Hindawi
Journal of Control Science and Engineering
Volume 2018, Article ID 1468172, 12 pages
https://doi.org/10.1155/2018/1468172

Research Article

Modelling the Embedded Control System Using iUML-B Pattern State Machine

Han Peng,[1] **Chenglie Du,**[1] **Lei Rao,**[2] **and Zhouzhou Liu**[1]

[1] *School of Computer Science, Northwestern Polytechnical University, Xi'an, China*
[2] *School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an, China*

Correspondence should be addressed to Han Peng; hansbeng2016@gmail.com

Received 15 November 2017; Revised 16 April 2018; Accepted 10 May 2018; Published 12 June 2018

Academic Editor: Carlos-Andrés García

Developing the formal model based on the Event-B design pattern is an excellent method to improve the development efficiency of the embedded control system and improve the reusability of the formal model. However, the instantiation of the Event-B design pattern requires the manual writing of a large number of model codes, which brings a great deal of learning cost and coding burden to the engineering staff. In this paper, we propose a modelling approach for formal development of control systems based on the application of iUML-B state machine patterns to model the four synchronization patterns of the typical control system. Then, we use the instantiation of iUML-B pattern state machine to establish a typical multilevel control system's Event-B model. The simulation results show that the event trace of the model obtained using our method is the same as that of the corresponding model obtained using the traditional Event-B design pattern. Compared with the traditional Event-B design pattern method, our method can greatly reduce the manual coding burden in the modelling process. The system model expressed using the iUML-B pattern state machine can be easily mapped to the labelled transition system so as to verify the behavioural properties of the model.

1. Introduction

The embedded control system has been widely used in aviation, aerospace, Internet of things, and cyber-physical system. Due to the complexity of the embedded control system, it is difficult to ensure its safety properties through test and simulation. Therefore, in a variety of safety standards, such as DO-178C [1] and IEC 61508 [2], it has been clearly stated that engineer must use formal methods to model and verify the embedded system.

Event-B [3] is a formal modelling language based on set theory and first order logic which is supported by a modern tool and built-in provers. There are a lot of successful cases of Event-B application including aircraft landing gear control system [4] and satellite communication system [5].

To improve the reusability of the Event-B model, the researchers proposed the concept of Event-B design patterns [6]. Unlike the software design pattern, the Event-B design pattern is a formal design pattern. The goal of Event-B design patterns is to instantiate some small formal models that have been proven to be correct into actual system models and use these small formal models to construct a larger formal model by model composition. This method not only avoids "reinvent the wheel" but also avoids repetitive refinement and proof.

However, according to our experience, one has to add a lot of guards and actions manually if he (she) wants to model the synchronization control flow of control-intensive system using Event-B. It is a heavy work for the modeller. Engineers need an intuitive, simple way to understand and model synchronous control flow patterns.

The main contribution of this paper is to show how to use iUML-B pattern state machine to model the synchronization patterns of the control-intensive embedded system and apply it to the Event-B model of the embedded control system. We proposed the concept of iUML-B pattern state machine and developed the pattern state machines of four synchronization patterns (strong synchronization, weak synchronization, strong-strong synchronization, and strong-weak synchronization) for the control-intensive embedded system. Then we establish a formal model of a typical embedded control system by instantiating these four pattern state machines. The experimental results show that, using our proposed method, the modeller can establish a model of the control-intensive embedded system easily according to some simple rules.

The remainder of this paper is organized as follows: Section 2 describes the related works of the Event-B design pattern. Section 3 introduces the basic knowledge of Event-B and its design patterns and iUML-B state machines. Section 4 uses the iUML-B pattern state machine to model the synchronization patterns of the embedded control system. In Section 5, we instantiated iUML-B pattern state machines into the Event-B model of the embedded control system. In Section 6, we present an evaluation of our approach. Section 7 summarizes the work of this paper and looks ahead to future work.

2. Related Work

Event-B design pattern has been widely used in many fields. Renato Silva [7, 8] proposed the design pattern and the "Generic Instantiation" approach and uses this technology to develop the Event-B model of a safety critical subway system. In the instantiation process of the design pattern, the "Generic Instantiation" method can use the rename plug-in to instantiate the design pattern, thus avoiding a lot of repetitive developments and proofs.

Abrial [3] proposed four synchronization patterns of embedded control system: strong synchronization, weak synchronization, strong-weak synchronization, and strong-strong synchronization pattern. Its main purpose is to model the "actuator-reactor" patterns of the reactive system. Abrial modelled a mechanical press controller model using these four synchronization patterns. We will introduce these patterns in detail in Section 4. Synchronization pattern is a template for control-intensive embedded systems. It makes modelling of multilayer control systems easy. The modeller needs to instantiate the patterns into the specific model and then composes them together to get a complex multilevel control system model.

Sanz Yeganefard [9] applied the monitored, controlled, mode, and commanded (MCMC) method to the control system and proposed four patterns, namely, the *monitor* pattern, *control* pattern, *mode* pattern, and *command* pattern, expressed by Event-B. Sanaz Yeganefard developed the Event-B model of the cruise control system [10], the automotive lane departure warning system [11], and the lane centering controller [12] using the four above-mentioned patterns. In the work of Sanaz Yeganefard, the composition of patterns is proposed to compose the simple patterns into a composition pattern. But the composition of patterns requires tool support. *MCMC* patterns can be considered as an extended "*actuator-reactor*" pattern because they add status monitoring to the "*actuator-reactor*" pattern to form a feedback loop. This is very useful for the development of the formal model of self-adaptive systems.

In addition to the typical Event-B design patterns described above, Ali Gondal [13] proposed some Event-B refinement patterns and decomposition/composition patterns to model the product line of feature-oriented control systems. Adisak Intana [14] proposed some Event-B refinement and composition patterns to model wireless sensor networks.

To the best of our knowledge, there is no literature on using iUML-B state machine to model the Event-B synchronous control flow patterns.

3. Preliminaries

3.1. Event-B Model. An Event-B model consists of two parts, machine and context. Context describes the static part of the system, including carrier sets, constants, axioms, and theorems. A machine uses variables and events to describe the changes of the system. The process of developing a system model with Event-B usually begins with an abstract model and then continues to refine the model until it approaches the implementation. In the process of machine refinement, new events and variables can be added step by step. The current development tool for Event-B is the mature Rodin platform [15].

In Event-B, an event is made up of the guards and actions parts. An event can usually be expressed as

$$\boxed{e \triangleq \ \textbf{WHEN}\ \textit{\textbf{guards}}\ \textbf{THEN}\ \textit{\textbf{actions}}\ \textbf{END}}$$

When the guards of an event are all satisfied, the event can be triggered and the expression in actions parts describes the change of state variables when an event occurs.

