J Inf Process Syst, Vol.15, No.3, pp.570~592, June 2019
https://doi.org/10.3745/JIPS.01.0043

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

LTS Semantics Model of Event-B Synchronization Control Flow Design Patterns

Han Peng*, Chenglie Du*, Lei Rao**, and Zhouzhou Liu*

Abstract

The Event-B design pattern is an excellent way to quickly develop a formal model of the system. Researchers have proposed a number of Event-B design patterns, but they all lack formal behavior semantics. This makes the analysis, verification, and simulation of the behavior of the Event-B model very difficult, especially for the control-intensive systems. In this paper, we propose a novel method to transform the Event-B synchronous control flow design pattern into the labeled transition system (LTS) behavior model. Then we map the design pattern instantiation process of Event-B to the instantiation process of LTS model and get the LTS behavior semantic model of Event-B model of a multi-level complex control system. Finally, we verify the linear temporal logic behavior properties of the LTS model. The experimental results show that the analysis and simulation of system behavior become easier and the verification of the behavior properties of the system become convenient after the Event-B model is converted to the LTS model.

Keywords

Behavior Semantic, Design Pattern Instantiation, Event-B Design Patterns, Labeled Transition System

1. Introduction

Event-B [1] is a formal language that is closest to software engineering. Its ideas of progressive refinement and ability of automatic code generation not only ensure the correctness and consistency of the model but also ensure the correctness of the final code. In order to enhance the reusability of the Event-B model to better support the software development process, the researchers proposed the Event-B design pattern [2], that is, the reusable Event-B models. The Event-B design pattern is similar to design pattern in software engineering but extends reusability to the correctness of the model, that is to say, the Event-B design patterns which have been verified can be instantiated as a part of the larger software model and do not need to be constructed it again and prove its correctness again. Now, the Event-B design pattern has been applied to embedded control systems [1], service-oriented architecture [3], software product line engineering [4,5], wireless sensor networks [6], and many other fields.

However, as a data-oriented modeling language, Event-B focuses only on the consistency of refinement, while it has limitations in the preservation and verification of behavioral properties. This is mainly because Event-B has no behavior semantics, which makes it difficult for the modeler to analyze and verify

Manuscript received September 29, 2017; first revision March 13, 2018; accepted May 17, 2018.
Corresponding Author: Han Peng (hansbeng2016@gmail.com)
* School of Computer Science, Northwestern Polytechnical University, Xi'an, China (hansbeng2016@gmail.com, {ducl, liuzhouzhou}@mail.nwpu.edu.cn)
** School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an, China (rl15829292198@gmail.com)

the linear temporal logic (LTL) behavior properties of the Event-B model directly. But it is well known that we should do more stringent verification on those components that will be more likely to be reused. Unfortunately, although the Event-B design pattern has been widely used, there is little research on its behavior semantics and verification of its behavioral properties.

In this paper, we propose a method to model the Event-B synchronize control design patterns using the labeled transition system (LTS). In detail, the contributions of this work are as follows.

- We analyzed in detail the event order of the four Event-B synchronous control flow patterns, and revealed the complexity of the Event-B control flow design patterns as well as the difficulty to characterize it.
- We proposed the concept of "atomic LTS" and used this concept to model the four synchronous control flow patterns. We proposed three transformation rules to construct "atomic LTS" and used rigorous logic derivation to prove the correctness of our transformation rules.
- We constructed an LTS model of a complex multi-level control system using the LTS semantic model of the Event-B synchronous control flow design patterns. Our approach is to map the instantiation process of the Event-B model to the instantiation process of its corresponding LTS semantic model, which guarantees the behavioral equivalence between the source model (i.e., the Event-B model) and the target model (that is, the LTS semantic model).
- We characterized the functional requirements and safety requirements of the complex multi-level control system using the LTL formula, and test these LTL behavioral properties of the target model using model checking tools to evaluate the feasibility of our method.

The rest of this paper is organized as follows: Section 2 discusses some related work and their flaws. In Section 3, we explain the principle of the Event-B synchronous control flow design pattern. Section 4 proposes three translation rules and gives the LTS behavior semantic model of the Event-B synchronous control pattern. Section 5 proposes the method to map the design pattern instantiation process of Event-B model to the instantiation process of LTS model and builds a complex control system model using this method. Section 6 gives the experimental results and discusses it. Section 7 concludes our work.

2. Related Work and Problem Definition

Event-B design pattern: Event-B design pattern is a reusable formal model that is different from the design pattern in the software engineering. The idea of the Event-B design pattern is to construct and prove the formal models of the relatively small problems in order to reuse these small formal models to construct the larger model. In this way, the modeler does not have to prove the correctness of these small models again. In other words, by using the Event-B design pattern, we can reuse not only the design strategy of the model but also the correctness of the model. Therefore, the direct benefit of the Event-B design pattern is that it can greatly reduce proof cost of the formal model.

Many Event-B design patterns have been proposed in recent years. For example, Silva and his colleague [7,8] proposed the Event-B design pattern and the "Generic Instantiation" approach and developed the Event-B model of the safety critical subway system. Yeganefard et al. [9] proposed the "MCMC" design patterns of Event-B model to model the generic formal components of the monitored, controlled, mode and commanded (MCMC) architecture. MCMC design patterns have been used to construct the Event-B model of cruise control system [10], automotive lane departure warning system [11], and lane centering

controller [12]. The MCMC design pattern reuses the static architecture of the model, allowing the modeler to use this architecture directly when solving similar problems.

Modeling control flow of Event-B: How to characterize the control flow (i.e., event order)of Event-B code clearly, has always been a concern of researchers because it enables researchers to model and analyze the system's behavior and further verify its behavioral properties (such as LTL properties). Fathabadi et al. [13,15] proposed a method named "atomic decomposition" which used a tree structure based on the Jackson structure diagram (JSD) to express the relationship between the abstract event and the subsequent concrete events as well as the order of concrete events. This method is used to construct the Event-B model of the multimedia protocol system [15] and the Space Craft system [16]. Iliasov [17] proposed a method named Flow language which uses *ena*, *dis*, and *fis* to express the order of the events, and uses the Flow plug-in to provide graphical symbols to model the event orders. Schneider et al. [18-20] proposed the CSP||B method, using communication sequential process (CSP) to express the control flow part of system, and using pure Event-B model to express the data processing part of system (that is, the computation part). Finally, he transforms the CSP model into the Event-B code and combines it with the computation part to form a complete Event-B model. We can verify the behavior of final Event-B model through verifying the behavior of CSP model. The inventor of Event-B, Abrial [1] proposed four synchronous control flow patterns, including strong synchronization pattern, weak synchronization pattern, strong-weak synchronization pattern, and strong-strong synchronization pattern. Its main purpose is to express the basic sub-problem in the reactive system, that is, the "trigger-response" relationship between components. We can use these patterns to establish the Event-B model of complex control system easily.

The LTL property of Event-B model: As we mentioned, one of the goals of modeling the control flow of Event-B code is to verify its behavioral properties. Literature in this area is rare. Current research is limited to how to preserve its LTL attributes during the refinement process of the Event-B model. For example, Schneider et al. [21] proposed a set of refinement strategies that can preserve the safety properties of Event-B models in the refinement chain. Further, Schneider et al. [22] extends this idea to the preservation of liveness properties and derives the refinement strategy that can preserve the temporal logic properties of the Event-B model throughout the refinement chain. Recently, Hoang et al. [23] gave more general results, which relax the constraints between adjacent refinement levels of Event-B model while ensuring that the LTL properties continue to hold during the refinement process.

2.1 Problems of Existing Research

Although many works have been done on the modeling of Event-B control flow and the LTL properties of Event-B, there are still some problems in current research.

First, these works cannot express Event-B's control flow explicitly. Although the "atomic decomposition" method claims that it can express the control flow of Event-B, the control flow is not visible in its tree structure. Flow method uses the relationship between events to express control flow, which is contrary to the style of the traditional transition system. The CSP||B method works best in this respect but still cannot express the control flow of a complete Event-B model. The synchronous control flow design pattern itself is expressed using Event-B code. The flaws in these methods make it impossible for people to clearly observe and analyze the behavior of the model.

Second, the results of these works are difficult to translate directly into LTS models. LTS is a traditional

and authoritative behavioral semantic model, which is also the theoretical basis of LTL model checking. However, the results of various Event-B control flow modeling methods, such as atomic decomposition and Flow methods, cannot be directly converted to LTS models. Although the CSP has formal behavioral semantics and can be directly mapped to the LTS model, the CSP||B method only expresses a part (that is, the control flow) of the Event-B model using the CSP. Therefore, using CSP||B, we cannot get a complete LTS model of Event-B model.

Finally, these works do not support verification of the LTL property of the Event-B model. Although Schneider et al. [21,22] and Hoang et al. [23] studied the behavioral semantics and LTL properties preservation of Event-B, their constraints on the refinement strategy of the Event-B model will limit the exploration of the design space for the Event-B model.

