Hindawi
Scientific Programming
Volume 2018, Article ID 8765186, 11 pages
https://doi.org/10.1155/2018/8765186

Research Article

A LTS Approach to Control in Event-B

Han Peng,[1] **Chenglie Du**,[1] **Lei Rao**,[1] **and Fu Chen**[2]

[1]*Northwestern Polytechnical University, Xi'an, China*
[2]*Xi'an Aeronautics Computing Technique Research Institute, Xi'an, China*

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

Received 1 November 2017; Revised 19 March 2018; Accepted 3 April 2018; Published 22 May 2018

Academic Editor: Emiliano Tramontana

In Event-B, people need to use control variables to constrain the order of events, which is a time-consuming and error-prone process. This paper presents a method of combining labeled transition system and iUML-B to complete the behavior modeling of system, which is more convenient and practical for engineers who are accustomed to using the automaton to build a system behavior model. First, we use labeled transition system to establish the behavior model of the system. Then we simulate and verify the event traces of the labeled transition system behavior model. Finally, we convert labeled transition system model into iUML-B state machine and use it to generate the corresponding control flow model. We use Abrial's bounded retransmission protocol to demonstrate the practicality of our approach. The simulation results show that the system behavior model generated by the iUML-B state machine has the same event trace as the corresponding labeled transition system model.

1. Introduction

Event-B [1] is a formal method that evolved from B method [2] and action system [3]. It uses simple symbols and structures to model the system, and it is well suited for different areas including distributed systems [4]. The Event-B method itself is very good at event refinement and data-oriented refinement, but it lacks formal semantics of behavior, and the control flow of Event-B machine is hidden in the state variables, which can enable the event and can be modified by actions. As a result, Event-B machine is not as intuitive as the automaton-based approach.

From the designer's point of view, to constrain the order of events using control variables is a time-consuming and error-prone process. People have to carefully consider the impact of each variable on the event. Control variables and other variables in Event-B machine entangled together, which makes the design process more complex. From the analyst's point of view, it needs a lot of effort to understand the control flow of system [5]. It is very inconvenient for people who want to understand the overall behavior and verify properties of the system and find the right solution.

To summarize, the problem in Event-B control flow is as follows: first, the control flow part and computation part of Event-B machine are entangled, which makes design and analysis more difficult; second, the control flow part is not visible, which makes it impossible to model and verify the control flow separately.

In order to solve these two problems, this paper presents a method of modeling, analyzing the control flow of Event-B machines using labeled transition system (LTS). The basic idea is to divide the model into control part and the computation part and further divide the control part into component control flow and interaction control flow. The procedure of our approach is as follows: first, we use LTS to model the component behavior and interaction behavior of the system so that we can visualize the control flow of each component and the entire system (we omit the computation part of the Event-B machine and do not consider it in the control flow model); second, we simulate system behavior using this formal model and get the event trace we need; third, we convert the component behavior LTS and the interaction behavior LTS into the iUML-B component state machine and the control flow state machine, respectively. Finally, we use these iUML-B state machines to generate Event-B machines. We proved in Section 2.3 that the Event-B machine generated using the above steps will only preserve those event traces that we need.

The remainder of this paper is organized as follows. Section 2 summarizes the issues related to Event-B control flow modeling, as well as the general approach we propose.

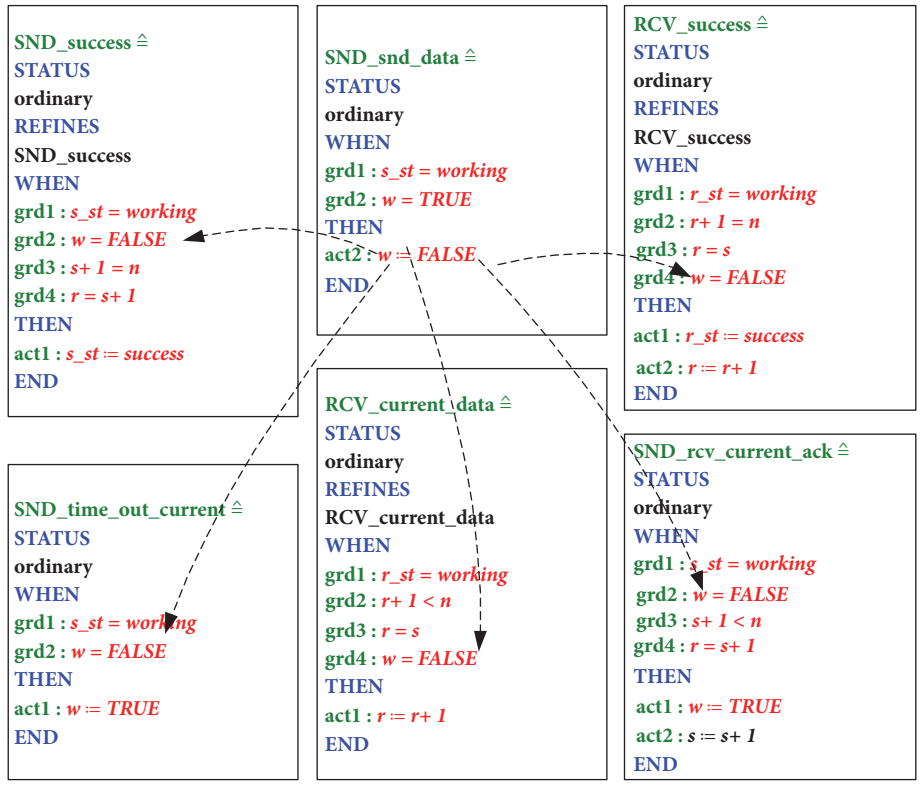


Figure 1: The control variables of Event-B machine.