3.2. iUML-B State Machine and Its Event Link Function. Snook and Butler invented a "UML-like" Event-B graphical front end, called UML-B [16]. UML-B uses the class diagrams and state diagrams familiar to the software engineers and system engineers to model the system requirements. System model expressed by UML-B can generate the corresponding Event-B machine directly on the Rodin platform. Recently, UML-B has evolved into iUML-B (integrated UML-B), which allows UML-B class diagrams and state diagrams to be embedded directly into an Event-B machine. iUML-B has been successfully applied to some large projects in Europe commission, such as [17–19].

The graphical symbol of the iUML-B state machine is similar to the state diagram of the UML, and we need not explain it. What we are interested in is a very powerful feature of the iUML-B state machine; that is, the transition edges in iUML-B state machine can be "linked" with the existing events in the Event-B model to control the order of these events. For example, suppose that we have already written four events: *INITIALISATION, a_on, a_off,* and *b_on*. If we want to control the order of these events like "after the *a_on* event has occurred, the *b_on* event must occur before the *a_off* event (*Req 1*)," then we can create an iUML-B state machine and "link" these four events to the transition edges of the state machine, as shown in Figure 1. The italic codes in Figure 1 are the codes automatically generated by the iUML-B state machine to control the event order.

We use the definition of "event trace" proposed by Butler [20] to describe the order of events in the Event-B model: "event trace represents a record of a possible execution trace

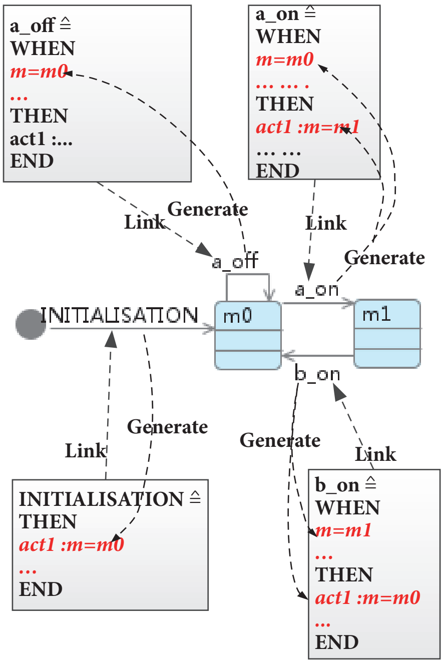


Figure 1: iUML-B state machine of Req 1.

of the model," and we use '$*$' to indicate an event that may occur 0 or more times. An event trace is usually represented by a set of events between "$\langle$" and "$\rangle$." For example, $\langle e1, e2\rangle*$ indicates that the $e2$ event must be executed after the $e1$ event occurs, and this event sequence will be repeated indefinitely.

3.3. Event-B Design Pattern and Its Instantiation. 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. As with the design pattern in software engineering, the Event-B design pattern is an abstract model of a class of problems. For example, the classic "trigger-response" pattern can be expressed in a nonformalized language: "once the trigger event *a_on* occurs, then the response event *r_on* will be enabled." Its corresponding Event-B design pattern is shown in Figure 2(a) (we named it *P1*). Suppose that when the system is initialized, the values of *a* and *r* are *a0* and *r0*, respectively. Then *r_on* event will be enabled after the *a_on* event occurs because the *a_on* event causes the value of variable *a* to change from *a0* to *a1*, which makes all the guards of *r_on* become *TRUE*.

We can express the "trigger-response" relationship in the specific control system by instantiating the pattern P1. For example, in an actual mechanical control system, there is a relationship between a motor controller button and a motor indicator light: "when the motor controller button is pressed, the motor indicator light must be illuminated." This relationship is an instance of the *P1* pattern. To get this instance, we just need to rename the variables, constants, and event names of the pattern *P1* according to renaming rules in Figure 2(b). The resulting Event-B model is shown in Figure 2(c) (we named it *I1*).

In this way, the modeller does not have to prove the correctness of *I1* again if he has proven the correctness of *P1*. 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.

4. Modelling Synchronous Patterns of the Embedded Control System

In this section, we use the iUML-B pattern state machine to model four typical synchronization patterns in the embedded control system, namely, strong synchronization pattern, weak synchronization pattern, strong-weak synchronization pattern, and strong-strong synchronization pattern. First, we explained in detail four synchronization patterns, which were proposed by Abrial in his book [3]. Then we modelled these four synchronization patterns with the iUML-B state machine.

We verify the correctness of our pattern state machine in three steps. First, we compare the Event-B code generated by our pattern state machine and that of the corresponding pattern proposed by Abrial (http://deploy-eprints.ecs.soton.ac.uk/113/3/ch3_pattern.zip). Then we use their corresponding labelled transitions system model to prove that they are equivalent in behaviour. Finally, we compared the event trace of our model with that of Abrial's Event-B model using the Rodin platform. Simulation results show that the event traces of them are identical.

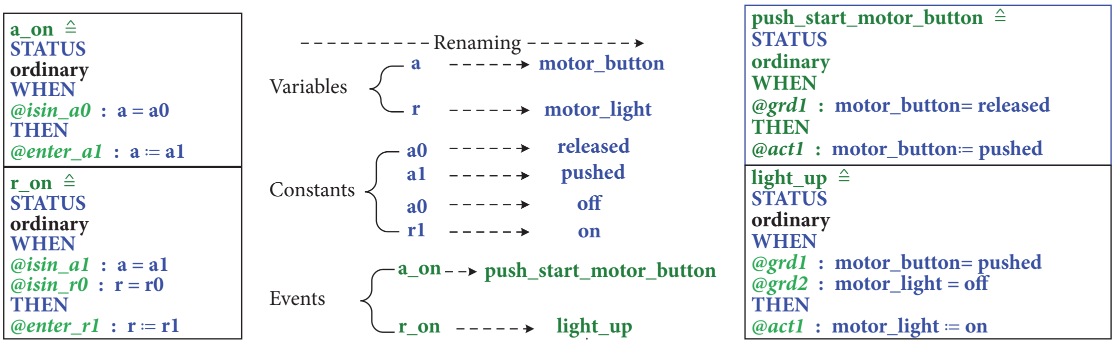
4.1. Synchronization Requirements of Control System. In the embedded control system, the "trigger-response" problem is the simplest and most basic model. In this model, the actuator executes an "*action*" event and sends an instruction to the reactor. The reactor receives the instruction after the "*action*" event and executes the "*reaction*" event, as shown in Figure 3.

In this paper, we follow the following naming convention.

The variables a and b represent the state changes of the actuators a and b, respectively. The variables r and s represent the state changes of the reactors r and s, respectively. The event *X_on* changes the value of the variable X from 0 to 1, where $X \in \{a, b, r, s\}$. The event *X_off* changes the value of the variable X from 1 to 0, where $X \in \{a, b, r, s\}$.

In addition, we refer to an "actuator-reactor" combination as a "subsystem." For instance, actuator a and reactor r compose a subsystem. We use "uninterruptible trace" to indicate an event trace that cannot be inserted in any other events. For example, if we require $\langle e1, e2, e3\rangle$ to be an uninterruptible trace, then event trace $\langle e1, e2, e4, e3\rangle$ is an illegal trace that deviated from our requirement.