In conclusion, the available control flow modeling methods have some limitations and do not support verification of LTL properties. In contrast, we propose an explicit way to model Event-B control flow and give its LTS behavioral semantics that can support LTL properties verification.

3. Event-B Synchronous Control Flow Design Patterns

Event-B is an event-based formal modeling language whose core concept is event. The general form of an event is as follows:

$$e \triangleq \text{WHEN } guards \text{ THEN } actions \text{ END}$$

An event is made up of *guards* and *actions*. An event *e* is enabled when its guards are satisfied, and actions express the effect of the event *e*, that is, the modification of the variables. Event-B uses variables to express the state of the system and perform the state changes with events.

Abrial [1] proposed four synchronous control flow patterns to abstract the basic "trigger-response" relationship in a multi-level control system. The basic principle of "trigger-response" relationship is shown in Fig. 1.

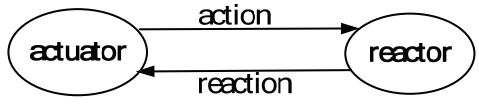


Fig. 1. The "trigger-response" relationship of the reactive system.

In Fig. 1, the actuator executes the *action* event, and modifies its own state. The reactor is enabled after the *action* event and can execute the *reaction* event.

In the synchronous control flow design pattern, the variables *a*, *b* and *r*, *s* are used to represent the state of the actuator and the reactor, respectively. The *x_on* event changes the value of the variable *x* from 0 to 1, and the event *x_off* changes the value of the variable *x* from 1 to 0, where $x \in \{a, r, b, s\}$. We refer to a pair of "actuators – reactors" as a "subsystem".

For the sake of discussion, we first consider an Event-B model including the actuator *a* and reactor *r* with no synchronization control relationship, named *non_control_model*:

a_on ≙	a_off ≙	r_on ≙	r_off ≙
STATUS	STATUS	STATUS	STATUS
Ordinary	Ordinary	Ordinary	Ordinary
WHEN	WHEN	WHEN	WHEN
@guard2:a = 0	@guard2:a = 1	@guard2:r= 0	@guard2:r =1
THEN	THEN	THEN	THEN
@action1:a := 1	@action1:a := 0	@action1:r := 1	@action1:r := 0
END	END	END	END

In *non_control_model*, there is no *r*-related expression in the guards of the *a_on* and *a_off* events. Thus, the occurrence of the above two events is not constrained by the variable *r*. Likewise, the *r_on* and *r_off* events are not constrained by the variable *a*.

3.1 Weak Synchronization Pattern

The *weak* synchronization pattern means that after the actuator executes trigger event *a_on*, the reactor can either respond to it (execute the *r_on* event) or make no response (do not execute the *r_on* event). This relationship also applies to events *a_off* and *r_off*. A realistic case of this pattern is the response of the keyboard to the keystroke event—in many cases, a user may trigger many "button is pressed" events in one second because of his finger jitter, but in fact, he just wants to trigger one event.

The principle of *weak* synchronization pattern is shown in Fig. 2. Where the two signals labeled by *a* and *r* represent the states of the actuator *a* and the reactor *r*, respectively. The rising edge indicates that the value of *a* or *r* changes from 0 to 1 while the falling edge changes their values from 1 to 0.

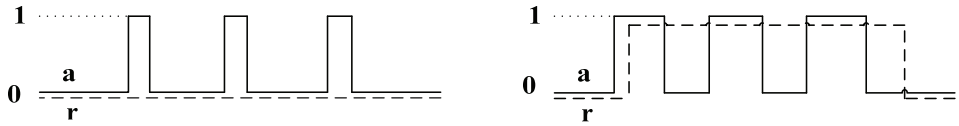
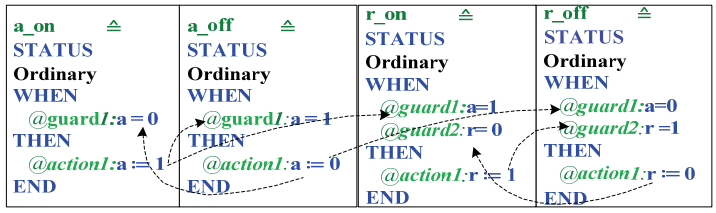


Fig. 2. The principle of weak synchronization pattern: (a) case 1 and (b) case 2.

The Event-B model of the *weak* synchronization pattern, *weak_model* is:



Compared with the *non_control_model*, *weak_model* adds some constraint in the event *r_on* and *r_off*, which specifies that the *r_on* event can occur only when *a* = 1 (event *a_on* has occurred). Similarly, *r_off* events can only occur after event *a_off*. We added some dashed arrows to the Event-B model to indicate the triggering relationship between events. They indicate that two events will be enabled, *a_off* and *r_on* after the *a_on* event occurs. Similarly, after the *action1*: *a*:= 0 in the *a_off* event is performed, there are two events be enabled, *a_on* and *r_off*.

3.2 Strong Synchronization Pattern

The *strong* synchronization pattern is similar to the synchronous call in the function call, that is, the reactor *r* must perform the event *r_on* after the actuator *a* executes the event *a_on*, otherwise the actuator will wait forever until the reactor responds. Thus, the actuator can only execute the *a_off* event after the reactor executes the *r_on* event, and the reactor must, in turn, execute the *r_off* event after *a_off* event occurs. The principle of *strong* synchronization pattern is shown in Fig. 3.

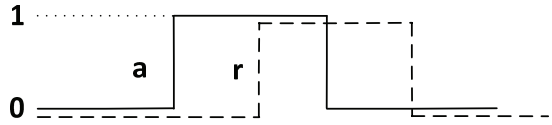


Fig. 3. The principle of *strong* synchronization pattern.

The Event-B model of the *strong* synchronization pattern, *strong_model* is:

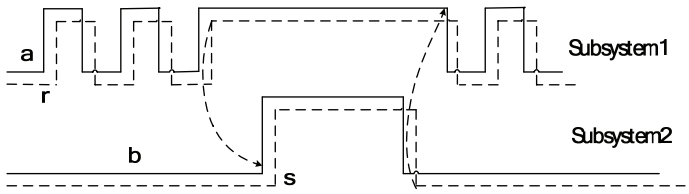
a_on ≙	a_off ≙	r_on ≙	r_off ≙
STATUS	STATUS	STATUS	STATUS
Ordinary	Ordinary	Ordinary	Ordinary
WHEN	WHEN	WHEN	WHEN
@guard1:r = 0	@guard1:r = 1	@guard1:a=1	@guard1:a=0
@guard2:a = 0	@guard2:a = 1	@guard2:r= 0	@guard2:r =1
THEN	THEN	THEN	THEN
@action1:a := 1	@action1:a := 0	@action1:r := 1	@action1:r := 0
END	END	END	END

Strong_model adds more constraints to the actuator events *a_on* and *a_off* of the *weak_model* so that the events in the model must only be executed in the order indicated by the dashed arrows.

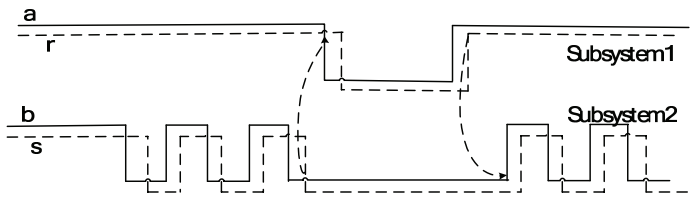
3.3 Strong-Weak Synchronization Pattern

In a multi-level control system, there is also a "trigger-response" relationship between subsystems. For example, if we name the strong synchronization subsystem composed of the actuator *a* and the reactor *r* as sub-system 1, while name the strong synchronization subsystem composed of the actuator *b* and the reactor *s* as sub-system 2, then the *strong-weak* synchronization relation describes the "*weak* trigger-response" relationship between these two strong synchronization subsystems. The principle of *strong-weak* synchronization is shown in Fig. 4.

Strong-weak synchronization pattern has two possible cases. In the first case, the *r_on* event of subsystem 1 can trigger the *b_on* event of subsystem 2, but it is not necessarily a strong synchronization relationship, that is, when *r_on* event is executed, subsystem 2 can either execute *b_on* event or maintain its own state unchanged, as shown in Fig. 4(a). In the second case, the *s_off* event of subsystem 2 can trigger the *a_ off* event of subsystem 1 and is not a *strong* synchronization relationship either, that is, after the *s_off* event occurs, the subsystem 1 may either execute *a_ off* event or maintain its state unchanged, as shown in Fig. 4(b).



