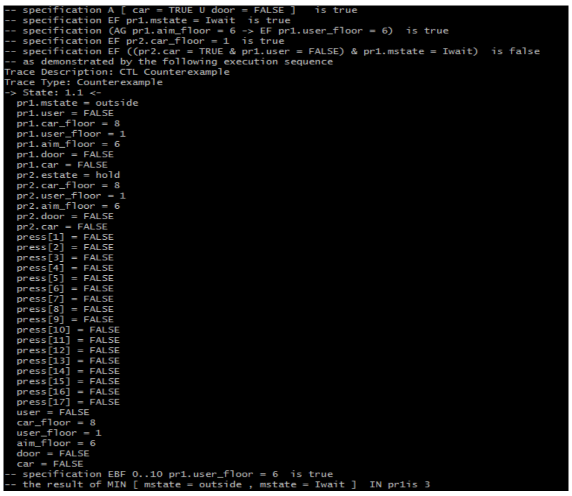
Fig. 11. The final verification results in UPPAAL

4.3 The result of verification and its analysis in NuSMV

Properties could be described in CTL, LTL and PSL, table 4 describes 7 properties above in CTL.

Table 4. Properties in CTL of NuSMV

Properties	Properties in CTL	Result
Safety	SPEC A[(car=TRUE)U(door=FALSE)]	TRUE
No deadlock	COMPUTE MIN[mstate=outside,mstate=Iwait];	TRUE
Activity	SPEC EF (pr2.car_floor=1)	TRUE
Reachability	SPEC AG(pr1.aim_floor=6)-> EF(pr1.user_floor=6)	TRUE
Real-time	SPEC EBF 0..10 pr1.user_floor=6	TRUE
Fault tolerance	SPEC EF ((pr2.car=TRUE)&(pr1.user=FALSE)&(pr1.mstate=Iwait))	FALSE
Concurrency	SPEC EF (pr3.ib=6)&(pr1.i=6)	FALSE



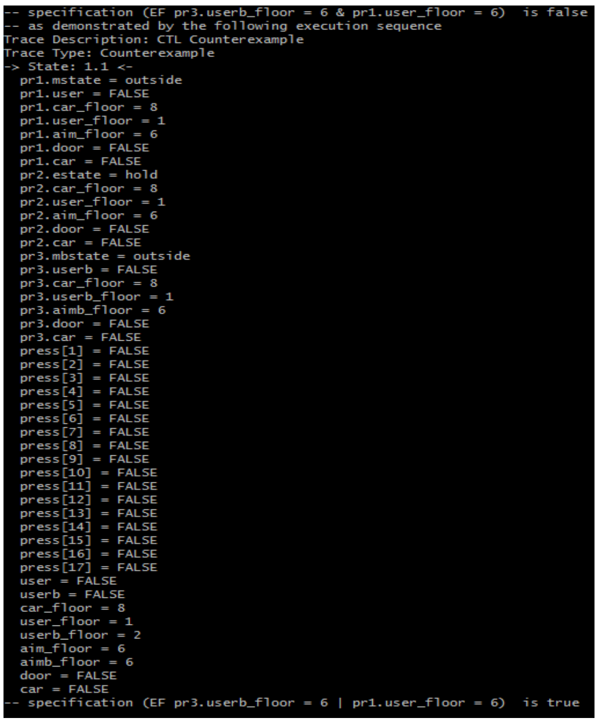
```
-- specification A [ car = TRUE U door = FALSE ]   is true
-- specification EF pr1.mstate = Iwait  is true
-- specification (AG pr1.aim_floor = 6 -> EF pr1.user_floor = 6)  is true
-- specification EF pr2.car_floor = 1  is true
-- specification EF ((pr2.car = TRUE & pr1.user = FALSE) & pr1.mstate = Iwait)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  pr1.mstate = outside
  pr1.user = FALSE
  pr1.car_floor = 8
  pr1.user_floor = 1
  pr1.aim_floor = 6
  pr1.door = FALSE
  pr1.car = FALSE
  pr2.estate = hold
  pr2.car_floor = 8
  pr2.user_floor = 1
  pr2.aim_floor = 6
  pr2.door = FALSE
  pr2.car = FALSE
  press[1] = FALSE
  press[2] = FALSE
  press[3] = FALSE
  press[4] = FALSE
  press[5] = FALSE
  press[6] = FALSE
  press[7] = FALSE
  press[8] = FALSE
  press[9] = FALSE
  press[10] = FALSE
  press[11] = FALSE
  press[12] = FALSE
  press[13] = FALSE
  press[14] = FALSE
  press[15] = FALSE
  press[16] = FALSE
  press[17] = FALSE
  user = FALSE
  car_floor = 8
  user_floor = 1
  aim_floor = 6
  door = FALSE
  car = FALSE
-- specification EBF 0..10 pr1.user_floor = 6  is true
-- the result of MIN [ mstate = outside , mstate = Iwait ]  IN pr1is 3
```

Fig. 12. The verification results in NuSMV

Fig. 12 is the verification result of first 6 properties by NuSMV, and there is an execution sequence for a test case. As demonstrated by this execution sequence, the property will run error.