(1) Strong Synchronization Requirement. A strong synchronization requirement means that once the actuator performs an action, the reactor must respond to it; otherwise, the actuator will not execute any further action. Figure 4 shows this strong synchronization relationship; that is, after *a_on* occurs, *r_on* must be executed; otherwise, the actuator will wait forever. Similarly, after *a_off* occurs, *r_off* must be



```
a_on ≙
STATUS
ordinary
WHEN
@isin_a0 : a = a0
THEN
@enter_a1 : a ≔ a1

r_on ≙
STATUS
ordinary
WHEN
@isin_a1 : a = a1
@isin_r0 : r = r0
THEN
@enter_r1 : r ≔ r1
```

```
push_start_motor_button ≙
STATUS
ordinary
WHEN
@grd1 : motor_button= released
THEN
@act1 : motor_button≔ pushed

light_up ≙
STATUS
ordinary
WHEN
@grd1 : motor_button= pushed
@grd2 : motor_light = off
THEN
@act1 : motor_light ≔ on
```

(a) Event-B pattern (b) Renaming rules (c) Instance of the pattern

FIGURE 2: Instantiation process of Event-B design pattern.

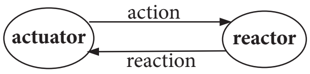


FIGURE 3: Actuator-reactor pattern.

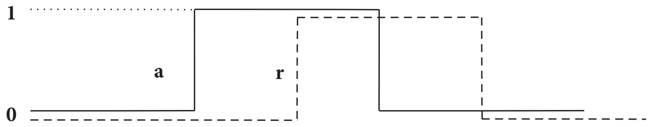
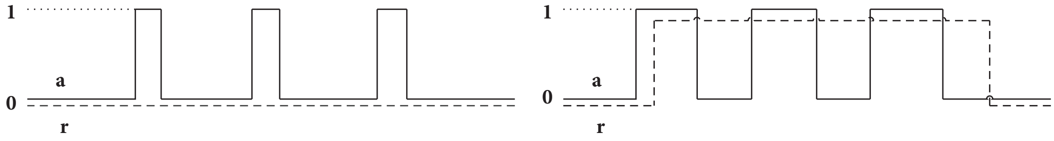


FIGURE 4: Strong synchronization requirement.



(a) (b)

FIGURE 5: Weak synchronization requirement.

executed. Therefore, the event trace of strong synchronization requirement is $\langle a\_on, r\_on, a\_off, r\_off\rangle *$.

(2) Weak Synchronization Requirement. The weak synchronization requirement means that after the actuator executed an action, the reactor may either respond to it or not respond. Thus, the behaviours of the actuator and the responder under the weak synchronization constraint will be the same as in the case of Figure 5. In Figure 5(a), the actuator performs *a_on* and *a_off* events multiple times and the reactor is always in the state $r = 0$. Figure 5(b) shows the case where the reactor is always in the $r = 1$ state.

(3) Strong-Strong Synchronization Requirement. Strong-strong synchronization requirement is the synchronization requirement between two subsystems. Assume that subsystem 1 comprises an actuator *a* and a reactor *r* and subsystem 2 comprises an actuator *b* and a reactor *s*.

The strong-strong synchronization requirement indicates that subsystem 1 and subsystem 2 are strongly synchronous; that is, the *b_on* event must occur after the *r_on* event of subsystem 1; otherwise subsystem 1 will wait forever. At the same time, after the *s_off* event has occurred in subsystem 2, subsystem 1 must execute *a_off* events. This is shown in Figure 6 (Abrial did not give the graphical presentation of strong-strong synchronization patterns in his book [3]. We obtain the graphical representation of the strong-strong synchronization shown in Figure 5 based on the Event-B code of strong-strong synchronous pattern given by Abrial and Figure 3.35 on page 147 in Abrial's book [3]). The event trace of strong-strong synchronization requirement is unique, that is, $\langle a\_on, r\_on, b\_on, s\_on, b\_off, s\_off, a\_off, r\_off\rangle *$.

(4) Strong-Weak Synchronization Requirement. Strong-weak synchronization requirement refers to weak synchronization relationship between subsystem 1 and subsystem 2. Specifically, there are two cases. The first case is that after the *r_on* event occurs in subsystem 1, the subsequent event may be either *b_on* event in subsystem 2 or *a_off* event in subsystem 1. The former indicates that subsystem 2 responds to the event of subsystem 1 and enters into an uninterruptible event trace $\langle b\_on, s\_on, b\_off, s\_off\rangle$; the latter

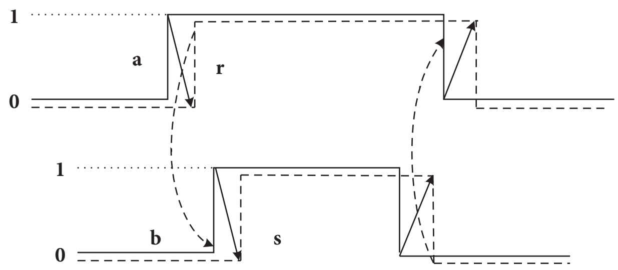
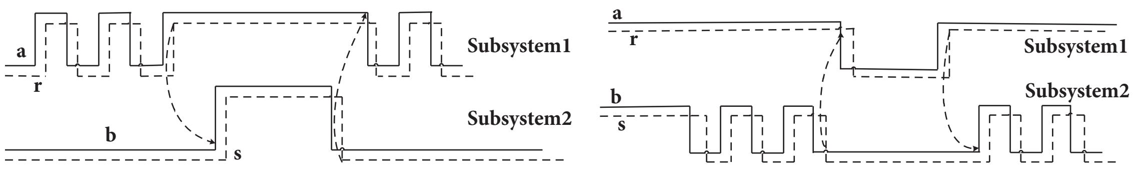


Figure 6: Strong-strong synchronization requirement.



(a)

(b)

Figure 7: Strong-weak synchronization requirement.

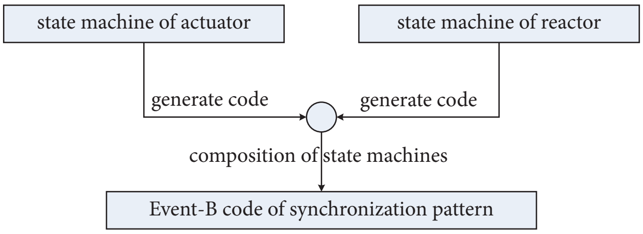
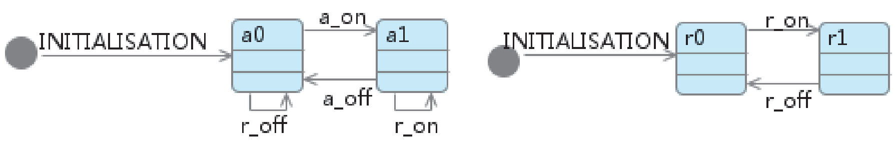

state machine of actuator
state machine of reactor
generate code
generate code
composition of state machines
Event-B code of synchronization pattern

Figure 8: The principle of modelling.