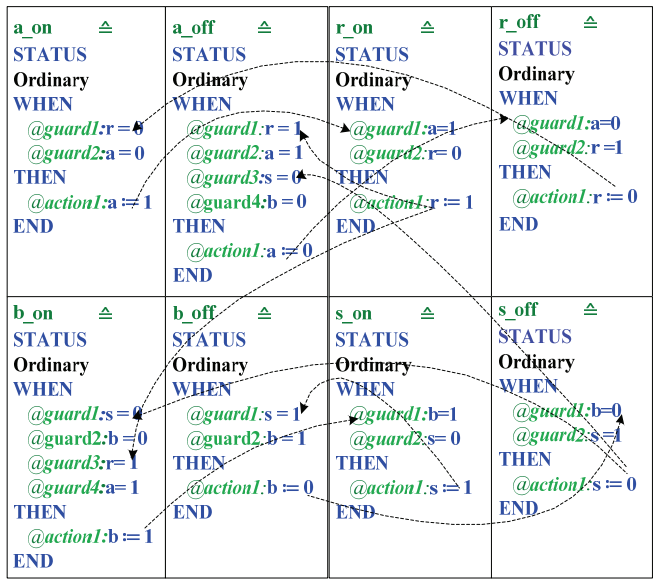
(a)



(b)

Fig. 4. The principle of *strong-weak* synchronization pattern: (a) case 1 and (b) case 2.

The Event-B model of the *strong-weak* synchronization pattern, *strong_weak_model* is:



Although we use the dotted arrows to express as clearly as possible the order of events in the *strong-weak* synchronization relationship, it is obvious that it is very difficult to clearly analyze the event order in such a two-level control system model. For the more complex multi-levels control system, it is almost impossible to intuitively and clearly express the event order of system model.

3.4 Strong-Strong Synchronization Pattern

The *strong-strong* synchronization pattern means that: firstly, the reactor *r* of the subsystem 1 is the trigger of the actuator *b* of the subsystem 2. The subsystem 2 must perform the *b_on* event as long as the

r_on event occurs. Secondly, once the *b_on* event occurs, then the subsequent event must be *s_on*, *b_off*, *s_off* because the subsystem 2 is a *strong* synchronization system. Thirdly, the subsystem 1 will maintain its state (that is, *a = 1, r = 1*) after *r_on* event occurs until the *s_off* event is executed by the subsystem 2. That is to say, the *a_off* event must be triggered by the *s_off* event. The principle of *strong- strong* synchronization is shown in Fig. 5.

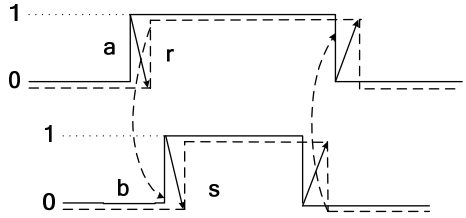
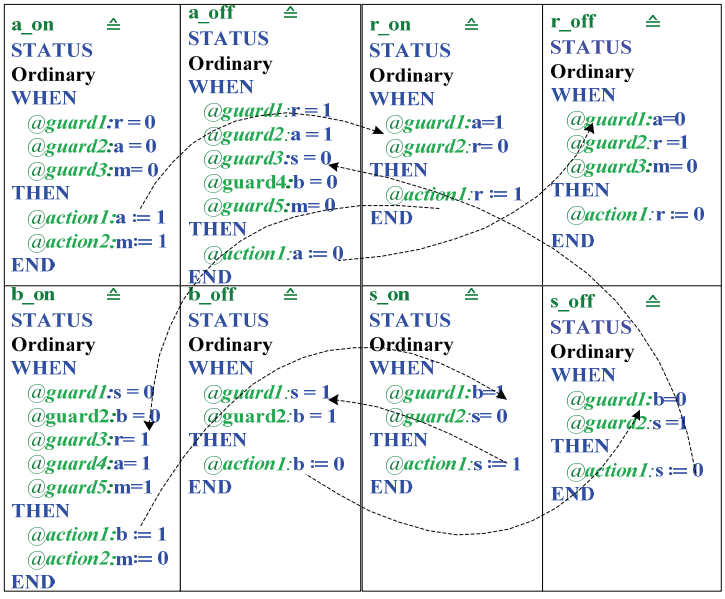


Fig. 5. The principle of *strong- strong* synchronization pattern.

The Event-B model of the *strong-weak* synchronization pattern, *strong_strong_model* is:



In the *strong_strong_model*, a state variable *m* has been added based on *strong_weak_model* to constrain the relationship between *a_on* and *a_off*, which means *a_off* can be executed only after the *b_on* event occurs. Since *b* and *s* composed a *strong* synchronization subsystem, when *b_on* occurs, it means that its subsequent events must be *s_on*, *b_off* and *s_off*, so *a_off* only can occur after the event sequence "*b_on*, *s_on*, *b_off*, *s_off*", that is, *a_off* only can occur after the *s_off* event.

We can see that: firstly, the event order of Event-B design patterns is difficult to analyze. Secondly, the process from the *non_control_model* to the *weak_model* and then to the *strong_model* is an incremental process that enhances the constraint. Similarly, from the *strong_weak_model* to *strong_strong_model* is also an increasing constraint enhancement process. This inspires us to establish the LTS semantic models of the four synchronic control flow patterns in an incremental way.

4. The LTS Model of the Event-B Synchronization Patterns

In this section, we will build the LTS behavior semantic model of the Event-B synchronous control flow design pattern. LTS is a state transition system in which the transition is labeled as an action. The action set of the LTS is called its communication alphabet.

Definition 1.1 (LTS [24]): Let *States* represent a universal set of states, *Acts* represents a universal action set, and then an LTS P is defined as a quaternion $P=<Q, \Sigma, \Delta, q>$ where:

- $Q \subseteq States$, representing the state set of P;
- $\Sigma=\alpha P(\alpha P \subseteq Acts)$, representing the action set of P;
- $\Delta \subseteq Q \times \Sigma \times Q$, representing the transition relationship in P, these transitions are labeled with the elements in Σ;
- $q \in Q$, representing the initial state of P.

An LTS P can be converted to LTS P' by action $a \in A$, denoted $P \xrightarrow{a} P'$, if $P' = < Q, \Sigma, \Delta, q' >$ and $(q, a, q') \in \Delta$.

We need to use the parallel composition of LTSs to express the interaction between multiple LTSs. The following gives the definition of LTS parallel composition.

Definition 1.2 Parallel composition of LTSs: The parallel composition of two *LTS* $M=< Q_1, \Sigma_1, \Delta_1, q_1 >$ and $N = < Q_2, \Sigma_2, \Delta_2, q_2 >$ are expressed as *LTS* $(M \parallel N) = < Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \Delta, (q_1, q_2)>$, where Δ is the minimum relation that satisfies the following constraint:

$$\frac{M \xrightarrow{a} M'}{M \| N \xrightarrow{a} M' \| N} \quad a \notin \alpha N \tag{1}$$

$$\frac{N \xrightarrow{a} N'}{M \| N \xrightarrow{a} M \| N'} \quad a \notin \alpha M \tag{2}$$

$$\frac{M \xrightarrow{a} M', N \xrightarrow{a} N'}{M \| N \xrightarrow{a} M' \| N'} \quad a \neq \tau \tag{3}$$

where $a \in \Sigma_1 \cup \Sigma_2$, τ represents an internal action that is not visible to the outside.

We start from the LTS model of the basic *non_control_model*, and gradually construct the LTS model of the four synchronization patterns.

Our basic idea is to treat an Event-B model as a composite LTS while treating each state variable of this Event-B model as an atomic LTS. Therefore, the LTS of Event-B model is the parallel composition of all atomic LTSs. The advantage of this idea is that, although the LTS model of an Event-B model is very complex, we can break it into some little atomic LTSs. Each atomic LTS corresponds to the state transition of a single variable which is easy to be modeled.

Our translation method is based on a basic fact in Event-B model, that is, the occurrence of an event will change the value of a variable. Thus, for each atomic LTS, we can mark the event that causes its state change to a transition edge in this LTS. The source node of this transition is the value of the variable before the event occurs, while the destination node is the value of the variable after the event occurs. In

this way, we get the following two translation rules:

Rule 1. Each variable in the Event-B model is modeled as an atomic LTS, and all possible values for this variable form the state space of this atomic LTS.

Rule 2. For each atomic LTS P, if an event *e* in the Event-B model changes the value of its corresponding variable from s_1 to s_2, then we add an element $(s_1 \xrightarrow{e} s_2)$ to the transition set of this atomic LTS.