Section 3 uses Abrial's bounded retransmission protocol to demonstrate the practicality of our approach. Section 4 discusses our approach. Section 5 presents related work and compares the results of this work with existing work. Finally, Section 6 concludes the paper and gives some future research directions.

2. Methodological Approach

2.1. Problem Description. We use Abrial's bounded retransmission protocol (BRP) model [1] to explain the control flow problems in Event-B machine. In the fourth refinement model of BRP machine (the example can be found at http://www.math.pku.edu.cn/teachers/jrabrial/course/BRP.zip), there are multiple control variables that affect the event order, as Figure 1 shows.

To simplify the problem, we omit the variables that do not affect the event order of Event-B machine. In Figure 1, we mark the position of these variables in italics. Now, if we want to analyze the effect of Boolean variable W on the event order, we will find that the event *SND_snd_data* changes the value of W from TRUE to FALSE, which will affect all the other events in the graph (we only list events affected by W), as indicated by the dashed arrows. But this does not mean that when W is FALSE, the other events will be enabled. In fact, the occurrence of other events depends on the value of more control variables, such as *s_st* and *r_st* etc. Therefore, in order to analyze the effect of W on the event order, we have to put these events together in a graph so that we can understand the intention of the modeler.

2.2. Control the Event-B Machine Using iUML-B State Machine. iUML-B [6, 7] is a graphical plug-in of Event-B, which use "UML like" class diagrams and state machines to describe the state and behavior of system. The iUML-B's graphical model can generate Event-B code directly on the Rodin platform [8] and automatically embed the generated code into the context part (set, constant, and axiom) and machine parts (variables, events, and invariants).

The transform edges in the iUML-B state machine can be "linked" to the events in the Event-B machine to control the event order in Event-B machine. The principle is shown in Figure 2.

For example, if we want to restrict the event order in the Event-B machine as

<INITIALISATION, SND_snd_data, RCV_current_data, SND_rcv_current_ack, brp>,

we can first draw the iUML-B state machine shown in Figure 2; then "link" each transition edge to an event to make the events in the Event-B machine occur in the order specified in the iUML-B state machine.

Although the iUML-B state machine can model the control flow of the Event-B machine, it lacks formal behavioral semantics. Therefore, we need to use LTS to model and analyze its behavior.

2.3. Behavior Modeling Strategy and Equivalence Proving. We present the following Event-B machine behavior modeling strategy based on the above analysis results:

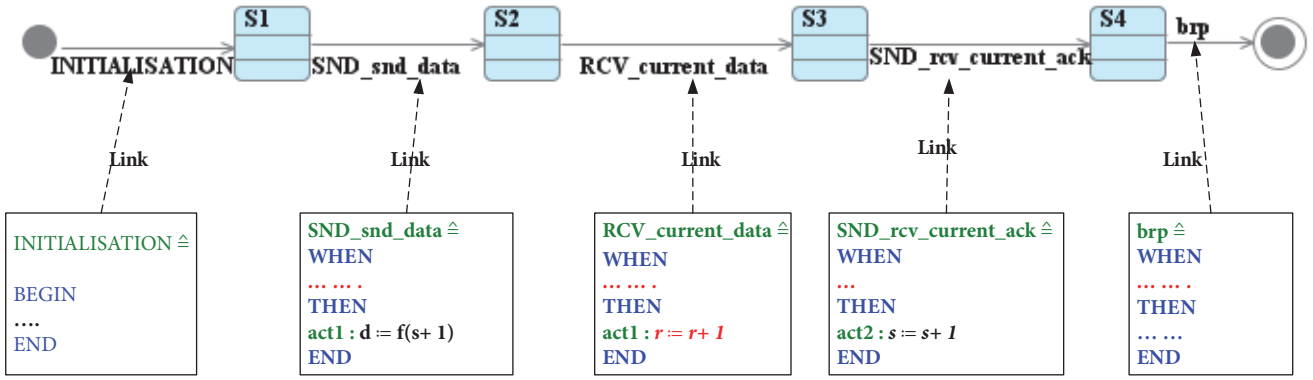


Figure 2: iUML-B state machine control flow modeling principle.

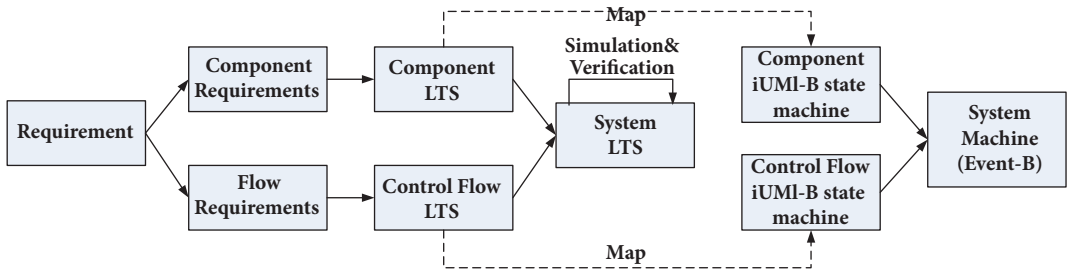


Figure 3: Event-B machine behavior modeling strategy.

(1) The system requirements are decomposed into component requirements and control flow requirements, where component requirements represent the constraints of the event order of a component itself and the flow requirements represent the event order between components.

(2) The behavior LTS of each component, which contains only the events that occurred in the component itself and the local variables of the component itself, is established according to the component requirements.

(3) The system control flow LTS, which describes event order between multiple components, is established according to the control flow requirements.

(4) The component LTS and control flow LTS are composed to establish the system LTS. Then we can simulate and analyze the event traces of the system LTS in the LTSA [9] environment and modify it until it satisfies our requirements.

(5) A component LTS is mapped onto an iUML-B state machine (we call it the component state machine). A system control flow LTS is mapped onto an iUML-B state machine (we call it the flow state machine), too.