Concurrency needs to be added to the code in a user process man2, the code is as follows.

```
MODULE main
VAR
  pr1:process man1(press);
  pr2:processele(press);
  pr3:process man2(press);
  press:array 1..17 of boolean;
  ......
ASSIGN
  init(press[1]):=FALSE;
  init(press[2]):=FALSE;
  ......
SPEC EF (pr3.userb_floor=6)&(pr1.user_floor=6)
SPEC EF (pr3.userb_floor=6)|(pr1.user_floor=6)
```



```
-- specification (EF pr3.userb_floor = 6 & pr1.user_floor = 6)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  pr1.mstate = outside
  pr1.user = FALSE
  pr1.car_floor = 8
  pr1.user_floor = 1
  pr1.aim_floor = 6
  pr1.door = FALSE
  pr1.car = FALSE
  pr2.estate = hold
  pr2.car_floor = 8
  pr2.user_floor = 1
  pr2.aim_floor = 6
  pr2.door = FALSE
  pr2.car = FALSE
  pr3.mbstate = outside
  pr3.userb = FALSE
  pr3.car_floor = 8
  pr3.userb_floor = 1
  pr3.aimb_floor = 6
  pr3.door = FALSE
  pr3.car = FALSE
  press[1] = FALSE
  press[2] = FALSE
  press[3] = FALSE
  press[4] = FALSE
  press[5] = FALSE
  press[6] = FALSE
  press[7] = FALSE
  press[8] = FALSE
  press[9] = FALSE
  press[10] = FALSE
  press[11] = FALSE
  press[12] = FALSE
  press[13] = FALSE
  press[14] = FALSE
  press[15] = FALSE
  press[16] = FALSE
  press[17] = FALSE
  user = FALSE
  userb = FALSE
  car_floor = 8
  user_floor = 1
  userb_floor = 2
  aim_floor = 6
  aimb_floor = 6
  door = FALSE
  car = FALSE
-- specification (EF pr3.userb_floor = 6 | pr1.user_floor = 6)  is true
```

Fig. 13. The verification result of concurrency in NuSMV

Assuming that the user, called user b, is on the 2nd floor and another user is on the 1th floor, and their destinations are the 6th floor, and we should verify whether both of them will reach the 6th floor. The result is shown in Fig. 13 that there is only one user will arrive the 6th floor. It shows that this elevator does not have concurrency.

4.4 Comprehensive comparison and analysis of property verification

Table 5 is the comparison of properties verification in SPIN, UPPAAL and NuSMV.

Table 5. The comparison of properties verification

	SPIN	**UPPAAL**	**NuSMV**
Safety	established	established	established
No deadlock	established	established	established
Activity	established	established	established
Reachability	established	established	established
Real-time	cannot be described	established	cannot be described
Fault tolerance	not established	not established	not established
Concurrency	not established	cannot be described	not established

From table 5, the performance of verifying the distributed real-time elevator system by three tools is different. The performance of real-time verification by SPIN and NuSMV is not as good as that by UPPAAL. SPIN can add a timing clock into PROMELA program, which serves to the user to calculate the time of using the elevator. NuSMV can calculate the shortest path to reach destination floor. However, SPIN and NuSMV do not have the clock system and can not set the clock constraint on each state while SPIN and NuSMVcan not verify the real-time. UPPAAL is less effective for concurrency verification than SPIN and NuSMV, because the UPPAAL channel can not store two messages at the same time and do description when the two users use the elevator at the same time. Therefore, it can be concluded that SPIN and NuSMV are more suitable for the verification of distributed systems, while UPPAAL is more propitious to the verification of real-time systems.

5 Concluding remarks and future work

SPIN, UPPAAL and NuSMV are used to model the distributed real-time elevator system, which adds user behavior as a process. Three tools with different modeling methods can be successfully used to model the elevator system, which shows that these three tools can be used to verify the common information system. And some results are found by modeling the same distributed real-time elevator system.

1. The state description for the next location be regarded as update for parameters in PROMELA, which is not as obvious and specific as update for parameters in UPPAAL and NuSMV.
2. UPPAAL does not have a direct loop structure, which leads to a worse performance compared with a do statement of PROMELA.
3. Compared with the other two tools, the modeling language of NuSMV is more convenient and simple, and updating for each parameter is more distinguishable where modeling code is more verbose for lack of channel.

SPIN, UPPAAL and NuSMV are used to verify several important properties of elevator model. Based on the comparison of the results, we find that SPIN and NuSMV are more suitable for verifying distributed systems while UPPAAL is better for verifying real-time systems.