(a) State machine a

(b) State machine r

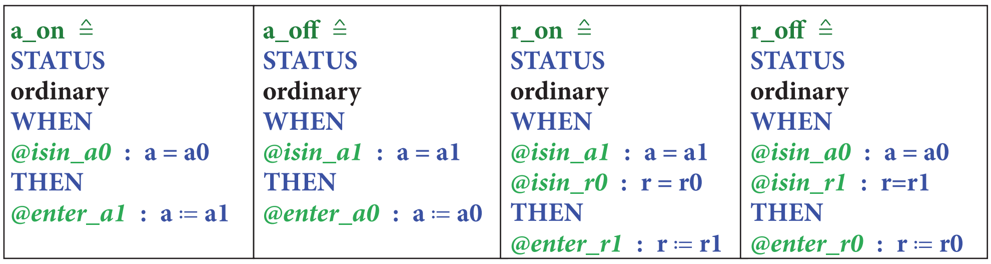
Figure 9: The iUML-B state machines of the weak synchronization pattern.

means that subsystem 2 does not respond to the event of subsystem 1, so subsystem 1 enters its uninterruptible event trace $\langle a\_off, r\_off, a\_on, r\_on\rangle$. The principle of strong-weak synchronization is shown in Figure 7(a) (based on the same reason as Section 4.1 (3), we obtain a graphical representation of the strong-weak synchronization pattern according to Figure 3.19 on page 129 in Abrial's book [3]). The second case is that after the s_off event in subsystem 2 occurs, subsystem 1 may either execute event a_off to respond to it or may not respond, which will cause subsystem 2 to execute the b_on event and return to its uninterruptible trace $\langle b\_on, s\_on, b\_off, s\_off\rangle$. This is shown in Figure 7(b).

4.2. Modelling the System Synchronization Pattern with the iUML-B State Machine. In this section, we use iUML-B state machine to model these synchronization patterns. The principle of our approach is to model the actuators and reactors in each design pattern (e.g., the actuators a and b and the reactors r and s) as an iUML-B state machine, respectively. Event-B code generated by these state machines is embedded in a single Event-B machine. The resulting Event-B machine is the model for various synchronous control flow patterns. The essence of this procedure is the combination of the state machine of the actuator and the state machine of the reactor, as shown in Figure 8.

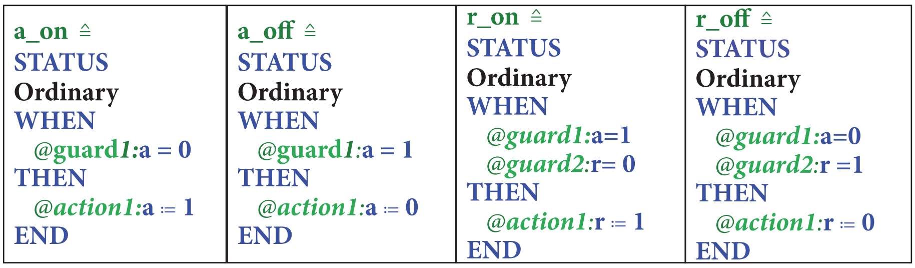
(1) Modelling the Weak Synchronization Pattern. According to weak synchronization requirement, r_on must occur after a_on. Therefore, we first draw the state machines for a and r themselves; then we add a reflexive edge on its state *a1* and "link" this edge to event r_on. Then we can add a reflexive edge on its state *a0* and "link" this edge to event r_off, as shown in Figure 9.

The Event-B codes automatically generated according to iUML-B state machines in Figure 9 are



a_on ≙	a_off ≙	r_on ≙	r_off ≙
STATUS ordinary WHEN @isin_a0 : a = a0 THEN @enter_a1 : a ≔ a1	STATUS ordinary WHEN @isin_a1 : a = a1 THEN @enter_a0 : a ≔ a0	STATUS ordinary WHEN @isin_a1 : a = a1 @isin_r0 : r = r0 THEN @enter_r1 : r ≔ r1	STATUS ordinary WHEN @isin_a0 : a = a0 @isin_r1 : r=r1 THEN @enter_r0 : r ≔ r0

We named it $Weak_{iUMLB.}$ To prove that $Weak_{iUMLB}$ satisfies weak synchronization requirement, we give the Event-B code of weak synchronization proposed by Abrial:



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

We named Abrial's weak synchronization model as $Weak_{EventB}$. It is easy to see that the only difference between $Weak_{iUMLB}$ and $Weak_{EventB}$ is the value of variables a and r. According to the algorithm given in literature [21], we convert both models to labelled transitions system (LTS) and get two identical LTSs, as shown in Figure 10.

If we only concern the behaviour of the model, that is, the event trace, we can conclude that $Weak_{iUMLB}$ and $Weak_{EventB}$ are equivalent.

In order to confirm the above verification results, we simulated the event traces of $Weak_{iUMLB}$ and $Weak_{EventB}$ with the aid of the Rodin platform and found that they are identical. Due to space limitations, in the following discussion, we will no longer give the equivalence proof of the other three synchronization patterns.

(2) Modelling the Strong Synchronization Pattern. According to the requirement of strong synchronization pattern, we modify the state machine of the reactor on the basis of the weak synchronization pattern and add two reflexive edges *a_on* and *a_off* on the $r = r0$ state and $r = r1$ state, respectively, as shown in Figure 11.

(3) Modelling the Strong-Weak Synchronization Pattern. We model the strong-weak synchronization pattern using iUML-B state machines as shown in the four subgraphs (a), (b), (c), and (d) in Figure 12. It should be noted that the strong-weak synchronization pattern is the synchronization between the two subsystems, so we need to add more reflexive edge to limit the event order between the subsystems.

(4) Modelling the Strong-Strong Synchronization Pattern. The strong-strong synchronization pattern adds more constraints on the basis of strong-weak synchronization pattern. We added a new auxiliary state machine m to impose these constraints, as shown in Figure 13. We use dotted arrows to point to the Event-B code generated by each edge in the state machine m. It can be seen that the Event-B codes that generated by state machine m in Figure 13 constrained that event *b_on* must occur between *a_on* and *a_off* event.

As we have analyzed in Section 4.1 (4), in the strong-weak synchronization pattern, after the event sequence $\langle a\_on, r\_on \rangle$ occurs, both of *a_off* event and a *b_on* event are enabled. However, after adding a constraint variable m on the strong-weak synchronization pattern, *a_off* can be enabled only when $a = a1$ and $m = m0$. Therefore, in the Event-B model generated by the strong-strong synchronous pattern state machines, after the event sequence *<a_on, r_on>* occurs, the enabled event can only be *b_on*. Furthermore, as long as *b_on* occurs, the system will go into an uninterruptible event trace $\langle b\_on, s\_on, b\_off, s\_off \rangle$. Thus, the event trace of this Event-B model can only be $\langle a\_on, r\_on, b\_on, s\_on, b\_off, s\_off, a\_off, r\_off \rangle$. That is, the strong-weak synchronization pattern state machine becomes the strong-strong synchronization pattern state machine after adding a new state machine m, as shown in Figure 13.