(6) The Event-B machine which meets our requirements is generated using the component state machine and flow state machine.

The proposed modeling process is shown in Figure 3.

In order to prove the correctness of our method, we give a complete proof of bisimulation equivalence between the LTS system and Event-B model. We first give the definition of LTS and its composition.

Definition 1 (LTS). Let *States* represent a universal set of states, *Acts* represents a universal action set, and then a LTS P is defined as a quaternion $P = \langle Q, \Sigma, \Delta, q\rangle$, where

(i) $Q \subseteq States$, representing the state set of P;

(ii) $\Sigma = \alpha P$ ($\alpha P \subseteq Acts$), representing the action set of P;

(iii) $\Delta \subseteq Q \times \Sigma \times Q$, representing the transition relationship in P; these transitions are labeled with the elements in Σ;

(iv) $q \in Q$, representing the initial state of P.

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

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

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

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

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

We use the bisimulation definition in the literature [10].

Definition 3 (bisimulation equivalent). Let LTSi = $(Qi, \Sigma_i, \Delta i, qi)$, $i = 1, 2$, be labeled transition systems over the set Σ of actions. An *bisimulation* for $(\mathrm{LTS}_1, \mathrm{LTS}_2)$ is a binary relation $R \subseteq Q1 \times Q2$ such that

(A) for the initial state $q1$ and $q2$, $(q_1, q_2) \in R$

(B) for any $(s_1, s_2) \in R$ it holds that

(1) if $s_1 \xrightarrow{a}_1 s_1'$, then $s_2 \xrightarrow{a}_2 s_2'$ with $(s_1', s_2',) \in R$ for some $s_2' \in Q2$

(2) if $s_2 \xrightarrow{a}_2 s_2'$, then $s_1 \xrightarrow{a}_1 s_1'$ with $(s_1', s_2',) \in R$ for some $s_1' \in Q1$

LTS_1 and LTS_2 are *bisimulation equivalent* (or bisimilar), denoted as $\mathrm{LTS}_1 \sim \mathrm{LTS}_2$, if there exists an bisimulation R for $(\mathrm{LTS}_1, \mathrm{LTS}_2)$. The bisimulation equivalence relationship is transmitted, i.e., $\mathrm{LTS}_1 \sim \mathrm{LTS}_2 \wedge \mathrm{LTS}_2 \sim \mathrm{LTS}_3 \Longrightarrow \mathrm{LTS}_1 \sim \mathrm{LTS}_3$.

We collectively refer to component LTS and control flow LTS as "atomic LTS." The state of each "atomic LTS" is the state of a single variable, not the composition of other LTSs. Therefore, a system's LTS is a composition of multiple atomic LTSs:

$$\mathrm{LTS}\left(\mathrm{system}\right) = \parallel_{i=1}^{n} \mathrm{AtomicLTS}_i, \tag{4}$$

where "$\parallel$" is the LTS composition operator, i is the sequence number of the atomic LTS, and n is the total number of atomic LTSs.

The proof process is as follows.

(1) We first establish an "atomic iUML-B state machine" based on the atomic LTS $AtomicLTS = \langle Q, \Sigma, \Delta, q \rangle$. The construction process of atomic iUML-B state machine is as follows.

(a) An "atomic iUML-B state machine" is defined as $AtomicStm = \langle Node, E, Edge, InitNode \rangle$, where $Node$ represents a set of nodes in the iUML-B state machine; E represents the set of events that linked on the iUML-B state machine's edge. $Edge \subseteq Node \times E \times Node$ represents the set of edges in the iUML-B state machine; $InitNode$ represents the initial node of the iUML-B state machine, which is the target node of the edge that is linked to the *Initialization* event.

(b) In the process of establishing the atomic iUML-B state machine, let $Node = Q$, $E = \Sigma$, $Edge = \Delta$, $InitNode = q$. For example, if there is a state s_1 (transition t) in $AtomicLTS$, a node s_1 (edge e) is also drawn in the corresponding "atomic iUML-B state machine." This mapping process is very easy to operate and we will not explain it further.

(c) Generate the Event-B code for this atomic iUML-B state machine using the automatic code generation tool of the Rodin platform.

(2) In the following we prove that the Event-B model generated by *AtomicStm* is bisimulation equivalent to *AtomicLTS*.

We define an Event-B model as $M = \langle V, Event, Guard, Action, V_{init} \rangle$, where V represents variables set of M, $Event$ represents event set of M, $Guard$ represents guard set of M, $Action$ represents action set of M, and V_{init} represents the initial value set for each element in the V. We define the LTS corresponding to the M as $\mathrm{LTS}(M) = (Q_M, \Sigma_M, \Delta_M, q_M)$.

We named an Event-B model generated by *AtomicStm* as $M_A = \langle V_A, Event_A, Guard_A, Action_A, V_{initA} \rangle$. It should be emphasized that at this time there is only one element v in V_A, and V_{initA} is the initial value of v (because an *AtomicStm* only describes the change of one variable). $Event_A$ represents events that modify the value of v, and $Guard_A$ represents those guards that contain v in the *when* clause of an event. Similarly, $Action_A$ represents actions that modify the value of v. We assume that the type of v is D, that is, $v \in D$; then the state space of v is D.

Since M_A's code is generated by *AtomicStm*, we have $\mathrm{LTS}(AtomicStm) \sim \mathrm{LTS}(M_A) = (Q_{MA}, \Sigma_{MA}, \Delta_{MA}, q_{MA})$. If an Event-B model M is composed of multiple atomic iUML-B states machine generated together, then we have

$$\mathrm{LTS}\left(M\right) = \parallel_{i=1}^{n} \mathrm{LTS}\left(M_{Ai}\right), \tag{5}$$

where i is the sequence number of the atomic iUML-B state machine, and n is the total number of atomic iUML-B state machines.