4.1 LTS Model of non_control_model

According to Rules 1 and 2, we can easily get the LTS model of a_n and r_n in *non_control_model,* namely the LTS A_N and the LTS R_N :

LTS A_N=< *{0, 1}, {a_on,a_off},{*$(0 \xrightarrow{a\_on} 1), (1 \xrightarrow{a\_off} 0)$*},0*>

LTS R_N =< *{0, 1}, {r_on,r_off},{*$(0 \xrightarrow{r\_on} 1), (1 \xrightarrow{r\_off} 0)$*},0*>

The LTS model of *non_control_model* is equal to the parallel composition of LTS A_N and LTS R_N , namely LTS N. According to the parallel composition rules in **Definition 1.2**, we get:

LTS N=LTS(A_N || R_N*)=<* Q_N, Σ_N, Δ_N, q_N>, where:

Q_N= {(0, 0), (0, 1), (1, 1), (1, 0)}

Σ_N= {*a_on, a_off, r_on, r_off*}

Δ_N= $\{(0,0) \xrightarrow{a\_on} (1,0), (0,1) \xrightarrow{a\_on} (1,1), (1,1) \xrightarrow{a\_off} (0,1), (1,0) \xrightarrow{a\_off} (0,0),$

$(0,0) \xrightarrow{r\_on} (0,1), (1,0) \xrightarrow{r\_on} (1,1), (0,1) \xrightarrow{r\_off} (0,0), (1,1) \xrightarrow{r\_off} (1,0)\}$

q_N=(0,0).

The state transition diagram for the atomic LTS of a_n and r_n as well as composed LTS N is shown in Fig. 6. The "||" symbol in Fig. 6 is parallel composition operator.

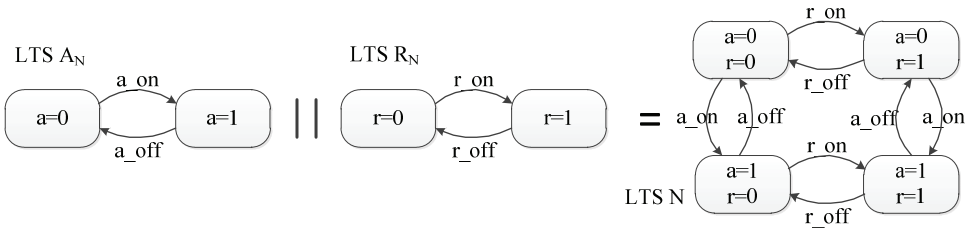


Fig. 6. LTS model of *non_control_model* and its atomic LTS.

4.2 LTS Model of weak_model

If there is no interaction between the atomic LTSs, that is, the change of one state variable in the Event-B model does not affect the change of another state variable, just like *non_control_model,* then the Rules 1 and 2 are sufficient to build the LTSs of Event-B model. But in the actual situation, this is usually not the case. Therefore, we also need to consider the impact of interaction in the Event-B model on the atomic LTSs.

In Section 3.1, we have already shown that the Event-B model of the *weak* synchronize pattern can be obtained by adding some guards to the *non_control_model* (e.g., add a guard "*a = 1*" for the *r_on* event of *non_control_model*). Now we need to map this change to the LTS of the actuator *a* and the reactor *r*.

However, there is currently no rule that can be applied to *r_on* because the *r_on* event does not change the value of *a*. Thus, we use a tricky approach to transform the *r_on* event into an *r_on′* event in which a "useless" action named *action2: a ≔ 1* is added:

```
r_on'      ≙
STATUS
Ordinary
WHEN
  @guard1:a=1
  @guard2:r= 0
THEN
  @action1:r ≔ 1
  @action2:a ≔ 1
END
```

It is obvious that *r_on* and *r_on′* has the same behavior semantics. They all do the same behavior, "change the value of variable *a* from 1 to 1". While this sounds ridiculous, it can help us to construct LTS through the code of *r_on*. Now, according to Rule 2, we can add a new transition $(1 \xrightarrow{r\_on} 1)$ to the transition set of atomic LTS of *a*. It should be noted that although *guard1*: *a = 1* appears in the guards of *r_on*, it describes the impact of event *r_on* on variable *a*, so the LTS we need to modify is the LTS of *a*.

After generalization of the above situation, we can get Rule 3.

Rule 3. Assume that *s* is a variable in an Event-B model; q_i is a possible value for *q*. If there is an expression "$q = q_i$" in the guards of event *e*, and the action of event *e* does not change the value of *q*, then we should add a transition $(q_i \xrightarrow{e} q_i)$ to the transition set of LTS of variable *q*.

Rule 3 is a special case of Rule 2. In the LTS transition graph, it means that a reflexive edge is added to the q_i state node in the LTS of *q*, and the edge is labeled with event *e*.

Now, according to Rule 3 and *weak_model*, we can easily get the LTS model of a_w and r_w in *weak* synchronization pattern, namely LTS A_w and LTS R_w:

LTS A_w=< {*0, 1*}, {*a_on,a_off,r_on,r_off*},{$(0 \xrightarrow{a\_on} 1), (1 \xrightarrow{a\_off} 0), (0 \xrightarrow{r\_off} 0), (1 \xrightarrow{r\_on} 1)$},*0*>

LTS R_w =< {*0, 1*}, {*r_on,r_off*},{$(0 \xrightarrow{r\_on} 1), (1 \xrightarrow{r\_off} 0)$},*0*>

The LTS model of *weak _model* is equal to the parallel composition of LTS A_W and LTS A_W, namely LTS W:

LTS $W=LTS(A_w||R_w)$= < Q_W, Σ_W, Δ_W, q_W>,where:

Q_W = {*(0, 0), (0, 1), (1, 1), (1, 0)*}

Σ_W= {*a_on,a_off,r_on,r_off*}

Δ_W= $\{(0,0) \xrightarrow{a\_on} (1,0), (0,1) \xrightarrow{a\_on} (1,1),(1,1) \xrightarrow{a\_off} (0,1), (1,0) \xrightarrow{a\_off} (0,0), (1,0) \xrightarrow{r\_on} (1,1),(0,1) \xrightarrow{r\_off} (0,0)\}$

q_W=(0,0).

The state transition diagram for the atomic LTS of a_w and r_w as well as composed LTS W is shown in Fig. 7.

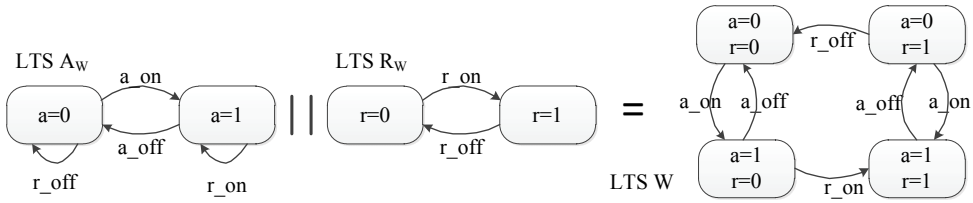


Fig. 7. LTS model of *weak_model* and its atomic LTS.

We can see that the LTS of the *weak* synchronization pattern is in fact a clipping of the LTS N. Compared to the transition set Δ_N, there are two transitions $((0,0) \xrightarrow{r\_on} (0,1)$ *and* $(1,1) \xrightarrow{r\_off} (1,0))$ removed from the transition set Δ_W, which is the effect of adding guards on events *r_on* and *r_off*.

4.3 LTS Model of strong_model

According to Rule 3 and *strong_model*, we continue to construct LTS models of a_s and r_s in strong synchronization patterns, named LTS A_S and LTS R_S:

LTS $A_S = <\{0, 1\}, \{a\_on,a\_off,r\_on,r\_off\},\{(0 \xrightarrow{a\_on} 1), (1 \xrightarrow{a\_off} 0), (0 \xrightarrow{r\_off} 0), (1 \xrightarrow{r\_on} 1)\},0>$

LTS $R_S = <\{0,1\},\{a\_on,a\_off,r\_on,r\_off\},\{(0 \xrightarrow{r\_on} 1), (1 \xrightarrow{r\_off} 0), (0 \xrightarrow{a\_on} 0), (1 \xrightarrow{a\_off} 1)\},0>$

The LTS model of *strong _model* is equal to the parallel composition of LTS A_S and LTS R_S , namely LTS S:

LTS $S=LTS(A_S \,||\, R_S) = < Q_S, \Sigma_S, \Delta_S, q_S>$,where:

$Q_S = \{(0,0),(0,1), (1,1) , (1,0) \}$

$\Sigma_S = \{a\_on,a\_off,r\_on,r\_off\}$

$\Delta_S = \{(0,0) \xrightarrow{a\_on} (1,0),\ (1,0) \xrightarrow{r\_on} (1,1)\ (1,1) \xrightarrow{a\_off} (0,1),(0,1) \xrightarrow{r\_off} (0,0) \}$

$q_S = (0,0)$.

The state transition diagram for the atomic LTS of a_s and r_s as well as composed LTS S is shown in Fig. 8.

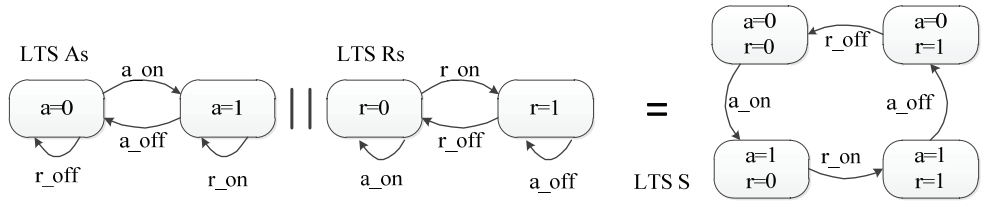


Fig. 8. LTS model of *strong_model* and its atomic LTS.