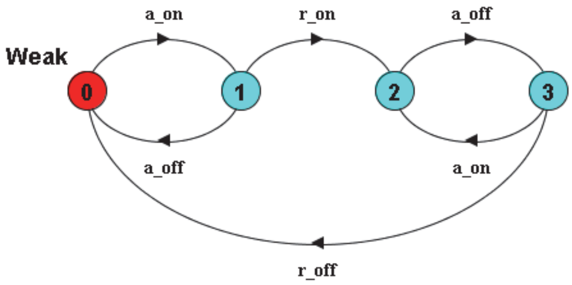
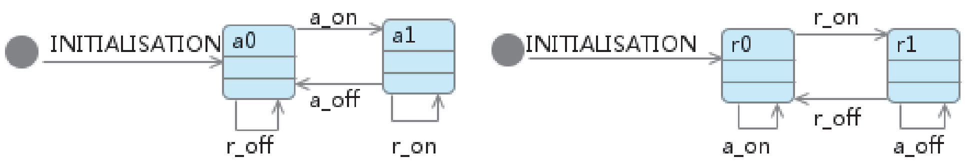


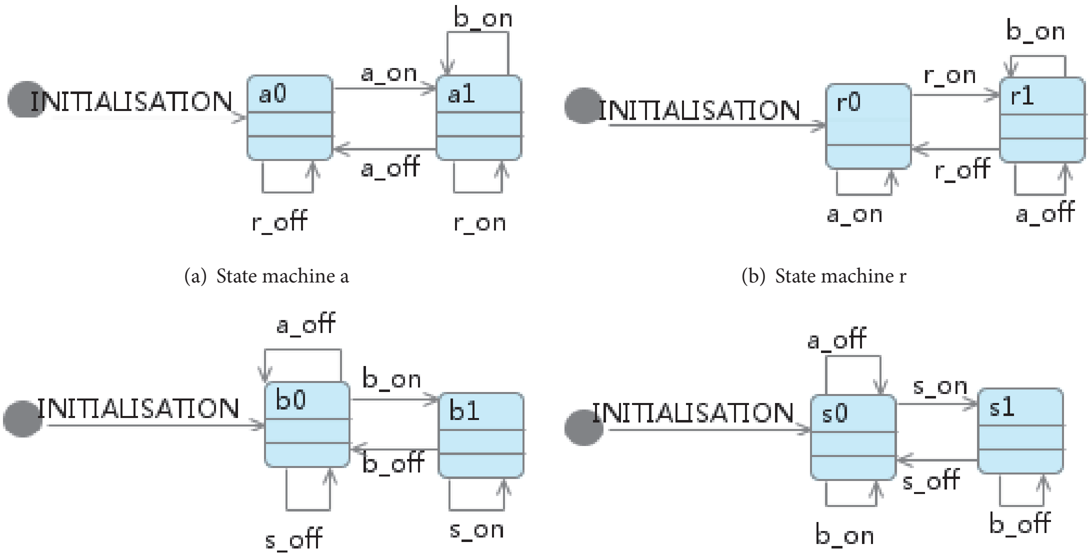
Figure 10: LTS model of weak synchronization pattern.



(a) State machine a

(b) State machine r

Figure 11: The iUML-B state machines of the strong synchronization pattern.



(a) State machine a

(b) State machine r

(c) State machine b

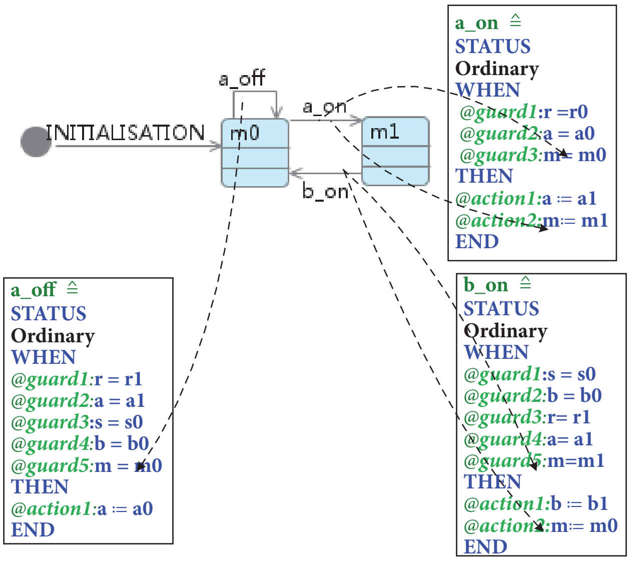
(d) State machine s

Figure 12: The iUML-B state machines of strong-weak synchronization pattern.

5. Modelling the Embedded Control System with Pattern State Machine Instantiation

In this section, we instantiate four synchronous pattern state machines and use these instances to build an Event-B model of an embedded control system. In order to prove the simplicity of our method and to compare it with the Event-B design pattern, we used the case of Abrial's mechanical press controller system in chapter 3 of literature [3]. In the remainder of this paper, we refer to this mechanical press controller system as the "**Press**" system.

5.1. System Overview. The principle of Press system is shown in Figure 14(a). The system consists of control parts and controlled parts. The controlled parts are *Motor*, *Clutch*, and *Door*. The control parts are four control buttons. The control buttons *B1* and *B2* control the start and stop of *motor*, respectively, and *B3* and *B4*, respectively, control the engage and disengage of the *Clutch*. When the *Motor* is working, as long as *Clutch* is engaged, the *Motor* will drive rod; thus slide is driven up and down. Then the tool below the slide will complete the processing of part. The *Door* is to ensure the safety of the staff; that is, when the *Motor* is working and