We explain the equivalence between *AtomicLTS* and M_A according to Rodin's rules for generating Event-B code from the iUML-B state machine.

(a) First, Rodin will generate a variable based on *one AtomicStm* and automatically generate a *SET* which contains all possible values of this variable. For example, an *AtomicStm* named "node" which contains n nodes (e.g., $s_1, s_2, \ldots, s_n$) will generate

$$\mathrm{partition}\left(\mathrm{Node}, \{s_1\}, \{s_2\}, \ldots, \{s_n\}\right) \tag{6}$$

which means $node \in Node$. As we mentioned in (1) (b), Node = Q. Therefore, the state space D of variable "node" of Event-B model M_A is equal to Q, and then $Q_{MA} = Q$.

(b) Secondly, Rodin generates the following code based on the edge that links the *Initialization* event:

$$\mathrm{INITIALISATION} \mathrel{\hat{=}} \mathrm{BEGIN\ node} = s_1\ \mathrm{END} \tag{7}$$

Since $s_1 = q$ (in the construction process (1) (b)), we have $V_{init} = q$, and then $q_{MA} = q$.

(c) Rodin will generate an event named "event_i" in the Event-B model M_A according to the event "event_i" which has been linked on the edge of iUML-B state machine, and will generate the following code according to each edge from the node s_i to s_j (where s_i and s_j are the node name):

$$\mathrm{event\_}i \mathrel{\hat{=}} \mathrm{when\ node} = s_i\ \mathrm{then\ node} \coloneqq s_j \tag{8}$$

Therefore, for a transition $s_i \xrightarrow{a} s_j$ in *AtomicLTS*, there will be a corresponding transition $s_i \xrightarrow{\text{event\_}i} s_j$ in LTS(M_A). At the same time, for each action a in *AtomicLTS*, there will be an event event_i corresponding to it in the Event-B model M_A. So we have $\Delta_{MA} = \Delta$.

(d) The reverse mapping process from M_A to *AtomicLTS* is similar, and we will not repeat them here.

(e) We can define a mapping relationship R so that *AtomicLTS* and LTS (M_A) comply with the requirement of bisimulation equivalence. In fact, this R can be a renamed function, such as R (Sender) = sender; now we get

$$LTS\,(AtomicStm) \sim LTS\,(M_A) \sim AtomicLTS \tag{9}$$

(3) Finally, we use the theorem in the literature [10].

Lemma 4 (congruence w.r.t LTS composition). *For labeled transition systems* LTS_1 *and* LTS_1' *over* Σ_1, LTS_2 *and* LTS_2' *over* Σ_2, *and* $H \subseteq \Sigma_1 \cap \Sigma_2$, *it holds that*

$$\mathrm{LTS}_1 \sim \mathrm{LTS}_1',$$

$$\mathrm{LTS}_2 \sim \mathrm{LTS}_2' \tag{10}$$

$$\textit{implies } \mathrm{LTS}_1 \|_H \mathrm{LTS}_1' \sim \mathrm{LTS}_2 \|_H \mathrm{LTS}_2'$$

According to expression (4) and expression (5) and Lemma 4, we have

$$\|_{i=1}^{n} \mathrm{LTS}\,(M_{Ai}) \sim \|_{i=1}^{n} \mathrm{AtomicLTS}_i \tag{11}$$

That is, the LTS model LTS (System) of the system and the LTS model *LTS*(M) of Event-B model obtained according to the mapping rule of (1) are bisimulation equivalent:

$$\mathrm{LTS}\,(M) \sim \mathrm{LTS}\,(\mathrm{system}). \tag{12}$$

3. BRP Case Study

In this section, we take Abrial's BRP protocol model in Chapter 6 of [1] as an example. We use our proposed method to establish the system component model and control flow model step by step and finally get a BRP protocol control flow model in Event-B.

3.1. System Overview. The purpose of BRP is to transfer sequential files from the sender to the receiver. After the transfer is complete, the recipient's file should be equal to the sender's original file. The sender should split the files that need to be sent into a series of data blocks and send the data blocks to the receiver in order. Once the receiver receives the data item, it stores it in its own file and sends an acknowledgement message to the sender on an acknowledgement channel. The sender will send the next data item after receiving this acknowledgement message. The principle of the BRP protocol is shown in Figure 4.

At the beginning of modeling process, there are only two components in the system, Sender and Receiver. During the refinement process, we add the new event and component LTS to model; then we add control flow LTS according to the flow requirement. We present the events and requirements for each level of refinement, as shown in Table 1.

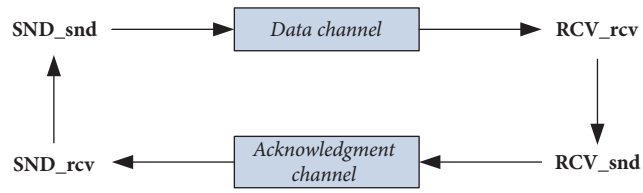


Figure 4: The principle of bounded retransmission protocol.

3.2. Modeling Process. In the modeling process, we represent component LTS with *SN*, *RN* (N is the refinement level) and represent the flow requirement with *FlowM* (M is the requirement number). The iUML-B component state machine is represented with *SNDMachineN*, *RCVMachineN* (N is the refinement level), and the iUML-B control flow state machine is represented with *FlowM* (M is the flow requirement number). We use finite state process (FSP) [9] to represent the LTS and use LTSA tool to simulate and verify the LTS.

FSP (finite state processes) is a process algebra that can describe LTS. The FSP operators used in this paper are as follows.

(1) Order: "->", indicating the order of the action, such as x -> P means that the action x is executed first, and the action sequence described by the process P is executed.

(2) Select: "|", that is, the choice of different execution actions, such as x -> P | y -> Q said: the first action x, and then follow the description of P; or the first action y, and then follow the description of Q.