4.4 LTS Model of strong_weak_model

Strong_weak_model seems to be the most complex, but in fact, we can easily get LTS models of each variable according to the Rules 1 to 3. First, according to Rulex 1 and 2, we establish the atomic LTS

models of a_{sw}, r_{sw}, b_{sw} and s_{sw} respectively. Then we add the synchronization control transitions on these atomic LTSs according to Rule 3 and *strong_weak_model*. The final atomic LTS models are:

$LTS\ A_{SW}$=< {0,1},{a_on,a_off,r_on,r_off ,b_on},{(0 $\xrightarrow{a\_on}$ 1), (1 $\xrightarrow{a\_off}$ 0), (0 $\xrightarrow{r\_off}$ 0), (1 $\xrightarrow{r\_on}$ 1), (1 $\xrightarrow{b\_on}$ 1)},0>

$LTS\ R_{SW}$ =< {0,1},{a_on,a_off,r_on,r_off,b_on},{(0 $\xrightarrow{r\_on}$ 1), (1 $\xrightarrow{r\_off}$ 0), (0 $\xrightarrow{a\_on}$ 0), (1 $\xrightarrow{a\_off}$ 1), (1 $\xrightarrow{b\_on}$ 1)}, 0>

$LTS\ B_{SW}$=< {0, 1}, {b_on,b_off,s_on,s_off,a_off},
{(0 $\xrightarrow{b\_on}$ 1), (1 $\xrightarrow{b\_off}$ 0), (0 $\xrightarrow{a\_off}$ 0), (0 $\xrightarrow{s\_off}$ 0), (1 $\xrightarrow{s\_on}$ 1)}, 0>

$LTS\ S_{SW}$=<{0, 1}, {b_on,b_off,s_on,s_off,a_off},
{(0 $\xrightarrow{s\_on}$ 1), (1 $\xrightarrow{s\_off}$ 0), (0 $\xrightarrow{b\_on}$ 0), (0 $\xrightarrow{a\_off}$ 0), (1 $\xrightarrow{b\_off}$ 1)},0>

The LTS model of *strong_weak_model* is equal to the parallel composition of these four LTSs, namely LTS SW:

$LTS\ SW=LTS(A_{SW}||R_{SW}||B_{SW}||S_{SW})$= < Q_{SW}, Σ_{SW}, Δ_{SW}, q_{SW}>,where：

Q_{SW} = {(0,0,0,0),(0,1,0,0), (1,1,0,0),(1,0,0,0),(1,1,0,1),(1,1,1,1),(1,1,1,0) }

Σ_{SW} = {a_on,a_off,r_on,r_off,b_on,b_off,s_on,s_off }

Δ_{SW} = {(0,0,0,0) $\xrightarrow{a\_on}$ (1,0,0,0), (1,0,0,0) $\xrightarrow{r\_on}$ (1,1,0,0), (1,1,0,0) $\xrightarrow{a\_off}$ (0,1,0,0), (0,1,0,0) $\xrightarrow{r\_off}$ (0,0,0,0),
(1,1,0,0) $\xrightarrow{b\_on}$ (1,1,1,0), (1,1,1,0) $\xrightarrow{s\_on}$ (1,1,1,1), (1,1,1,1) $\xrightarrow{b\_off}$ (1,1,0,1), (1,1,0,1) $\xrightarrow{s\_off}$ (1,1,0,0)}

q_{SW} = (0,0,0,0).

The state transition diagram for the atomic LTS of a_{sw}, r_{sw}, b_{sw}, s_{sw} as well as composed LTS SW is shown in Fig. 9.

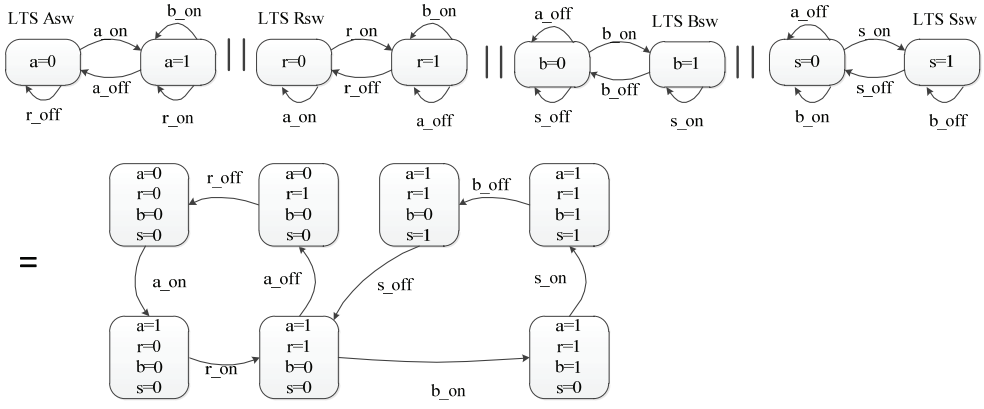


Fig. 9. LTS model of *strong_weak_model* and its atomic LTS.

4.5 LTS Model of strong_ strong_model

In Section 3.4, we already know that *strong_strong_model* is actually just adding some constraints on the basis of *strong_weak_model* (by adding a variable *m* to constrain the execution order of *a_on*, *b_on* and *a_off*). So we only need to add a new LTS M base on LTS SW to constrain it:

$LTS\ A_{SS}$ =$LTS\ A_{SW}$

$LTS\ R_{SS}$ =$LTS\ R_{SW}$

$LTS\ B_{SS}$ =$LTS\ B_{SW}$

$LTS\ S_{SS}$ =$LTS\ S_{SW}$

LTS M= < *{0, 1}, {a_on,b_on,a_off},{*(0 $\xrightarrow{a\_on}$ 1), (1 $\xrightarrow{b\_on}$ 0), (0 $\xrightarrow{a\_off}$ 0)*},0*>.

The state transition diagram for LTS M is shown in Fig. 10.

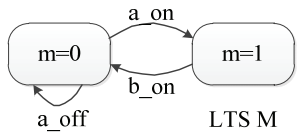


Fig. 10. LTS model of *m*.

The LTS model of *strong_strong _model* is equal to the parallel composition of LTS A_{SS}, LTS R_{SS}, LTS B_{SS}, LTS S_{SS}, and LTS M, namely LTS SS:

LTS SS=LTS(A_{SS} || R_{SS} || B_{SS} || S_{SS} ||*M*)= < Q_{SS}, Σ_{SS}, Δ_{SS}, q_{SS}>,where:

Q_{SS} = {*(0,0,0,0,0),(0,1,0,0,0), (1,1,0,0,0),(1,1,0,1,0),(1,1,1,1,0),(1,1,1,0,0), (1,1,0,0,1) ,(1,0,0,0,1)* }

Σ_{SS} = {*a_on,a_off,r_on,r_off,b_on,b_off,s_on,s_off* }

Δ_{SS} = {(0,0,0,0,0) $\xrightarrow{a\_on}$ (1,0,0,0,1)， (1,0,0,0,1) $\xrightarrow{r\_on}$ (1,1,0,0,1)， (1,1,0,0,1) $\xrightarrow{b\_on}$ (1,1,1,0,0),
(1,1,1,0,0) $\xrightarrow{s\_on}$ (1,1,1,1,0), (1,1,1,1,0) $\xrightarrow{b\_off}$ (1,1,0,1,0), (1,1,0,1,0) $\xrightarrow{s\_off}$ (1,1,0,0,0),
(1,1,0,0,0) $\xrightarrow{a\_off}$ (0,1,0,0,0), (0,1,0,0,0) $\xrightarrow{r\_off}$ (0,0,0,0,0) }

q_{SS} = (0,0,0,0,0).

The state transition diagram for LTS SS is shown in Fig. 11.

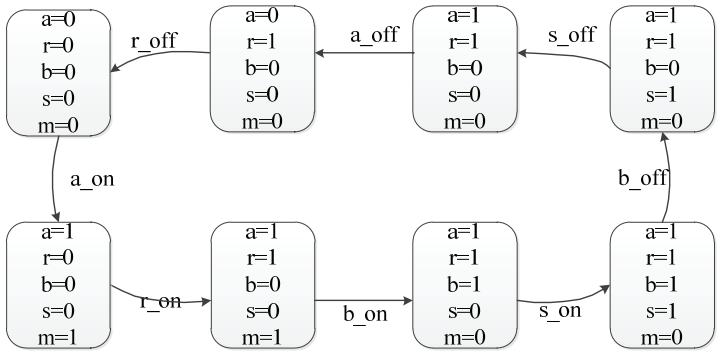


Fig. 11. LTS model of *strong_strong_model*.