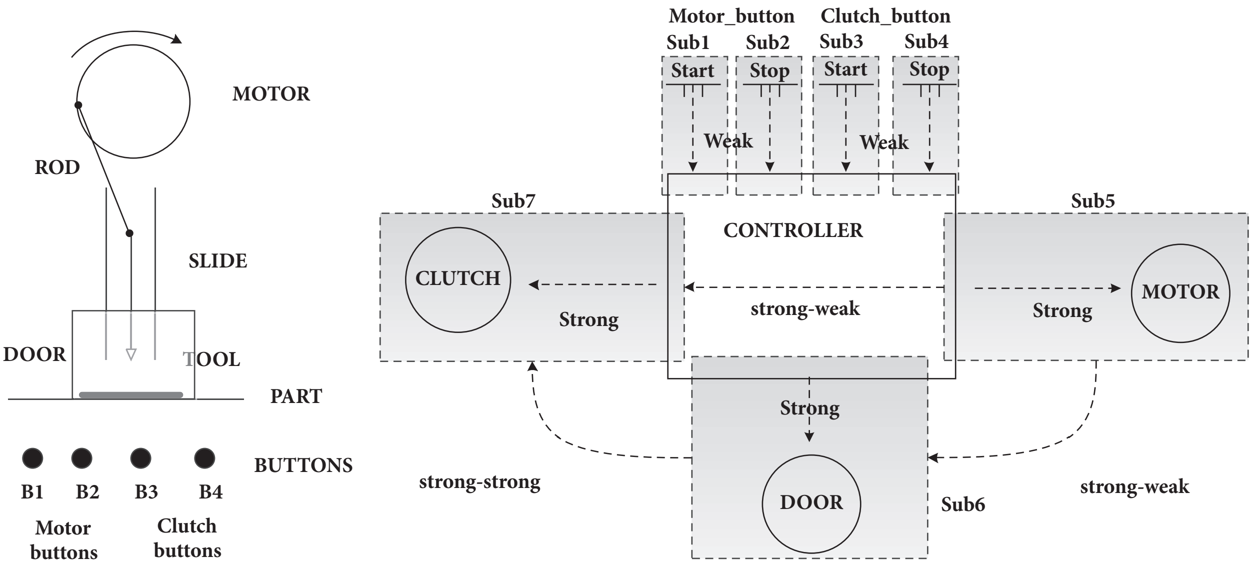


```
a_on ≙
STATUS
Ordinary
WHEN
@guard1:r =r0
@guard2:a = a0
@guard3:m= m0
THEN
@action1:a ≔ a1
@action2:m:= m1
END
```

```
a_off ≙
STATUS
Ordinary
WHEN
@guard1:r = r1
@guard2:a = a1
@guard3:s = s0
@guard4:b = b0
@guard5:m = m0
THEN
@action1:a ≔ a0
END
```

```
b_on ≙
STATUS
Ordinary
WHEN
@guard1:s = s0
@guard2:b = b0
@guard3:r= r1
@guard4:a= a1
@guard5:m=m1
THEN
@action1:b ≔ b1
@action2:m:= m0
END
```

Figure 13: The iUML-B state machine *m* and the Event-B codes it generated.



(a)

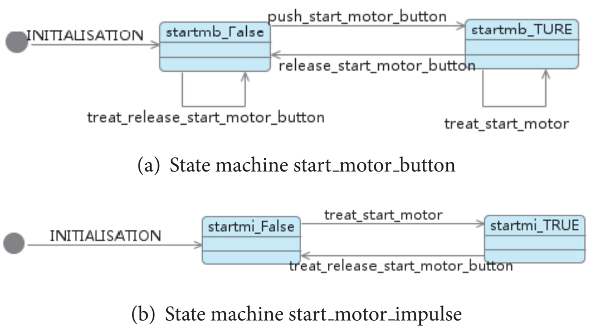
(b)

Figure 14: Mechanical press controller system.

the *Clutch* is engaged, the *Door* must be closed. Likewise, the *Door* can be opened only after the *Clutch* is disengaged. *Door* object is indirectly controlled by *B3* and *B4*.

The synchronization relationships of Press system are shown in Figure 14(b).The relationships between four buttons and the *controller* are weak synchronization. That is, pressing a control button (e.g., *B1*) does not guarantee that the corresponding controlled object will respond to it (e.g., the *Motor* is started). This is because a button may be pressed several times in an instant. But the controlled object can just respond to one of them. The relationship between the *controller* and the *Clutch* (the *Motor*, the *Door*) is a strong synchronization.

In addition, there are strong-weak synchronization or strong-strong synchronization relationships between subsystems, as shown in Figure 14(b).The strong-strong synchronization relationship between subsystems *Sub5* and *Sub7* means that, first, the *Clutch* can be engaged only when the *Motor* is working; at the same time, the *Motor* can be stopped only after the *Clutch* is disengaged; second, the *Motor* can be stopped and started more than once before the *Clutch* is engaged; similarly, the *Clutch* can be engaged and disengaged multiple times before the *Motor* is stopped. The strong-strong synchronization relationship between subsystems *Sub6* and *Sub7* is a safety requirement to ensure the safety of the operator. The strong-weak synchronization between subsystems



(a) State machine start_motor_button

(b) State machine start_motor_impulse

FIGURE 15: Sub1's instance state machine.

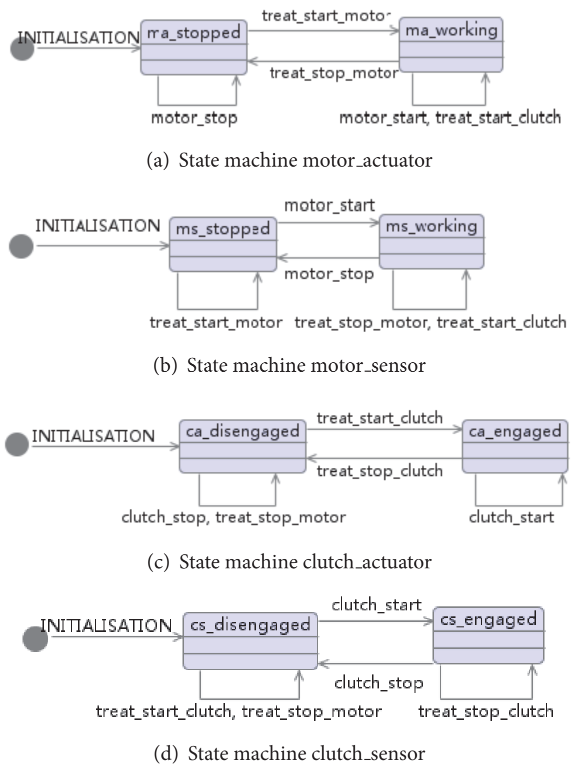
Sub5 and *Sub6* is similar to the relationship between *Sub5* and *Sub7*.

5.2. System Modelling Process. Based on the analysis of the synchronization requirements in Section 4.1, we instantiate the four synchronous pattern state machines of iUML-B into the model of the mechanical press controller system. The instantiation of the Event-B design pattern is a renaming process for the variable names, constant names, and the event names of a pattern. We give only examples of instantiation of subsystem 1 (weak synchronization pattern) and subsystem 5-subsystem 7 (strong-weak synchronization pattern); the other subsystems can be instantiated in the same way as these two examples.

(1) Instantiation of the Weak Synchronization Pattern. According to the renaming relation, we get the two instance state machines *start_motor_button* and *start_motor_impulse* of *Sub1*, as shown in Figures 15(a) and 15(b), respectively.