(3) Recursion: the behavior of a process may be defined in terms of itself, in order to express repetition.

(4) End state: "END": describes a process that has terminated successfully and cannot perform any more actions.

(5) Condition control: "when", that in given conditions to perform a certain action, such as *when* B x-> P | y-> Q said: if the condition B is established, in addition to y -> Q can choose to perform action x and then execute the action sequence of process P; if B is not established, only the action y can be executed, and then the action sequence of process Q is executed. Here, B is a state predicate.

(6) Parallel composition: "||", representing a composition of multiple simple processes, such as P || Q means that the sequence of actions in processes P and Q is executed concurrently where P and Q are also called subprocesses.

System Abstraction Model. The system abstract model includes only three events, which represent the sender's action *(SND_progress)*, the receiver's action *(RCV_progress)*, and the communication completion event (brp), respectively. We build the sender and the receiver's component LTS according to Com0.1 and Com0.2, as shown in Figures 5(a) and 5(b). The corresponding FSP is expressed as (in FSP, action names cannot start with uppercase letters, so we can only express actions with lowercase letters. The mapping rule is: *snd_XXX* action in FSP corresponds to event *SND_XXX* in Event-B):

S0 = (snd_progress-> BRP), BRP = (brp-> END).

R0 = (rcv_progress-> BRP), BRP = (brp-> END).

Table 1: System requirements.

	Introduced Events	System Requirements.
Level0	SND_progress, RCV_progress, brp	*Component Requirements* Com0.1: The event brp cannot occur unless the sender has completed the event SND_progress. Com0.2: The event brp cannot occur unless the event RCV_progress has been completed. *Flow Requirements* Flow0.1: SND_progress, and RCV_progress can occur interleaving in arbitrary. Flow0.2: The event brp cannot occur unless events SND_progress and RCV_progress have both completed.
Level1	SND_success, SND_failure, RCV_success, RCV_failure.	*Component Requirements* Com1.1: After the sender is initialized, it may send successfully or may fail to send, and then the brp event occurs. Com1.2: After the receiver is initialized, it may be successful to receive or fail to receive, and then the brp event occurs. *Flow Requirements* Flow1.1: When the system is initialized, only event SND_failure or RCV_success can occur. Flow1.2: Sender failed event will cause the receiver to fail. Flow1.3: Receiver succeed event will cause the sender to succeed.
Level2	RCV_rcv_current_data	*Component Requirements* Com2.1: If the received data is not the last data, the event RCV_current_data occurs. If the received data is the last data, the event RCV_success occurs.
Level3	SND_snd_data, SND_timeout, SND_rcv_curr_ack	*Component Requirements* Com3.1: The event SND_snd_data occurs if the message to be sent is not the last message. Com3.2: If the message to be sent is the last message, the event SND_success occurs. Com3.3: The sender may receive an acknowledgement message only after the data has been sent. Com3.4: After sending the data, the sender may occur a time out event because the acknowledgment message has not been received. Com3.5: The sender will send the next data after receiving the acknowledgment message. Com3.6: The sender will send the current data again after a time out event occurs. *Flow Requirements* Flow3.1: The receiver may accept the current data only after the sender has sent the data. Flow3.2: After the sender sends the data, the event RCV_success may occur (that is, accept the last data).

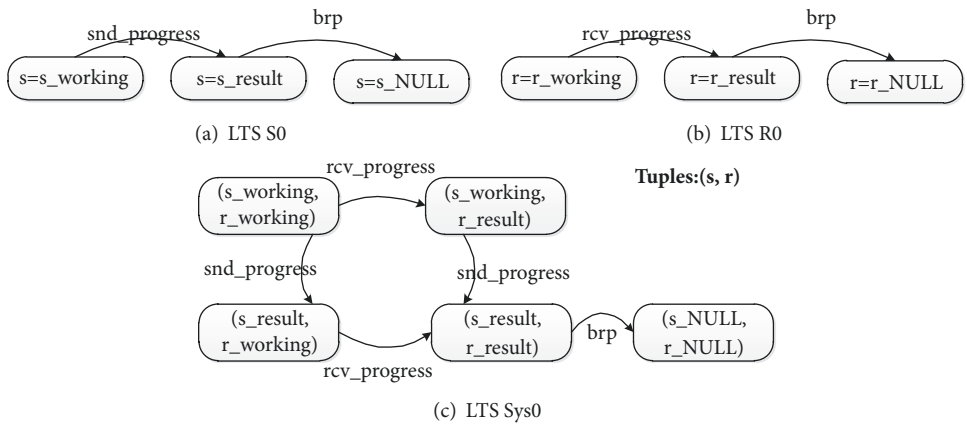


Figure 5: The LTS of the abstract model and the composition of the component LTS.

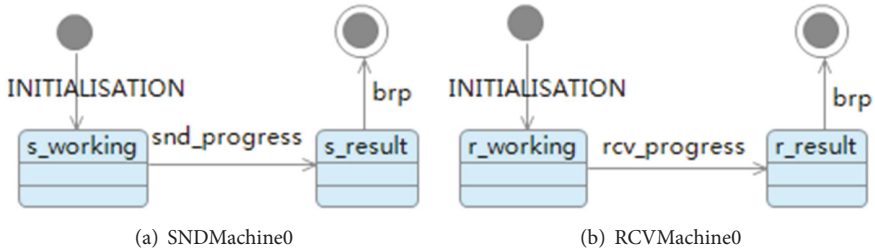


Figure 6: System abstract model in iUML-B.

