Hindawi
Scientific Programming
Volume 2021, Article ID 6672717, 21 pages
https://doi.org/10.1155/2021/6672717

Research Article

A Time Refinement Framework Based on iUML-B State Machine

Han Peng,[1] **Xiaoli Zhang**,[1] **Guozhen Cao**,[1] **Zhouzhou Liu**,[1] **Yuejuan Jing**,[1] **and Lei Rao**[2]

[1]*Xi'an Aeronautical University, Xi'an, China*
[2]*Hiroshima University, Hiroshima, Japan*

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

Received 9 November 2020; Revised 13 February 2021; Accepted 10 March 2021; Published 25 March 2021

Academic Editor: Shah Nazir

Event-B is a formal modeling language that is very suitable for software engineering, but it lacks the ability of modeling time. Researchers have proposed some methods for modeling time constraints in Event-B. The limitations with existing methods are that, first of all, the existing research work lacks a systematic time refinement framework based on Event-B; secondly, the existing methods only model time in the Event-B framework and cannot be smoothly converted to automata-based models such as timed automata that facilitate the verification of time properties. These limitations make it more difficult to model and verify real-time systems with Event-B because it is very time-consuming to prove time properties in the Event-B framework. In this paper, we firstly proposed a systematic time refinement framework to express and refine time constraints in Event-B. Secondly, we also proposed various vertical refinement patterns and horizontal extension patterns to guide modelers to refine the Event-B real-time model step by step. Finally, we use a real-time system case to demonstrate the practicality of our method. The experimental results show that the proposed method can make the real-time system modeling in Event-B more convenient and the models are easier to convert to the timed automata model, thereby facilitating the verification of various time properties.

1. Introduction

The complexity of the real-time concurrent system makes it necessary for engineers to model and verify the system using formal methods. The formal method enables researchers to use strict mathematical models to describe the system requirements and system behavior and verify whether a given system or system model meets all required properties. Event-B [1] is a formal language that is closest to software engineering at present and is very suitable for modeling the behavior of real-time concurrent systems. The stepwise refined framework and automatic code generation capabilities of Event-B not only ensure the correctness and consistency of the model but also provide good support for software engineering. However, Event-B itself does not support the modeling of time, and its ability of expressing time has some limitations; therefore it is difficult for Event-B to verify the time properties of real-time systems.

In order to model and verify real-time systems using Event-B, the work done by the researchers includes the following: the first direction is to use the capabilities of the Event-B itself to model time, which is usually modeling discrete clock ticking using a "tick_tock" event as well as modeling various time patterns; the other direction is to translate the Event-B model to a verifiable time model, such as the UPPAAL model, and then verify its TCTL properties.

The abovementioned research has solved some problems of time modeling in Event-B. However, these works still have some problems: first, the current work lacks a systematic and comprehensive time refinement framework. The problems they have solved are specific and detailed questions. Secondly, although there are some methods that can systematically model time in the Event-B framework, they make the proof of time properties more difficult and onerous, too. According to the viewpoint of Dominique Cansell et al. [2], it is very difficult and time-consuming to verify the time properties in the Event-B model. Third, there are some problems to translate the Event-B model to UPPAAL timed automata directly, because many syntactic elements in Event-B modeling language which are based on first-order predicate logic cannot be directly converted into timed automata.

In short, the current formal modeling and verification work of real-time systems puts forward such requirements on Event-B: first, we need a framework that supports a top-down time modeling and refinement process. Secondly, we need a "middle model" to make the translation process from Event-B to timed automata "smoother" to support the verification of the time properties of the Event-B model.

The invention of the iUML-B state machine [3] enables researchers to describe the stepwise refinement process of the Event-B model in an intuitive and visual way. It also enables us to express the time interval in Event-B graphically. We can use the iUML-B state machine to "mimic" the behavior patterns of the timed automata so that people can easily translate the iUML-B state machine into the corresponding timed automata model and verify the time properties of the Event-B model easier.

The "timed automata patterns" [4] method proposed by Dong also enlightens us. Based on his experience in modeling real-time systems, Dong proposed more than dozen timed automata patterns, which represent the common problems when modeling real-time systems. We introduced Dong's ideas into the Event-B modeling framework and used iUML-B state machines to express various timed automata patterns to support the modeling and verification of real-time systems in Event-B.