(2) Instantiation of Strong-Weak Synchronization Patterns. Strong-weak synchronization pattern is the relationship between subsystems, which is more complex than the internal synchronization of subsystems because they contain the synchronization relationships inside each subsystem and the synchronization relationships between subsystems. Without the help of design pattern, this development will be a painful process. Here, we can get the iUML-B state machines of *Sub5* and *Sub7* just by instantiating the strong-weak synchronization pattern state machines into the instance state machines. We can get four instance state machines of *Sub5* and *Sub7*, namely, motor*_actuator, motor_sensor, clutch_actuator,* and *clutch_sensor*, as shown in Figure 16.

(3) Composition of Instance State Machines. To facilitate comparison with traditional design methods, we followed the same refinement approach that has been described by Abrial [3]. We get the iUML-B state machines of *Sub1* to *Sub7* through the instantiation of the pattern state machines and modelled the various synchronization relations between them. The final model of the system consists of 15 iUML-B state machines, of which 14 state machines are shown in Table 1. The 15th state machine is an auxiliary state machine named *m_s6_s7*, which is used to model the strong-strong synchronization relationship between *Sub6* and *Sub7*. It is an instance of the state machine *m* shown in Figure 13.



(a) State machine motor_actuator

(b) State machine motor_sensor

(c) State machine clutch_actuator

(d) State machine clutch_sensor

FIGURE 16: Instance state machines of Sub5 and Sub7.

As we said at the beginning of Section 4.2, the state machine for each subsystem is actually a composed state machine for two state machines within the subsystem. For example, if we use "⊗" to express the composition of state machines, then we have

$$StateMachine_{sub1} = (start\_motor\_button \otimes start\_motor\_imulse) \quad (1)$$

We can get the state machine for the Press system by the composition of all 15 state machines, which is the composition of all 7 subsystems and *m_s6_s7*:

$$StateMachine_{Press} = \left(\left(\bigotimes_{i=1}^{7} Subi\right) \otimes m\_s6\_s7\right) \quad (2)$$

The ultimate manifestation of $StateMachine_{Press}$ is the Press system's Event-B machine.

In fact, "composition of iUML-B state machines" here is not "composition of Event-B machines" but an operation similar to the composition of LTSs. In our previous work [22],we have proven that as long as an iUML-B state machine only describes a single variable's change (we refer to this kind of iUML-B state machine as an "atomic state machine"), we can easily convert it to its corresponding LTS model (we call it "atomic LTS"). We have also mapped the composition of iUML-B state machine to that of LTS. That is, if we

Table 1: Subsystems of Press system and its corresponding iUML-B state machines.

Subsystem	iUML-B state machine		Synchronization pattern
	Actuator	Reactor	
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

have multiple atomic state machines S_1, S_2... S_n, and their corresponding atomic LTS: $LTS\ (S_1)$, $LTS(S_2)\ldots LTS(S_n)$, then we have

$$LTS\left(\bigotimes_{\mathrm{i=1}}^{n} \mathrm{Si}\right) \sim \left(\|_{i=1}^{n} LTS\,(Si)\right) \qquad (3)$$

where "‖" represents the composition of LTS and "~" represents the bisimulation equivalence relation between two LTSs.

Therefore, according to expression (3), for the iUML-B state machines of this paper, we have

$$\begin{aligned} &LTS\,(StateMachine_{sub1}) \\ &\quad \sim (LTS\,(start\_motor\_button) \parallel \\ &\quad LTS\,(start\_motor\_imulse)) \end{aligned} \qquad (4)$$

and

$$\begin{aligned} &\mathrm{LTS}\,(StateMachine_{Press}) \\ &\quad \sim \left(\left\|_{\mathrm{i=1}}^{7} \mathrm{LTS}\,(Sub_i)\right)\right) \parallel \mathrm{LTS}\,(m\_s6\_s7) \end{aligned} \qquad (5)$$

In this way, we can get the LTS model of an Event-B model, which is obtained by integrating of many iUML-B state machines. Further, it allows us to use a variety of model checking tools to verify the behaviour properties of the Event-B model.

6. Evaluation

In this section, we compared the development method based on iUML-B pattern state machine with the development method based on traditional Event-B design pattern. In general, using iUML-B state machines to model the synchronization control flow patterns has three benefits: (1) iUML-B is as easy as UML to learn and can reduce the manual coding costs; (2) iUML-B state machine can express the control flow explicitly; (3) iUML-B state machine can be easily converted to labelled transition system (LTS).

We used the iUML-B pattern state machine to establish the seven-level refinement model of the mechanical press controller system step by step, according to the refinement step that has been described by Abrial [3]. The simulation results on the Rodin platform show that, at each layer, the event trace of the model obtained using our method is the same as that of the corresponding model obtained using Abrial's traditional design pattern. The growth trend of these elements is shown in Figure 17.

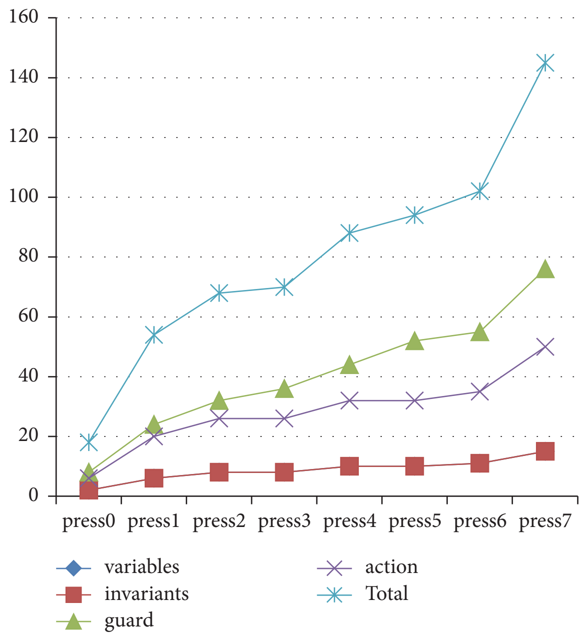


Figure 17: Statistics of Press model generated by iUML-B state machine.

We use Abrial's nongraphical model of the Press system (http://deploy-eprints.ecs.soton.ac.uk/113/2/ch3_press.zip), which is a direct hand-coded Event-B model, for comparison with our work. For comparison, we also present the statistical data for the mechanical press controller model developed by the traditional Event-B design pattern. The growth trend of its elements is shown in Figure 18 (the growth trend of variables and invariants in the figure is the same, so the two curves coincide).