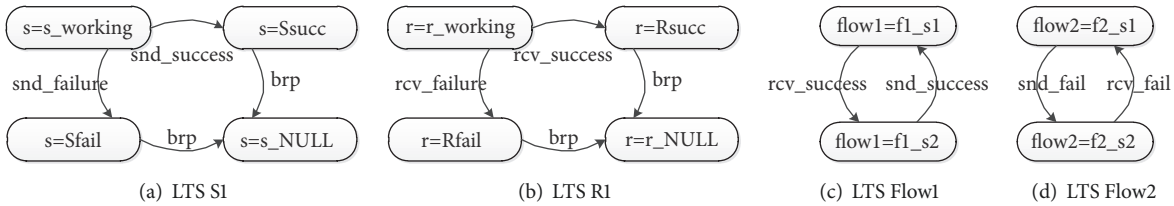


Figure 7: The component LTSs and flow LTSs of the first refinement model.

We found that the composition of the sender component LTS and the receiver component LTS was able to meet the flow requirements of Flow 0.1 and Flow 0.2, so we did not need to add any control flow LTS. We get the system LTS of abstract model using LTS composition; as Figure 5(c) shown, the FSP expression is

$$\| Sys0 = (S0 \| R0).$$

We map the Level0 LTS model onto the iUML-B state machine and get the two state machines as shown in Figure 6.

First Refinement. For the introduced events of the first refinement, we first add them to the two component LTS according to Com1.1 and Com1.2, as shown in Figures 7(a) and 7(b). The FSP code is

R1 = (rcv_success->Rsucc | rcv_failure->Rfail), Rsucc = (brp-> END), Rfail = (brp-> END).

S1 = (snd_success->Ssucc | snd_failure->Sfail), Ssucc = (brp-> END), Sfail = (brp-> END).

Flow1.2 and Flow1.3 constrain the enable relationship between the sender and the receiver in the first refinement, while Flow1.1 constrains which event occurs first when the system is initialized. Combined with these three flow requirements, we get the control flow LTS as Figures 7(c) and 7(d) show. The FSP code is

Flow1 = (rcv_success->snd_success->Flow1).

Flow2 = (snd_failure->rcv_failure->Flow2).

It should be noted that LTS Flow1 not only restricts that a sender's failure will cause a receiver's failure, but also restricts that when the system is initialized, only the event *snd_failure* may occur. If we put *rcv_failure* before the event *snd_failure* in LTS Flow1, this LTS will violate the flow requirement. We get the system-level LTS of Level1 as shown in Figure 8, and its FSP expression is

$$\| Sys1 = (S1 \| R1 \| Flow1 \| Flow2).$$

We map the LTS models of Level1 to the iUML-B state machine and get the four state machines as shown in Figure 9,

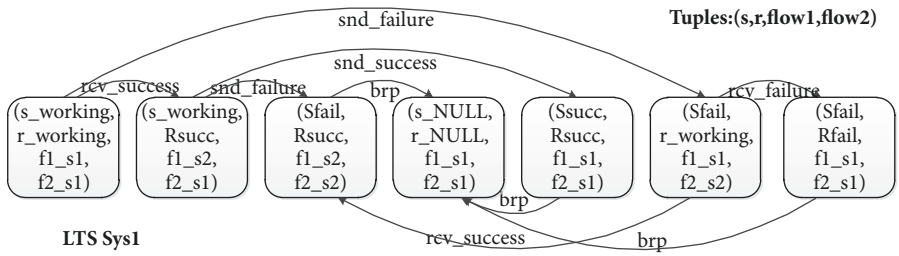
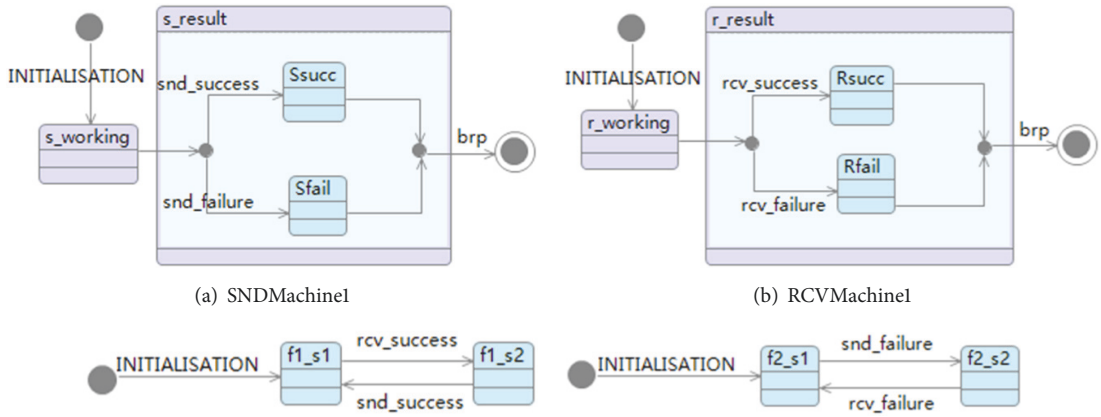


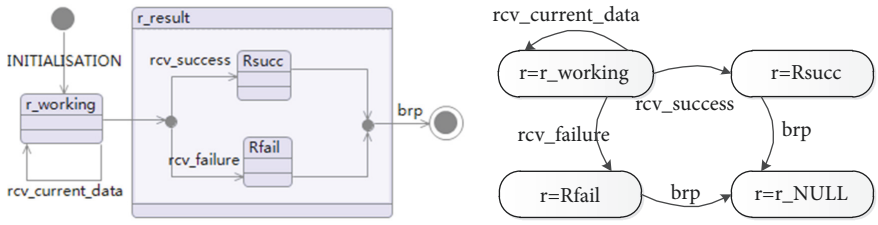
Figure 8: The system LTS of first refinement.



(a) SNDMachine1 (b) RCVMachine1

(c) Flow1 state machine (d) Flow2 state machine

Figure 9: iUML-B state machines of first refinement.