5. Case Study

In this section, we apply the LTS semantic model of the four synchronous control flow design patterns of Event-B to the behavior model construction of a complex multi-level control system. We map the instantiation process of the Event-B model to the instantiation process of its corresponding LTS semantic model, and obtain the LTS semantic model of the Event-B model of the complex control system. On this basis, we use LTL to describe the behavior properties of the Event-B model and verify them. In this way, we can verify the behavioral properties of each refinement level of the Event-B model until the final

refinement model is verified. Since the Rodin platform (an Event-B modeling environment) provides a mechanism for refinement checking, we do not need to be concerned with the correctness of refinement, but only to verify the behavioral properties of each LTS model.

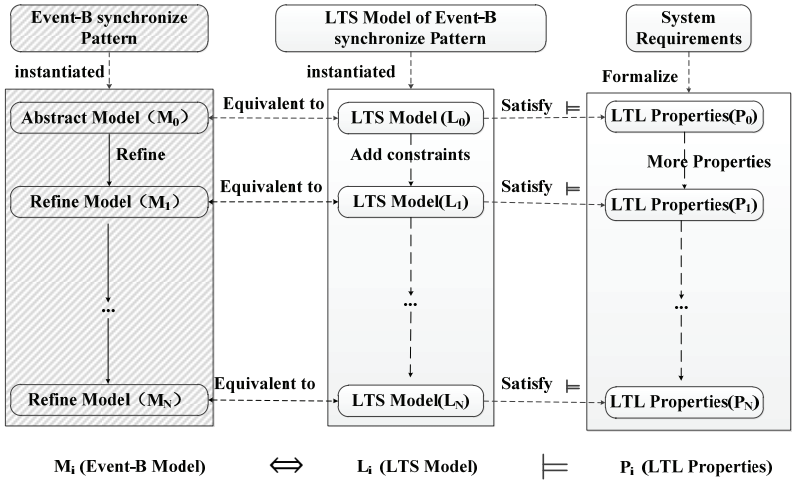


Fig. 12. The process of modeling and verification.

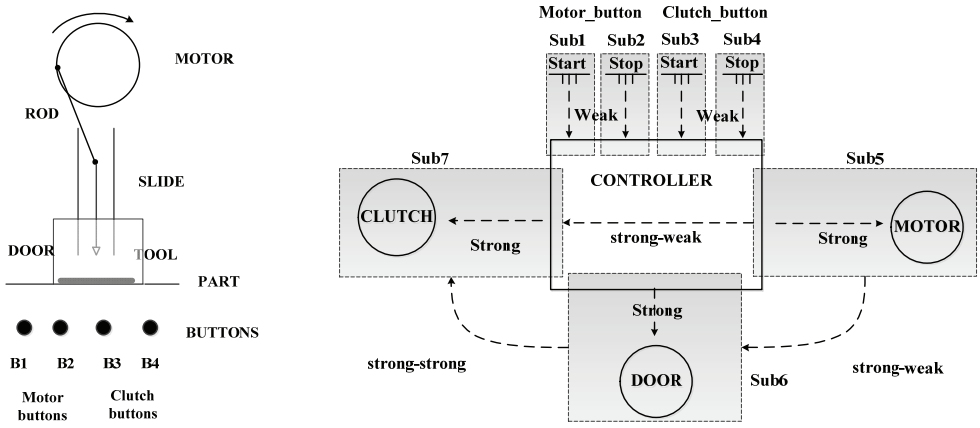
The modeling and verification processes are shown in Fig. 12. The leftmost part of the Fig. 12 is a common process to get the Event-B model of a complex control system by the instantiations of Event-B design patterns. The middle of Fig. 12 is the process of enhancing the constraints of the LTS according to the refinement process of the Event-B model. The rightmost part of Fig. 12 is the property verification process of the LTS model corresponding to the Event-B model. We guarantee that at each refinement level, there are $M_i \Leftrightarrow L_i$, and then verify the LTL properties of the LTS model, i.e. $L_i \models P_i$. In this way, we can guarantee $M_i \models P_i$. The formula at the bottom of Fig. 12 is a formal representation of this conversion and verification process.

5.1 System Description

We use the complex control system design patterns in the study of Abrial [1] to illustrate our approach. The system is called the mechanical press control system. The system has four control buttons, which include two motor control buttons and two clutch control buttons to start and stop the system components. After the motor is started, if the clutch is engaged, the rod will drive the slide up and down to use the tool to process the part. A door is used to prevent the worker from putting his hand under the tool while processing the part. The physical composition of the system is shown in Fig. 13(a).

The complete internal composition of the system and the synchronization control relationship between the components are shown in Fig. 13(b).We use the dotted line shadow box to represent the subsystem. For example, there is a "trigger-response" relationship between the "start motor button" and the "controller", so we put them into a subsystem, that is, Sub1. At the same time, we use the dotted arrows to express the "trigger-response" relationship between the components or subsystems. The starting point of the arrow indicates the actuator component or the actuator subsystem, and the end point of the arrow represents the reactor part or the reactor subsystem. For example, the arrow with the "start motor button"

as the starting point and the "controller" as the end point indicate that in the subsystem 1, the "start motor button" is the actuator and the "controller" is the reactor. We marked the type of synchronization relationship on the arrow. For example, the relationship between Sub5 and Sub7 is a strong - weak synchronization relationship.



(a) (b)

Fig. 13. The mechanical press control system: (a) the physical structure of the press system and (b) the synchronization relationship in the press system.

Abrial [1] used the instantiation methods to progressively introduce the various subsystems and various synchronic control flow relationships into the model during the refinement process and ended this process at the seventh refinement layer. The shaded portion of Fig. 12 represents this refinement process. We need to complete the process shown in the middle part and the rightmost part of Fig. 12.

5.2 Instantiation Process of LTS Model

The instantiation process from the Event-B design pattern to the Event-B system model is a very simple renaming process, as we will see in Table 1. The principle behind this process is that modifying the variable name, constant name, and event name will not affect the correctness and refinement consistency of the system. We apply this idea to the LTS behavior semantic model of Event-B design pattern, and get the LTS model of Event-B system model by instantiating the LTS behavior semantic model of Event-B design pattern. The instantiation process of the LTS model is also very simple, that is, renaming the name of the state, conversion and atomic LTS of LTS behavior semantics model. It is obvious that the instantiation of the LTS behavior semantic model does not change the behavior of the LTS model itself, either.

We use an example of *strong* synchronization pattern to illustrate the correspondence between the process of instantiation of Event-B design pattern and the process of instantiation of the LTS behavior semantic model of Event-B design pattern.

The relationship between the controller and the motor in Sub5 in Fig. 13 are a strong synchronization relationship. Abrial [1] instantiates the Event-B design pattern *strong_model* in Section 3.2 into the Event-B model of Sub5, namely *controller_motor_model*. Corresponding to this process, we instantiate the atomic LTSs model of the *strong* synchronization pattern, that is LTS A_S and LTS R_S in Section 4.3, into

their LTS model, namely *Motor_actuator0* and *Motor_sensor0*, respectively, then perform the parallel composition operation on them to get the LTS behavior semantic model of *controller_motor_model*, named LTS Press0. The correspondence between these two processes is shown in Table 1. The instantiation process of Event-B design pattern is just to rename the elements of column 2 of Table 1 to the elements of column 3 of Table 1. Similarly, the instantiation process of LTS is just to rename the elements of column 5 of Table 1 to the elements of column 6 of Table 1.

Table 1. The corresponding relationship between Event-B design pattern instantiation and LTS model instantiation

	Instantiation of Event-B design pattern			**Instantiation of LTS model**	
	strong_model	**controller_motor_model**		**LTS S**	**LTS Press0**
Variables	a	motor_actuator	LTS	a	motor_actuator0
	r	motor_sensor		r	motor_sensor0
Constants	0	stopped	States	0	0
	1	working		1	1
Events	a_on	treat_start_motor	Transitions	a_on	treat_start_motor
	a_off	treat_stop_motor		a_off	treat_stop_motor
	r_on	motor_start		r_on	motor_start
	r_off	motor_stop		r_off	motor_stop

The *controller_motor_model* of Sub5 obtained by instantiation of *strong_model* is:

treat_start_motor ≙ STATUS Ordinary WHEN @guard1:motor_actuator = stopped @guard2:motor_sensor= stopped THEN @action1:motor_actuator := working END	treat_stop_motor ≙ STATUS Ordinary WHEN @guard1:motor_actuator = working @guard2:motor_sensor= working THEN @action1:motor_actuator := stopped END
motor_start ≙ STATUS Ordinary WHEN @guard1:motor_actuator = working @guard2:motor_sensor= stopped THEN @action1:motor_sensor:= working END	motor_stop ≙ STATUS Ordinary WHEN @guard1:motor_actuator = stopped @guard2:motor_sensor= working THEN @action1:motor_sensor:= stopped END

The LTS behavior model of *controller_motor_model* is the parallel composition of the two LTSs in Fig. 14, that is, LTS Press0 shown in Fig. 15.