To find which model checking tool is fit for the distributed elevator system, we will further contrast the effect and performance of the distributed elevator system verified by SPIN and NuSMV in the next step.

6 Acknowledgment

Thanks go first to the anonymous referees for their sound comments and suggestions. This work is partly supported by the National Natural Science Foundation of China under Grant Nos. 61762041 and 61462030, and the Science and Technology Project of Jiangxi Provincial Department of Education of China under Grant No. GJJ160427.

7 References

[1] Clarke E M. The Birth of Model Checking[M]. Springer-Verlag, 2008: 1-26. https://doi.org/10.1007/978-3-540-69850-0_1
[2] Clarke, E. M, Emerson, et al. Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications [J]. ACM Transactions on Programming Languages & Systems, 1994, 8(2): 244-263. https://doi.org/10.1145/5397.5399
[3] Vardi M Y, Wolper P. An Automata-theoretic Approach to Automatic Program Verification [J]. Proc. of the First Annual IEEE Symp on Logic in Computer Science, 1986: 322-331.
[4] Holzmann G J. The Model Checker SPIN [J]. IEEE Transactions on Software Engineering, 1997, 23(5): 279-295. https://doi.org/10.1109/32.588521
[5] Cimatti A, Clarke E, Giunchiglia F, et al. NUSMV: A New Symbolic Model Checker [J]. International Journal on Software Tools for Technology Transfer, 2000, 2(4): 410-425. https://doi.org/10.1007/s100090050046
[6] Holzmann G. The SPIN Model Checker: Primer and Reference Manual[M]. Addison-Wesley Professional, 2011.
[7] Mikk E, Lakhnech Y, Siegel M, et al. Implementing Statecharts in Promela/SPIN[C]// IEEE Workshop on Industrial Strength Formal Specification Techniques, 1998: 90-101.
[8] Nagafuji K, Yamaguchi S. Éclair: An Elevator Group Controller Model Checking System based on S-ring and SPIN[C]// Consumer Electronics. IEEE, 2014: 178-181.

[9] Dai Sheng-Xin, Hong Mei, Guo Bing. Schedulability Analysis Model for Multiprocessor Real-Time Systems Using UPPAAL [J]. Journal of Software (in Chinese), 2015(2): 279-296.
[10] Pnueli A, Zaks A. PSL Model Checking and Run-Time Verification Via Testers[M]// Formal Methods.Springer, Berlin Heidelberg, 2006: 573-586.
[11] Szpyrka M, Biernacka A, Biernacki J. Methods of Translation of Petri Nets to NuSMV Language[C]// 23rd International Workshop on Concurrency, Specification and Programming. Chemnitz, Germany, Sept. 2014.
[12] Samat P A, Zin A M, Shukur Z. Analysis of The Model Checkers' Input Languages for Modeling Traffic Light Systems[J]. Journal of Computer Science, 2011, 7(2): 225-233. https://doi.org/10.3844/jcssp.2011.225.233
[13] Daw Z, Cleaveland R. Comparing Model Checkers for Timed UML Activity Diagrams[J]. Science of Computer Programming, 2015, 111: 277-299. https://doi.org/10.1016/j.scico.2015.05.008
[14] Frappier M, Fraikin B, Chossart R, et al. Comparison of Model Checking Tools for Information Systems[C]// International Conference on Formal Engineering Methods and Software Engineering. Springer-Verlag, 2010: 581-596.
[15] Choi Y. From NuSMV to SPIN: Experiences with Model Checking Flight Guidance Systems [J]. Formal Methods in System Design, 2007, 30(3): 199-216. https://doi.org/10.1007/s10703-006-0027-9
[16] Morimoto S. A Survey of Formal Verification for Business Process Modeling [J]. 2008, 14(4): 514-522.
[17] Aydal E G, Utting M, Woodcock J. A Comparison of State-Based Modelling Tools for Model Validation [M]// Objects, Components, Models and Patterns. Springer, Berlin Heidelberg, 2008: 278-296.
[18] Fatima T, Saghar K, Ihsan A. Evaluation of Model Checkers SPIN and UPPAAL for Testing Wireless Sensor Network Routing Protocols[C]// International Bhurban Conference on Applied Sciences and Technology. IEEE, 2015: 263-267.

8 Authors

Qian Zhongsheng is a professor in School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, 330013, China. His main research direction is software testing and model checking. (changesme@163.com)

Li Xin is a postgraduate student in School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, 330013, China.. His main research direction is model checking and software verification. (1594919301@qq.com)

Wang Xiaojin is a postgraduate student in School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, 330013, China.. His main research direction is model checking and intelligent algorithm. (wxjin107@qq.com)

Article submitted 16 October 2017. Resubmitted 07 February, 28 February and 15 March 2018. Final acceptance 31 March 2018.