(a) RCVMachine2 (b) LTS R2

Figure 10: The iUML-B state machine RCVMachine2 and LTS R2.

where *SNDMachine1* and *RCVMachine1* are the component state machines of the sender and receiver, respectively, and Flow1 and Flow2 correspond to LTS Flow1 and LTS Flow2, respectively.

Second Refinement. The second level refinement only introduces one event *rcv_current_data*, which does not involve the flow requirement associated with the sender, so we do not need to modify the sender component but only need to add event *rcv_current_data* to the receiver's component LTS, as Figure 10(b) shows. The FSP code is

const N = 2

range T = 0⋯N

R2 = R2 [0],

R2 [r: T] = (when (r + 1 <N) rcv_current_data [r] -> R2 [r + 1]

| when (r + 1 == N) rcv_success->Rsucc

| rcv_failure->Rfail),

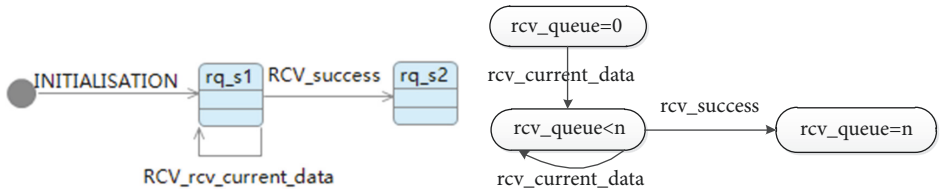
Rsucc = (brp-> END),

Rfail = (brp-> END).

In the above codes, we introduce the message counter *r* of the receiver which represents the number of messages currently received. The constant *N* represents the total number of messages.

We map Level2's LTS model to the iUML-B state machine and get the new receiver state machine as shown in Figure 10(a).

In addition, we find that *r* is a newly introduced variable, so we must model its changes using a new atomic LTS. We introduce a new atomic LTS RCV_QUEUE to express the change of *r:*



(a) Rcv_queue state machine (b) LTS Rcv_queue

Figure 11: The LTS and the corresponding iUML-B state machine of *RCV_QUEUE*.

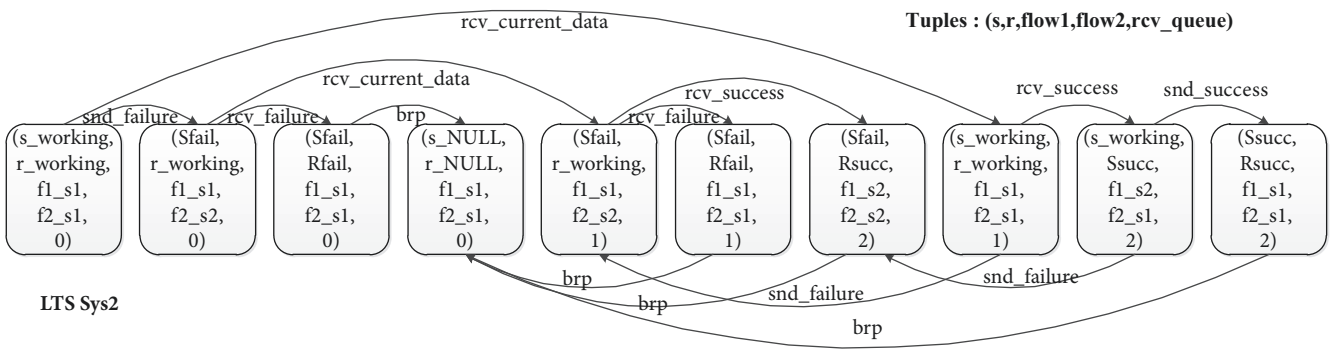


Figure 12: The system LTS of second refinement.

RCV_QUEUE = RCV_QUEUE [0][0],

RCV_QUEUE [s:T][r:T] = (when (r+1<N) rcv_current_data ->RCV_QUEUE [s][r+1]

| when (r+1==N)rcv_success-> RCV_QUEUE [s][r+1]).

The LTS and the corresponding iUML-B state machine of *RCV_QUEUE* are shown in Figure 11.

This system-level LTS of level2 is shown in Figure 12; the FSP expression is

$$\| Sys2 = (S2 \| R2 \| Flow1 \| Flow2 \| RCV\_QUEUE),$$

where *S2 = S1*.

At this level of refinement, there are no changes in the sender's LTS and control flow LTS, so we do not need to modify the corresponding iUML-B state machine.

Due to space limitations, we omitted the description of LTS model and iUML-B state machine model of the third layer refinement.

3.3. Property Verification of the Event-B Model. We use the iUML-B state machine to generate the BRP protocol control flow model in Event-B. These automatically generated control flow models have covered all the control variables of the fourth layer of Abiral's BRP model. Thanks for iUML-B, we can refine the original component state machine and the flow state machine by adding new substates in the original state in the refinement of each layer and preserve the consistency of the Event-B model by refinement checks to ensure that concrete machine is the effective refinement of abstract machine.

As we proved in Section 2.3, the Event-B model we constructed is bisimulation equivalent to the original LTS model. Therefore, we can verify the properties of the Event-B model by verifying the properties of the original LTS model. We formalized the requirements of the BRP system of Table 1 using linear temporal logic (LTL) formulate; some of them are shown in Table 2. In Table 2, the symbol "□", "U", and "◇" represent "always", "until", and "eventually" in linear temporal logic, respectively, while the symbols "→", "¬" and "∧" represent "implication", "negative", and "conjunction" in proposition logic, respectively.

We verified these properties with the help of the LTSA tool and the results show that the behavior of our model meets the requirements in Table 1.