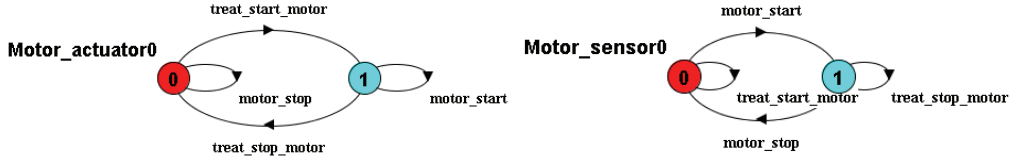


Fig. 14. Atomic LTSs of LTS Press0.

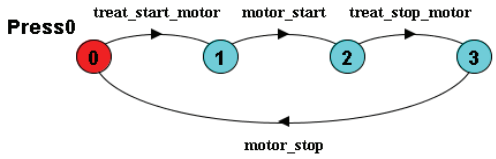


Fig. 15. LTS Press0.

We use this method to convert each layer of the Event-B model of mechanical press control system into the corresponding LTS semantic model. In the last layer, that is, the seventh layer, we get 15 atomic LTSs, of which 14 are shown in Table 2. The LTS that not be shown in Table 2 is the LTS M in Section 4.5, which is used to model the *strong-strong* synchronization pattern.

The final system LTS model, named LTS Press7, is the parallel composition of the above 15 LTSs, that is:

LTS Press7=LTS Sub1||LTS Sub2||LTS Sub3||LTS Sub4||LTS Sub5||LTS Sub6||LTS Sub7||LTS M, where:

LTS Sub1 =start_motor_button||start_ motor_impulse;

LTS Sub2 =stop_motor_button||stop_ motor_impulse;

LTS Sub3 =start_clutch_button||start_clutch_impulse

LTS Sub4 =stop_clutch_button||stop_clutch_impulse

LTS Sub5 =motor_actuator||motor_sensor

LTS Sub6 =door_actuator||door_sensor

LTS Sub7 =clutch_actuator||clutch_sensor

Table 2. The atomic LTS models of mechanical press control system

Subsystem LTS	LTS model		Synchronization relationship
	Actuator LTS	Reactor LTS	
Sub1 (B1-Controller)	start_motor_button	start_ motor_impulse	Weak synchronization
Sub2 (B2-Controller)	stop_motor_button	stop_ motor_impulse	Weak synchronization
Sub3 (B3-Controller)	start_clutch_button	start_clutch_impulse	Weak synchronization
Sub4 (B4-Controller)	stop_clutch_button	stop_clutch_impulse	Weak synchronization
Sub5 (Controller-Motor)	motor_actuator	motor_sensor	Strong synchronization
Sub6 (Controller-Door)	door_actuator	door_sensor	Strong synchronization
Sub7 (Controller-Clutch)	clutch_actuator	clutch_sensor	Strong synchronization

5.3 Verification of the Behavior Properties

Now we can complete the rightmost process in Fig. 12, that is, the behavior properties verification of the LTS model. Due to space limitations, we only give behavior properties verification process of the final model Press7. Abrial [1] specifies the basic requirements for the mechanical press control system, as shown in Table 3.

Table 3. The basic requirements for the mechanical press control system

Requirement No.	Requirement description
FUN_1	When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE
FUN_2	When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE
SAF_1	When the clutch is engaged, the motor must work
SAF_2	When the clutch is engaged, the door must be closed

Table 4. The LTL expressions of basic requirements

Requirement No.	LTL expressions
FUN_1	□(DOOR_CLOSED → □(DOOR_CLOSED U¬CLUTCH_ENGAGED)
FUN_2	□(¬CLUTCH_ENGAGED → □(¬CLUTCH_ENGAGED U DOOR_CLOSED)
SAF_1	□(CLUTCH_ENGAGED → MOTOR_WORKING)
SAF_2	□(CLUTCH_ENGAGED → DOOR_CLOSED)

We write LTL expressions according to the requirements in Table 3, as shown in Table 4. The symbol "□"and "U" in Table.4 represent "always" and "Until" in linear temporal logic, respectively, while the symbols "¬" and "→" represent "negation" and "implication" in proposition logic, respectively.

We verified the LTL behavior properties of LTS model of each refinement layer of the Event-B model and find that the system model at some intermediate layers does not satisfy the required behavior properties. But in the last layer, that is, the seventh layer, all of the behavioral properties are satisfied. This is because Event-B's refinement strategy does not take into account the preservation of behavioral properties. It also suggests that our method is necessary.

6. Experimental Results and Discussion

In this section, we compare the results of our work with previous research in three aspects: explicit expression of control flow, model simulation and analysis capabilities, and support for LTL properties verification.

6.1 Comparison of Explicit Expression Ability of Control Flow

We show the LTS model of strong-weak synchronization pattern in Fig. 16. As can be seen from Fig. 16, while the event order of the Event-B design pattern given in Section 4 seems to be complicated, it is easier to observe the event order in its LTS model. Compared with the way of expression of LTS model, the CSP||B method uses the textual form of "Process P-> action. Process Q" to express the sequence of events. The advantage of this is that the code is concise, but also causes the order of its actions to be invisible; "atomic decomposition" method and Flow methods also use a graphical way to describe the relationships between events. However, the main advantage of the "atomic decomposition" method is to express the relationship between abstract and concrete events, not the event order. We must analyze its tree structure carefully to understand the actual control flow of Event-B model; The Flow method uses a

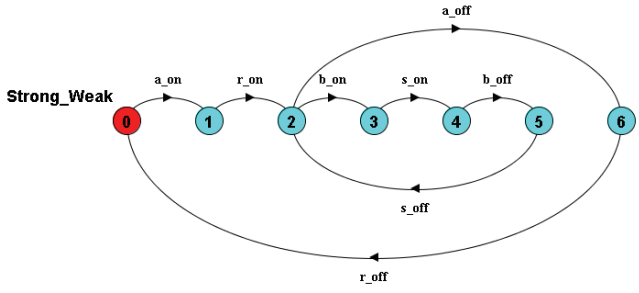


Fig. 16. The LTS model of *strong - weak* synchronization pattern.

style like "event *a* enable/disable event *b*" to express the relationship between events, which does not conform to the traditional way of expression of control flow (i.e., the state transition system style).

6.2 Comparison of Model Simulation and Analysis Capabilities

We counted the relevant data of the Event-B model of the mechanical press control system, as shown in Table 5 and Fig. 17. At the same time, we presented the statistics of the LTS model of the mechanical press control system, as shown in Table 6 and Fig. 18. Since the LTS model is just the behavior model of the Event-B model, the growth of state and transitions in the LTS model reflect the actual effect of the addition of guards and actions to the Event-B model. By comparing the graphs, we can see that although the Event-B model has a small amount of code (only 348 lines of Event-B code in the 7th refinement layer), the 74 guards and 49 actions in these codes will produce 2816 states and 16384 transitions into the corresponding LTS model. If we use Rodin to simulate and analyze the transition of the system state, that would be a nightmare. The CSP||B method, the "atomic decomposition" method and the Flow method, can only generate Event-B code and use the similar method (in Rodin environment) to complete the simulation. These methods do not by themselves support the simulation of control flow. Compared to these approaches, we model the system as a series of separated components using the concept of "atomic LTS" which enable people to observe the individual behavior of each component and the interaction between components during system execution, just like in Uppaal [25].

Table 5. Statistical data of mechanical press control system Event-B model

Refinement level	Elements of Event-B press model		
	Variables	Guards	Actions
Press0	2	8	6
Press1	6	24	20
Press2	8	32	26
Press3	8	36	26
Press4	10	44	32
Press5	10	52	32
Press6	11	54	35
Press7	15	74	49

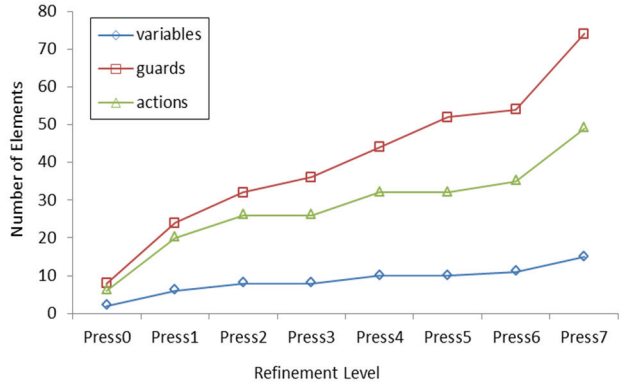


Fig. 17. Statistical data of mechanical press control system Event-B model.

