ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, *25-27 June*, *2017*, *pp. 1179-1186*
doi: 10.18178/wcse.2017.06.206

Modeling Control Flow of Event-B Using State Transition System

Han Peng [1+], Chenglie Du [1] and Haobin Wang [2]

[1] College of Computer Science, Northwestern Polytechnical University, Xi'an, China
[2] College of Computer Science, Xi'an Aeronautical University, Xi'an China

Abstract. There are some limitations of Event-B method in expressing the event order of system. In order to solve this problem, event refinement structure method was proposed to model the system refinement structure and control flow. However, the event refinement structure diagram cannot directly map to a behavioral semantic model such as communication sequence process or labeled transition system, and is inconvenient to verify the behavior properties of system. In this paper, we propose a general method to model the control flow of the Event-B model with the iUML-B state machine; so that it has the same event traces as the event refinement structure method. Then, we use a simple case to prove the practicality of this method. Finally, we map the iUML-B state machine to a labeled transition system and verify the behavior properties of the system.

Keywords: Event-B, control flow modeling, labeled transition system, iUML-B state machine

1. Introduction

Event-B[1] is a formal method evolved from B method[2] and action system[3]. It uses simple symbols and structures to model the system, and is well suited for different fields including distributed systems. In order to solve the limitation of event B in control flow modelling, event refinement structure (ERS) method [4-8] is proposed. The ERS method uses a tree structure similar to the Jackson Structure Diagram (JSD) style to represent the association between the abstract event and the refinement event and the order in which the event occurred. Then, with the support of the Rodin platform, one can generate Event-B code from the ERS model.

However, there are some limitations of ERS method. For example, for those engineers who are accustomed to using the state transition system modeling, there is a gap between ERS's JSD diagram style and state transition diagrams. In addition, we cannot get the events order of the system directly from the ERS event refinement structure diagram. Compared with the JSD graph, the state diagram is not prone to ambiguity and is easier to map to the behavior semantic model such as the labeled transition system (LTS).

In this paper, we propose a method to model Event-B control flow with the iUML-B state machine[9]. First, we use the iUML-B state machine to express the event order of the ERS decomposition pattern and get the event decomposition pattern represented by the iUML-B state machine. Secondly, we use an example of an elevator control system to demonstrate the practicality of our method. Finally, we map the iUML-B state machine model to the LTS model, and use the LTS analysis tool to verify the behavior properties of the elevator control system. Compared with the ERS method, our modeling method visualizes the behavior model of the system and uses LTS as the behavior semantic model. Therefore, we can use LTS analysis tool (LTSA)[10] to verify the behavior properties of the system.

2. Event refinement using iUML-B state machine

[+] Corresponding author. Tel.: + 8613474113698; fax: +86(029)84252366.
E-mail address: hansbeng2016@gmail.com.

2.1. Event refinement structure

The atomic decomposition pattern of the ERS method consists of four control flow patterns and four replicator patterns. Our main concern is the four control flow patterns, including *Sequence* pattern, *Loop* pattern, *And* pattern and *Xor* pattern. The idea of the ERS method is shown in Figure 1.

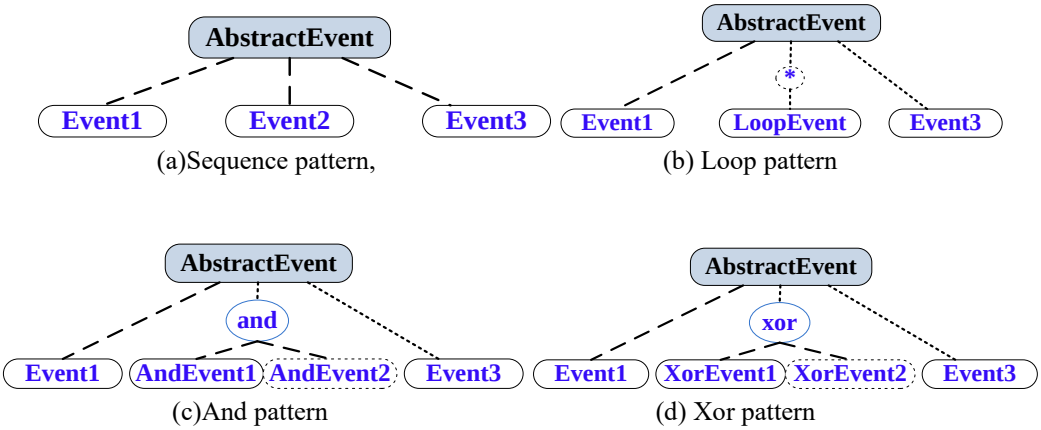


Fig. 1: The idea of the ERS method

- *Sequence* pattern

The *Sequence* pattern is shown in Figure 1 (a), which is intended to break down an abstract events into several refinement events that occur in orders, where there must be a refinement event to refine the abstract event. The event trace of *Sequence* pattern is:

$$<Event1, Event2, Event3>.$$

- *Loop* pattern

The loop pattern is shown in Figure 1 (b), which is intended to break down an abstract event into an event which can occur repeatedly (loop event) as well as two ancillary events. The event trace of *Loop* pattern is (We use '*' to indicate that events can be repeated):

$$<Event1,(LoopEvent)^{*},Event3>.$$

- *And* pattern

The *And* pattern is shown in Figure 1 (c), with the intention of breaking down an abstract event into two or more refinement events that can be executed in any order. Moreover, *Event3* can be enabled only after both of two "*And*" events have been completed. The event traces of *And* pattern is:

$$<Event1，AndEvent1，AndEvent2，Event3>$$

or

$$<Event1,AndEvent2,AndEvent1，Event3>.$$

- *Xor* pattern

The *Xor* pattern is shown in Figure 1 (d), with the intention of decomposing an abstract event into two or more refinement events, with only one of them being executed .The event traces of *Xor* pattern is:

$$<Event1,XorEvent1,Event3>$$

or

$$<Event1,XorEvent2，Event3>$$

2.2. Using iUML-B state machine to express event refinement structure

- General method

Our method is inspired by Stefan Hallerstede's paper[11]and[12]. Hallerstede proposed an idea of the control flow modelling of the Event-B model, and pointed out that during the refinement of the Event-B model, edge refinement (or event refinement) and node refinement (or state refinement) are similar. Hallerstede gives the equivalence between edge refinement diagram and node refinement diagram. Based on Hallerstede's works, we derived a general approach to express ERS using the iUML-B state machine, which is described as follows:

1) An abstract state machine is used to describe the trace of execution of the abstract model.

2) In the refinement model, we change a node in the abstract state machine into a super node in the concrete state machine.

3) We add new states and events in the super node, as refinement events in the next level.

4) The refinement event is assigned to the edge of the state machine, and ensures that its execution trace meets the requirements for decomposition / refinement.

According to the above general method, we first give the initial abstract state machine model. Then we give the iUML-B state machine representation of the four decomposition patterns described above.

The initial abstract state machine model is shown in the Figure 2.

Fig. 2: The initial abstract state machine

- Event refinement using iUML-B state machine

1) *Sequence* pattern

In order to get the correct trace of events, in the *Sequence* decomposition pattern, we change the *S0* state of the abstract state machine to super node and add three sub-states *S0_1*, *S0_2*, *S0_3* and two events *Event1*, *Event2* in its nested state machine, then let *Event3* refine the *AbstractEvent*, as the Figure 3 shows.

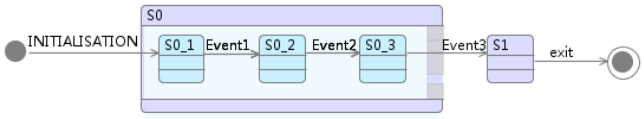
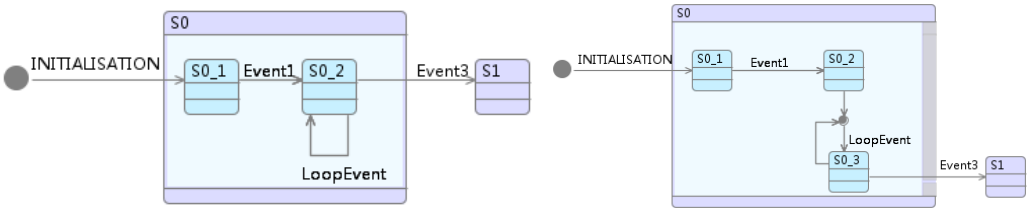


Fig. 3: Sequence decomposition pattern

2) *Loop* pattern

In the *Loop* decomposition pattern, we change the S0 state of the abstract state machine to the super node, and add two sub-states of *S0_1*, *S0_2* ,*Event1* and *LoopEvent* in its sub state machine, where *LoopEvent* is the reflex edge of state *S0_2*, and then let *Event3* refine the *AbstractEvent*, as the Figure 4 (a)shows. We can also model *Loop* pattern using the pseudo-state node, as shown in the Figure 4(b).