From the statistical data, we can observe a phenomenon: no matter what method is used to model a control-intensive system, the number of model codes grows fast with the growth of refinement level. For the traditional Event-B design pattern method, the total number of variables, invariants, guards, and actions is 142 when refinement level is 7. For a method based on the iUML-B pattern state machine, this

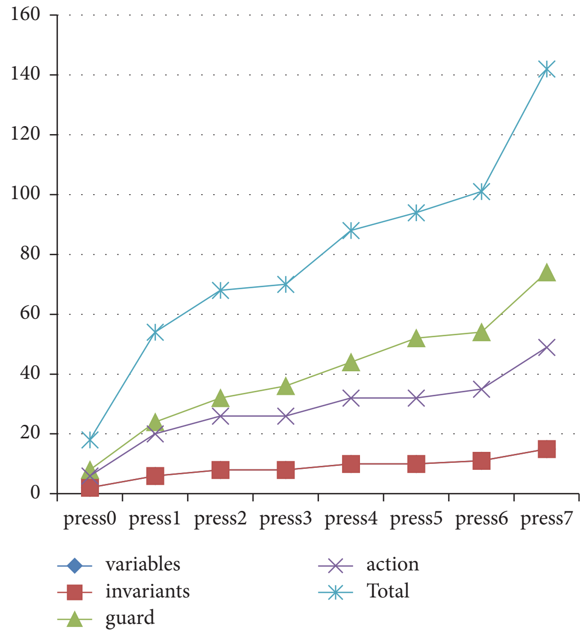


Figure 18: Statistics of Abrial's Press model.

number is 145. From the graphs in Figures 17 and 18, it can be seen that the growth rate of guards is the fastest, followed by actions. For more complex multilevel control system, the slope of these two curves will be greater.

Although the statistics in Figures 17 and 18 are similar, the guards and actions in Figure 17 are automatically generated from the iUML-B state machine and the guards and actions in Figure 18 must be manually coded by the modeller. In other words, using the iUML-B pattern state machine to model control-intensive systems, the amount of manual coding we can save at least is equal to (74 + 49)/142 = 86.6%.

Another benefit of our approach is that the control flow of the Event-B model becomes visible. Using the iUML-B state machine, we can express and analyze the event order of the system model easily. And the larger Event-B model is decomposed into smaller subsystems, each with its separated control flow. In this way, the complex multilevel control problem becomes a simple subsystem modelling problem. The final Event-B model is a composition of these subsystems. From the engineering point of view, at the time of decomposing the system, we also decompose the complexity of modelling problems and distribute this complexity to some smaller state machines.

Finally, the iUML-B state machine can be easily converted to LTS, so that the behaviour properties of the Event-B model can be analyzed and verified. From the point of view of behavioural semantic verification, this is a very desirable advantage.

7. Conclusion and Future Work

In this paper, we use the iUML-B pattern state machine to model the four synchronization design patterns in the embedded control system. Then we instantiate the four iUML-B pattern state machines to get the Event-B model of a complex, multilayer control system. The simulation results show that the event trace of the system model obtained using the iUML-B pattern state machine is the same as that of the model obtained using the traditional Event-B design pattern.

The advantage of our approach lies in the following points. First, the process of modelling synchronous control flow patterns is visible. The system's control flow synchronization patterns are expressed using the iUML-B state machines. This makes the system model be understood more easily. Second, the modelling process of the system becomes simpler. Using the traditional approach based on the Event-B design pattern, the modeller who wants to add a new control flow needs to find the locations in hundreds or even thousands of lines of code to insert the guards and actions. However, using the iUML-B pattern state machine, people only need to add some transition edges in a state machine to complete these tasks. Finally, the decomposition and composition of the subsystems are visible. The system's control flow model is decomposed into some iUML-B state machines, and each pair of iUML-B state machines forms a subsystem. The final system model is the composition of these iUML-B state machines.

Event-B is a data-oriented formal modelling language. Therefore, it cannot guarantee the consistency of behaviour between the refined model and the abstract model. In the future, we want to propose an integrated formal method that can guarantee the consistency of the refined model and abstract model in both behaviour aspect and data aspect.

Disclosure

Han Peng is a teacher at School of Computer Science, Xi'an Aeronautical University. Now he is studying for a doctoral degree at the School of Computer Science, Northwestern Polytechnical University.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper

Acknowledgments

The authors are very grateful to Professor Colin Snook and Dr. Thai Son Hoang of the University of Southampton for their valuable constructive suggestions in this research work.

References

[1] G. Gigante and D. Pascarella, *Formal Methods in Avionic Software Certification: The DO-178C Perspective*, Springer, Berlin, Germany, 2012.

[2] F. De Rosa, R. Cesoni, S. Genta, and P. Maggiore, "Failure rate evaluation method for HW architecture derived from functional safety standards (ISO 19014, ISO 25119, IEC 61508)," *Reliability Engineering & System Safety*, vol. 165, pp. 124–133, 2017.

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

[4] D. Méry and N. K. Singh, "Modeling an Aircraft Landing System in Event-B," *Communications in Computer and Information Science*, vol. 433, pp. 154–159, 2014.

[5] A. Iliasov, E. Troubitsyna, L. Laibinis et al., "Developing mode-rich satellite software by refinement in Event-B," *Science of Computer Programming*, vol. 78, no. 7, pp. 884–905, 2013.

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

[7] R. Silva, "Application of Decomposition and Generic Instantiation," *Environmental Modelling Software*, vol. 47, pp. 138–147, 2011.

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

[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 2nd NASA Formal Methods Symposium*, pp. 182–191, 2010.

[10] S. Yeganefard and M. Butler, "Problem decomposition and sub-model reconciliation of control systems in Event-B," in *Proceedings of the IEEE 14th International Conference on Information Reuse and Integration*, pp. 528–535, August 2013.

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

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

[13] A. Gondal, M. Poppleton, and M. Butler, "Composing Event-B specifications—case-study experience," in *Software Composition*, vol. 6708 of *Lecture Notes in Computer Science*, pp. 100–115, Springer, Berlin, Germany, 2011.

[14] A. Intana, *Formal engineering methodologies for wireless sensor network development with simulation*, University of Southampton, 2015.

[15] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: An open toolset for modelling and reasoning in Event-B," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, 2010.

[16] C. Snook and M. Butler, "UML-B: Formal modeling and design aided by UML," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 92–122, 2006.

[17] A. S. Fathabadi, C. Snook, and M. Butler, *Applying an integrated modelling process to run-time management of many-core systems*, 2014, International Conference on Integrated Formal Methods, Springer, Cham.

[18] T. S. Hoang, C. Snook, L. Ladenberger, and M. Butler, "Validating the requirements and design of a hemodialysis machine using iUML-B, BMotion studio, and co-simulation," in *Proceedings of the International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 360–375, 2016.

[19] C. Snook, T. S. Hoang, and M. Butler, "Analysing security protocols using refinement in iUML-B," in *Proceedings of the NASA Formal Methods Symposium*, pp. 84–98, 2017.

[20] M. Butler, *Decomposition Structures for Event-B*, Springer, Berlin, Germany, 2009.

[21] D. L. Chaudhari and O. P. Damani, "Generating hierarchical state based representation from event-B models," *Electronic Notes in Theoretical Computer Science*, vol. 280, no. 1, pp. 35–46, 2011.

[22] H. Peng, C. Du, L. Rao, and F. Chen, "A LTS approach to control in event-B," *Scientific Programming*, vol. 2018, Article ID 8765186, 11 pages, 2018.