Table 6. Statistical data of mechanical press control system LTS model

Refinement level	Elements of LTS press model		
	Sub LTSs	States	Transitions
Press0	2	4	4
Press1	6	64	200
Press2	8	256	1056
Press3	8	112	384
Press4	10	448	1984
Press5	10	160	568
Press6	11	176	608
Press7	15	2816	16384

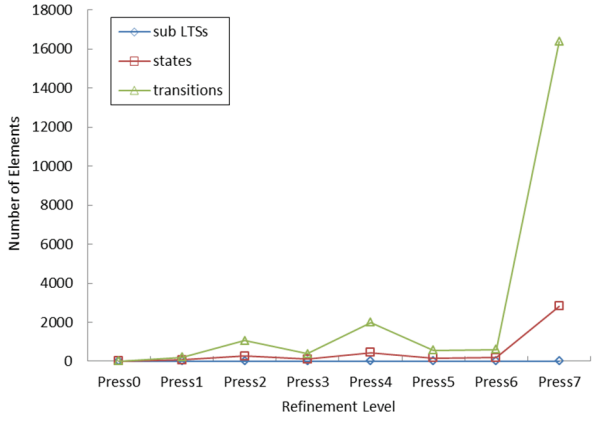


Fig. 18. Statistical data of mechanical press control system LTS model.

6.3 Comparison of Support for LTL Property Verification

The results of the "atomic decomposition" method and the Flow method cannot be directly converted to LTS models; therefore it is difficult for them to support the verification of LTL properties. The CSP||B method is not perfect because the changes of many state variables have not been modeled with CSP. Compared to the CSP||B method, we model the behavior of the entire Event-B model as an LTS model. Therefore, our behavior modeling method is more comprehensive than the CSP||B method, which makes the result of our method for LTL properties verification more credible than that of CSP||B.

The drawback of LTS model is that it does not support the refinement checking. But we can use Event-B to prevent this flaw. The complementary capabilities of Event-B and LTS allow us to analyze and verify the system's refinement consistency and behavioral properties from multiple views. This is the idea of the integrated formal method that we are currently advocating.

7. Conclusion

As a reusable formal model, the Event-B design pattern has a very important role for the rapid formal modeling of the system. But it is difficult to model, analyze, verify and preserve the LTL behavior properties of the Event-B model because it lacks behavioral semantics. In this paper, we establish the LTS model of Event-B synchronous control flow design pattern and map the design pattern instantiation

process of Event-B to the instantiation process of LTS model. We get the LTS model of a complex multi-level control system using this method and verify its LTL behavior attributes. The experimental data show that it is more convenient to model, analyze and verify the LTL behavior properties of Event-B model if we use LTS as behavior semantic model of Event-B. The LTS behavior semantic model can be a useful complement to the Event-B model.

In the future, we will study how to establish the refinement checking mechanism of the LTS model to ensure the behavior refinement of the Event-B model, which is lacking in the current refinement checking mechanism of Event-B.

References

[1] J. R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge, UK: Cambridge University Press, 2010.

[2] T. S. Hoang, A, Furst, and J. R. Abrial, "Event-B patterns and their tool support," *Software & Systems Modeling*, vol. 12, no. 2, pp. 229-244, 2013.

[3] I. Tounsi, M. H. Kacem, A. H. Kacem, K. Drira, and E. Mezghani, "Towards an approach for modeling and formalizing soa design patterns with Event-B," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, Coimbra, Portugal, 2013, pp. 1937-1938.

[4] A. Gondal, M. Poppleton, and M. Butler, "Composing Event-B specifications: case-study experience," in *Software Composition*. Heidelberg: Springer, 2011, pp. 100-115.

[5] A. Gondal, "Feature-oriented reuse with Event-B and Rodin," Ph.D. dissertation, University of Southampton, UK, 2013.

[6] A. Intana, "Formal engineering methodologies for wireless sensor network development with simulation," Ph.D. dissertation, University of Southampton, UK, 2015.

[7] R. Silva, "Application of decomposition and generic instantiation," 2011; https://eprints.soton.ac.uk/272195/.

[8] R. Silva and M. Butler, "Supporting reuse of Event-B developments through generic instantiation," in *Formal Methods and Software Engineering*. Heidelberg: Springer, 2009, pp. 466-484.

[9] S. Yeganefard, M. Butler, and A. Rezazadeh, "Evaluation of a guideline by formal modelling of cruise control system in Event-B," in *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, Washington, DC, 2010, pp. 182-191.

[10] S. Yeganefard and M. Butler, "Problem decomposition and sub-model reconciliation of control systems in Event-B," in *Proceedings of 2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, San Francisco, CA, 2013, pp. 528-535.

[11] S. Yeganefard and M. Butler, "Structuring functional requirements of control systems to facilitate refinement-based formalisation," *Electronic Communications of the EASST*, vol. 46, pp. 1-15, 2011.

[12] S. Yeganefard and M. Butler, "Control systems: phenomena and structuring functional requirement documents," in *Proceedings of 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, Paris, France, 2012, pp. 39-48.

[13] A. S. Fathabadi, M. Butler, and A. Rezazadeh, "A systematic approach to atomicity decomposition in Event-B," in *Software Engineering and Formal Methods*. Heidelberg: Springer, 2012, pp. 78-93.

[14] E. Alkhammash, M. Butler, A. S. Fathabadi, and C. Cirstea, "Building traceable Event-B models from requirements," *Science of Computer Programming*, vol. 111, pp. 318-338, 2015.

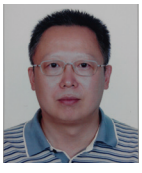
[15] A. S. Fathabadi and M. Butler, "Applying Event-B atomicity decomposition to a multi media protocol," in *Formal Methods for Components and Objects*. Heidelberg: Springer, 2009, pp. 89-104.

[16] A. S. Fathabadi, A. Rezazadeh, and M. Butler, "Applying atomicity and model decomposition to a space craft system in Event-B," in *NASA Formal Methods*. Heidelberg: Springer, 2011, pp. 328-342.

[17] A. Iliasov, "Use case scenarios as verification conditions: Event-B/Flow approach," in *Software Engineering for Resilient Systems*. Heidelberg: Springer, 2011, pp. 9-23.
[18] S. Schneider, H. Treharne, and H. Wehrheim, "A CSP approach to control in Event-B," in *Integrated Formal Methods*. Heidelberg: Springer, 2010, pp. 260-274.
[19] S. Schneider, H. Treharne, and H. Wehrheim, "Bounded retransmission in Event-B || CSP: a case study," *Electronic Notes in Theoretical Computer Science*, vol. 280, pp. 69-80, 2011.
[20] S. Schneider, H. Treharne, and H. Wehrheim, "Stepwise refinement in Event-B CSP. Part 1: Safety," Department of Computing, University of Surrey, UK, 2011.
[21] S. Schneider, H. Treharne, and H. Wehrheim, "The behavioural semantics of Event-B refinement," *Formal Aspects of Computing*, vol. 26, no. 2, pp. 251-280, 2014.
[22] S. Schneider, H. Treharne, H. Wehrheim, and D. M. Williams, "Managing LTL properties in Event-B refinement," in *Integrated Formal Methods*. Cham: Springer, 2014, pp. 221-237.
[23] T. S. Hoang, S. Schneider, H. Treharne, and D. M. Williams, "Foundations for using linear temporal logic in Event-B refinement," *Formal Aspects of Computing*, vol. 28, no. 6, pp. 909-935, 2016.
[24] R. Gorrieri, "Labeled transition systems," in *Process Algebras for Petri Nets*. Cham: Springer, 2017, pp. 15-34.
[25] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134-152, 1997.

Han Peng https://orcid.org/0000-0001-8400-4663

He received both the Bachelor's degree in Computer Engineering and the Master's degree in Computer Engineering from the Xidian University in 2004 and 2007, respectively. He is now PhD candidate student in the Northwestern Polytechnical University. He is now an associate professor at Xi'an Aeronautical University. His present research interests include software engineering, embedded system, formal modeling, and model checking.



Chenglie Du https://orcid.org/0000-0002-1042-9228

He received both the Bachelor's degree in Computer Science and the Master's degree in Computer Science from the Northwestern Polytechnical University in 1991 and 1994, respectively. In January 1999, he also received the Ph.D. degree in Automation Engineering from the same institution. Presently, he is a Professor of Computer Engineering at the Department of Computer Science, Northwestern Polytechnical University. His present research interests include software engineering, distributed computing, embedded computing, and cyber-physical System.

Lei Rao https://orcid.org/0000-0001-6832-6347

He received his Bachelor's degree in 2017 and he is studying at the School of Software and Microelectronics of Northwestern Polytechnical University for a Master's degree. His current research interests include formal modeling, model checking and so on.

Zhouzhou Liu https://orcid.org/0000-0001-7532-9749

He received both the Bachelor's degree in Information Engineering and the Master's degree in Telecommunication Engineering from the Northwestern Polytechnical University in 2004 and 2007, respectively. In April 2016, he also received the Ph.D. degree in Information Engineering from the same institution. He is now a professor at Xi'an Aeronautical University. His present research interests include intelligent computing, mathematical modeling, communication and signal processing applications in wireless sensor networks.