(a) (b)

Fig. 4: Loop decomposition pattern

3) *And* pattern

In order to get the parallel event, in the And decomposition pattern, we first change the *S0* state of the abstract state machine to super node and add two sub-states of *S01, S02* and *Event1* event to its sub state machine. Then states *S02* is spitted into two orthogonal state machines and add the corresponding sub-states and events, *AndEvent1* and *AndEvent2*, and finally let *Event3* refine the abstract event *AbstractEvent*, as the Figure 5 shows.

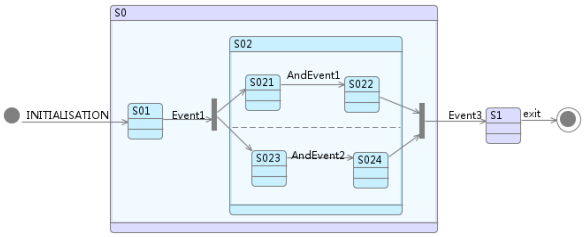


Fig.5: And decomposition pattern

4) *Xor* pattern

In the *Xor* decomposition pattern, we change the *S0* state of the abstract state machine to super node and add three sub states *S0_1*, *S0_2*, *and S0_3* as well as *Event1*, *XorEvent1* and *XorEvent2* events to the sub state machine. Let *XorEvent1* event and *XorEvent2* event have the same source state and the distinct target state, and finally let *Event3* refine the abstract event *AbstractEvent*, as Figure 6 shows.

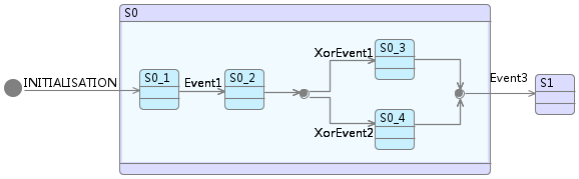


Fig. 6: Xor decomposition pattern

We use the iUML-B state machine to implement the ERS method and the atomicity decomposition method of the four control class event decomposition patterns. The event traces of the model are same as the corresponding patterns in the ERS method.

3. Case study

3.1. Modeling the elevator system using iUML-B State Machine

We use a simple example of an elevator system to demonstrate the practicality of an event decomposition model based on the iUML-B state machine. In order to make our work clearer, we briefly describe the requirements of the elevator control system as follows:

The elevator control system consists of three key objects: *elevator*, *door* and *button*. The *elevator* can be moving or stopped. The *door* can be closed or opened. After the passenger enters the elevator and presses the *button*, the elevator will stop at the requested floor.

The complete requirements for the elevator system can be found in the literature[7].In this paper, we only care the events added during the system refinement and the constraints imposed by the atomicity decomposition on the order between events.

- Control flow requirements of elevator system

The control flow requirements for the elevator system are shown in Table 1, which is come from the[7].

Table 1: Description of flow requirements

Flow requirements	**Example Description**
Sequencing requirements	LIFT7-The floor door closes before the lift is allowed to move
Selection requirements	LIFT8-If a lift is stopped then the floor door for that lift may be open. In this requirement the lift door can be either opened or left closed when the lift is stopped.
Repetition requirements	LIFT9-There might be more than one external floor request in a particular floor, the lift will respond to them (stop) only once

- Control flow refinement process based on iUML-B state machine

We used the iUML-B state machine to refine the control flow. The original model of the system is shown in the Figure 7.

1) Top-level abstraction model

Since the top-level abstraction model does not involve the interaction between the elevator and the door, there is no need to describe the relations between *LiftMove* event and *LiftStopevent*, because this requirement has been modelled in the *liftStatemachine0*, as shown in Figure 7.

2) First refinement

As with the [7], we introduced three events, namely *OpenLiftDoor*, *CloseLiftDoor* and *NotOpenLiftDoor* in the first refinement to express the behaviours of door, as shown in Figure 8.

The difference is that we use another state machine, *FlowStateMachine1*, to constrain the event traces of system, as shown in Figure 9. We use the pseudo-state to express the *Xor* relationship between the *OpenLiftDoor* event and *NotOpenLiftDoor* , and put them in a nested super state. Then we make the *LiftStop* event to be a super-state's ingoing event, which limits the *LiftStop* event must occurs before *OpenLiftDoor* event or *NotOpenLiftDoor*. Similarly, we let the *LiftMove* event to be the outgoing edge of the super-state, which specifies that the *LiftMove* event must be executed after these two events.