4. Discussion

There are four types of variables in Event-B model: Boolean variables (such as $s \in \{\text{TRUE, FALSE}\}$), variables with a finite number of states (such as $s \in \{s_1, s_2, \ldots, s_n\}$), data variables such as $a \in N$ or $a \in Z$), and collection-type variables (e.g., $g \subseteq D$). As we can see, it is very easy to model the first two kinds of control variables using LTS. For data variables whose state space is an infinite set, we can use the following methods to model its changes: first, we bound the range of variable a using its lower bound and upper bound. That is, let $a \in$ [lower bound, upper bound]; second, we divide the state space of a into three subsets according to the equivalence class: $\{a =$ lower bound$\}$, $\{$lower bound $< a <$ upper bound$\}$, and $\{a =$ upper bound$\}$. In this way, we can convert the infinite state space of data variable into a finite number of states and further establish the LTS model of this data variable. In fact, we have used this approach when we modeled Rcv_queue changes in the second level of refinement. However, the above

Table 2: The LTL formulate of basic requirements.

Requirement number	LTL formulate
Com0.1, Com0.2, Flow0.2	□(¬brp U (snd_progress∧rcv_progress))
Flow1.2	□(snd_failure→ ◇rcv_failure)
Flow1.3	□(rcv_success→ ◇snd_success)

Table 3: Comparison of major control flow modeling methods.

Method	Ability		
	Formal behavior semantics	Way of expression	Convertible to LTS
ERS	No	Tree structure	No
Flow Method	No	Events and relationships	No
CSP‖B	Yes	Process algebra	Yes
LTS+iUML-B	Yes	States and transitions	Yes

method only works when we already know the range of data variable (such as the BRP model in this paper). For those data variables that we cannot determine their boundary, we need further research to model them. In addition, we still have not found an effective way to model changes of collection-type variables.

5. Related Work

There has been some research on explicitly simulating the control flow of the Event-B model. Fathabadi et al. [11] proposed a method named "Event Refinement Structure" (ERS) method which uses a tree structure based on the Jackson structure diagram (JSD) to express the control flow of Event-B model. However, one cannot establish the equivalence relationship between the tree structure of ERS and LTS. Therefore, it is also impossible to verify the behavioral properties of the ERS method. In contrast, our work uses the LTS and iUML-B state machine to visually express the control flow of the Event-B model and can verify its behavioral properties easily. Iliasov [12] proposed a method named flow language which uses *ena*, *dis*, and *fis* to express the order of the events. However, the modeling elements of the flow method are the events and the relationships between events, rather than states and transitions of the state transition system. This makes flow language difficult to map onto LTS semantic model. Compared with the flow method, the iUML-B state machine uses the state-based style to express the control flow. This makes the control flow easier for engineers to understand.

The combination of CSP and Classic B is also studied in [13, 14]. For the explicit control flow modeling of the Event-B, CSP‖B method [15] proposed an integrated formal method that combines Event-B as a state-based formal system and CSP as a control-based formal system to model the control flow in Event-B. Our approach is inspired by the CSP ‖ B method, but we use FSP to model the control flow of Event-B. Compared to the CSP ‖ B method, we systematically consider all the control variables that may affect the control flow of the Event-B machine. This makes our results of behavioral properties verification more reliable than CSP ‖ B. We compare our method (named "LTS+iUML-B") with other methods in this field, as shown in Table 3.

6. Conclusion

The Event-B model has some flaw in expressing the model's control flow. In this paper, we use LTS to express Event-B control flow, which makes the control flow of Event-B model become visible and makes the verification of Event-B model's behavior properties easier. We model each variable in the Event-B model that affects the model execution process as a single "atomic LTS" and use the LTS composition operation to obtain the control flow model of the entire system. By using our method, the modeler can easily observe and analyze the behavior of the system with the help of the LTSA tool. At the same time, we map the LTS model onto the Event-B model and prove the bisimulation equivalence between the original LTS model and the corresponding Event-B model. In this way, the modeler can verify the behavior of the Event-B model by verifying the properties of the original LTS model.

At present, our modeling method is only suitable for Event-B's Boolean variables, bounded state variables, and certain data variables. In the future, we will study how to use LTS to model the change of the collection-type variables in the Event-B model.

Conflicts of Interest

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

References

[1] E. Boiten, *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, 2010.

[2] J. R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, Cambridge, UK, 2005.

[3] R. J. R. Back and R. Kurki-Suonio, "Distributed Cooperation with Action Systems," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 10, no. 4, pp. 513–554, 1988.

[4] J. Chen, C. Du, and P. Han, "Scheduling independent partitions in integrated modular avionics systems," *PLoS ONE*, vol. 11, no. 12, Article ID e0168064, 2016.

[5] J. Chen, C. Du, F. Xie, and Z. Yang, "Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems," *Real-Time Systems*, vol. 52, no. 3, pp. 239–271, 2016.

[6] M. Y. Said, M. Butler, and C. Snook, "A method of refinement in UML-B," *Software and Systems Modeling*, vol. 14, no. 4, pp. 1557–1580, 2015.

[7] 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," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 9675, pp. 360–375, 2016.

[8] 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.

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

[10] C. Baier and J.-P. Katoen, *Principles of model checking*, MIT Press, Cambridge, MA, 2008.

[11] 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, no. 3, pp. 499–523, 2015.

[12] A. Iliasov, "Use Case Scenarios as Verification Conditions: Event-B/Flow Approach," in *Software Engineering for Resilient Systems*, vol. 6968 of *Lecture Notes in Computer Science*, pp. 9–23, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[13] M. Butler, "csp2B: A practical approach to combining CSP and B," *Formal Aspects of Computing*, vol. 12, no. 3, pp. 182–198, 2000.

[14] S. Schneider and H. Treharne, "Verifying controlled components," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 2999, pp. 87–107, 2004.

[15] S. Schneider, H. Treharne, and H. Wehrheim, "A CSP Account of Event-B Refinement," *Electronic Proceedings in Theoretical Computer Science*, vol. 55, pp. 139–154, 2011.