Fig. 7: LiftStateMachine

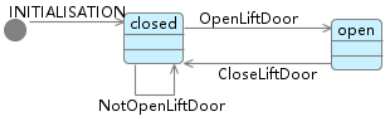


Fig.8: DoorStateMachine

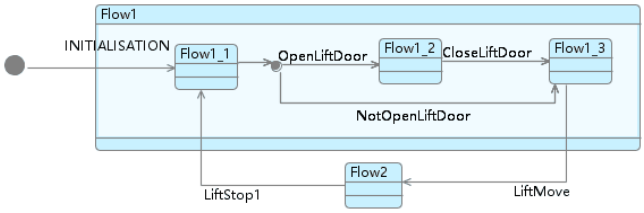


Fig. 9: FlowStateMachine1

3) Second refinement

This refinement introduces the *RequestFloor* event, which indicates that the passenger pressed the button and chose a floor. As suggested in the [7], passengers should at least choose one floor and the elevator will stop at the corresponding floor. So the *RequestFloor* event should occur at least once before the *LiftStop* event occurs. The inventors of the ERS method also admitted that they could not express such "at least once" requirement. But the iUML-B state machine can express it, as shown in the Figure 10.

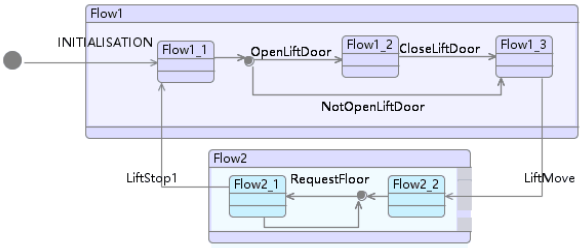


Fig. 10: FlowStateMachine2

We use the iUML-B state machine to construct a simple elevator control system for the control flow refinement process. At each refinement level, we distinguish between object state machines and control flow state machines. The object state machine describes the action of an object (*elevator*, *door*) itself, while the control flow state machine constraint the overall event order of the elevator control system. By combining these two types of state machines, we get a framework of correct event traces of elevator control system. The event order of this framework is consistent with the order in which the ERS method generated.

3.2. Formalization of system behavior

In order to facilitate the observation of the event order of the system and verify its behaviour properties, we convert the iUML-B state machine to LTS and verify its behaviour using the LTSA analysis tool.

- Convert iUML-B state machine to LTS

We use the finite state process (FSP)[10] to express LTS and visualize it with LTSA.

1) Top-level abstract LTS model

The top-level abstraction model of the elevator control system described by LTS is:

Lift = (liftstop-> liftmove-> Lift).

The graphical representation of the above LTS model is shown in Figure 11.

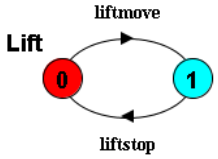


Fig. 11: Abstract LTS model of elevator system

2) First refinement LTS model

To express the state transition of the door, we write LTS as follows:

Door = (openliftdoor-> closeliftdoor-> Door | notopendoor-> Door).

The graphical representation of the above LTS model is shown in Figure 12.

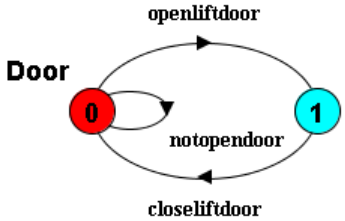


Fig. 12: Abstract LTS model of door system

The LTS control flow model corresponding to the first refinement is:

Flow1 = (liftmove-> liftstop-> Flow2),

Flow2 = (openliftdoor-> closeliftdoor-> Flow1 | notopendoor-> Flow1)

The graphical representation of the above LTS model is shown in Figure 13.

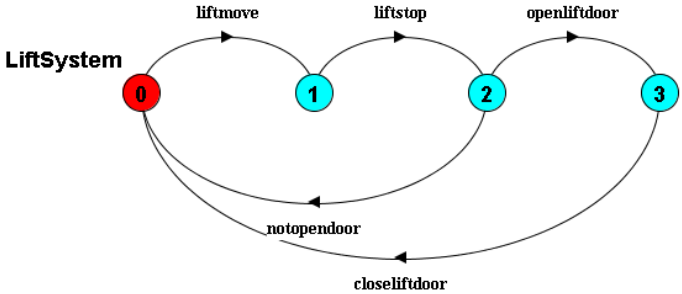


Fig. 13: First refinement of entire system

2) Second refinement LTS model

The LTS control flow model corresponding to the second refinement is:

Flow1=(liftmove->Flow2),

Flow2=(requestFloor->Flow2|requestFloor->Flow3),

Flow3=(liftstop->Flow4),

Flow4=(openliftdoor->closeliftdoor->Flow1|notopendoor->Flow1).

The graphical representation of the above LTS model is shown in Figure 14.

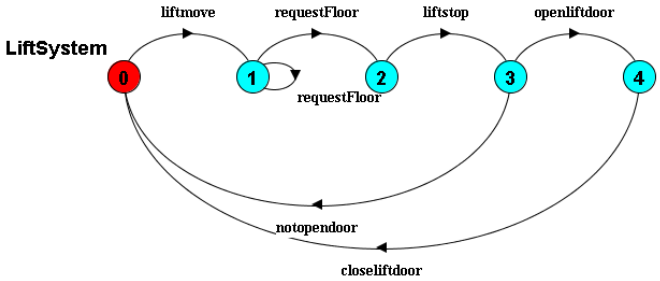


Fig. 14: Second refinement of entire system

- Simulation and verification of system behavior

We use the LTSA tool to simulate the final behavioural model of the system. The simulation result is shown in Figure 15.

It can be seen that after the *Liftmove* event occurs, the *RequestFloor* event can occur one or more times before the *Liftstop* event can occur. This is in line with the system requirements, because only after the passengers press the floor button, the elevator will stop in a certain floor. The event traces of model also show that it satisfies the three control flow requirements described in Table 1.

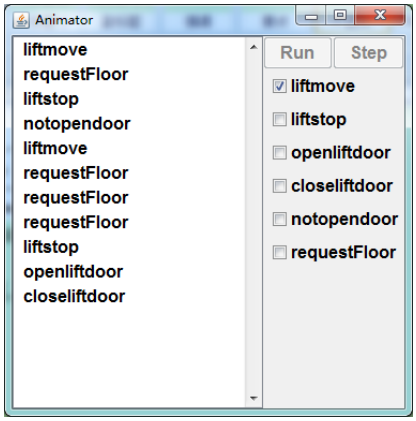


Fig. 15: Simulation of system behaviour

We used the LTSA tool to verify the safety property of the final elevator control system's behavior model. The results show that the model is deadlock-free.

3.3. Comparison with atomicity decomposition(AD) method

By comparing we found that the AD method is good at description of the relationship between different refinement levels, but it cannot express event orders intuitively. iUML-B state machine is more intuitive in expressing the event order at the same level, but cannot express the refinement relationship between different levels. Moreover, the AD plug-in has some constraints in the Event-B code generation. Sometime we need add control variable manually, while iUML-B state machine can generate Event-B control code without manual operations.

4. Conclusion

In this paper, we have proposed a method to construct an Event-B control flow model using iUML-B state machine, which explicitly expresses the control flow of the Event-B model by state transitions. The control flow model constructed with the iUML-B state machine can automatically generate Event-B's state

control variables to control the event traces of the model. Furthermore, we have mapped iUML-B state machine to a LTS behavioural semantic model, this allows us to verify the behaviour properties of the system as early as possible.

In the future, we will use a variety of temporal logic properties to constrain the behaviour of the system model, and get the corresponding Event-B control flow model.

5. Acknowledgements

We would like to express our very great appreciation to Professor Colin Snook for his valuable and constructive suggestions during this research work.

6. References

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

[2] J. R. Abrial, "The B-book: assigning programs to meanings," 1996.

[3] R. J. R. Back and F. Kurki-Suonio, "Distributed cooperation with action systems," *Acm Transactions on Programming Languages & Systems,* vol. 10, pp. 513-554, 1988.

[4] A. S. Fathabadi and M. Butler, *Applying Event-B Atomicity Decomposition to a Multi Media Protocol*: Springer Berlin Heidelberg, 2009.

[5] A. S. Fathabadi, A. Rezazadeh, and M. Butler, "Applying Atomicity and Model Decomposition to a Space Craft System in Event-B," 2011, pp. 328-342.

[6] A. S. Fathabadi, "An approach to atomicity decomposition in the Event-B formal method," *University of Southampton,* 2012.

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

[8] A. S. Fathabadi, M. Butler, and A. Rezazadeh, "Language and tool support for event refinement structures in Event-B," *Formal Aspects of Computing,* vol. 27, pp. 499-523 2015.

[9] C. Snook and M. Butler, "UML-B: Formal modeling and design aided by UML," *Acm Transactions on Software Engineering & Methodology,* vol. 15, pp. 92-122, 2006.

[10] J. Magee and J. Kramer, *Concurrency: state models & Java programs*: John Wiley & Sons, Inc., 2000.

[11] S. Hallerstede, "Structured event-b models and proofs," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2010, pp. 273-286.

[12] S. Hallerstede and C. Snook, *Refining Nodes and Edges of State Machines*: Springer Berlin Heidelberg, 2011.