The contributions of this paper include the following: first, we proposed a time refinement framework of the Event-B model based on the iUML-B state machine. It provides a guiding framework for modeling real-time systems using Event-B. Secondly, we proposed "vertical refinement pattern" and "horizontal refinement pattern" based on timed automata patterns proposed by Dong. These patterns can help the modeler to construct a real-time Event-B model quickly. Third, we proposed the concept of the iUML-B functional state machine and iUML-B clock state machine to model the untimed part and timed part of the system, respectively. Finally, we modeled a pacemaker system using our method and compared our method with the existing real-time Event-B method.

The rest of this paper is arranged as follows: in Section 2, we introduce the various methods to model and verify real-time systems using Event-B and other formalisms. In Section 3, we introduce the basic knowledge necessary for this paper, including the introduction of Event-B, iUML-B state machine, and the definition of timed automata and timed automata patterns. Section 4 details the methodology framework for this paper, including various iUML-B pattern state machines corresponding to various timed automata patterns based on the methodology we proposed. In Section 5, we use the framework proposed in this paper to construct a real-time model of the pacemaker system. The method of this paper is evaluated in Section 6. In Section 7, we concluded our works.

2. Related Works

As we have mentioned in Section 1, there are two main directions in modeling and verifying the real-time system using Event-B. Researchers have already done some work in using Event-B itself to model real-time systems. Butler and Falampin [5] first proposed a method to model discrete-time in classic B. This method uses a natural number variable (named clock variable) to express the current time and express the elapse of time by increasing the value of this variable. The biggest difference between Butler's method and the classic time modeling method is that in the former method if the current time is equal to one of the deadlines, certain operations are used to prevent the elapse of time. This idea has been adopted by all subsequent similar works including our method. Cansell [2] first proposed the concept of "time constraint patterns" and used a "tick_tock" event to express the elapse of time. Subsequently, Joris Rehm [6–8] proposed a "duration pattern" to express the time interval between two events, which enable people to model and reason about real-time properties in Event-B. Sarshogh [9, 10] proposed a method named "Timing Properties", which includes three time patterns: Delay pattern, Expiry pattern, and Deadline pattern. These patterns are used to express the concepts of delay, timeout, and deadline of real-time systems. Sulskus [11, 12] proposed a time interval method based on Sarshogha's work. The time interval method used the state node of the iUML-B state machine to represent the time interval and used the input and output edges of the state node to represent the trigger event and response event of the time interval, respectively. The abovementioned methods are very helpful for modeling the real-time properties of the system, but the verification of time properties still requires heuristic proofs in the theorem proving framework. Therefore, as we have mentioned in Section 1, this places a great burden on researchers to model and verify the system.

The other direction of modeling and verifying real-time system using Event-B is to translate the Event-B model into a UPPAAL timed automata. For example, Alexei Iliasov [13] proposed a method to translate the Event-B model to a timed automata model which used the process view as the intermediate representation language between the Event-B model and the UPPAAL model. The limitation of this method is that it needs to extract the process view from the Event-B model before translation. Juri Vain and Jesper Berthing et al. [14–16] also proposed a time refinement framework that supports the Correct-By-Construction method which integrated Event-B and UPPAAL timed automata. Their method starts with a simple Event-B model and its corresponding UPPAAL timed automata. After the refinement of Event-B model and the proof of consistency, the Event-B model is translated into a corresponding timed automata model, and verification of time properties is performed in the UPPAAL environment. The final model (including Event-B model and timed automata model) of the system is got gradually by adopting this iterative process. The advantage of this method is that it does not require any modifications to the Event-B model and timed automata model, nor does it need to learn new formal systems. The shortcoming of this method is that many elements in Event-B modeling language are based on first-order predicate logic which cannot be directly converted into timed automata.

3. Preliminary

3.1. Event-B and iUML-B State Machine

3.1.1. Event-B. Event-B is an event-based formal modeling language which uses the event to be its first class object. The general form of an event is as follows:

$e \triangleq$ WHEN guards THEN actions END

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

3.1.2. iUML-B State Machine and Its Refinement. C. Snook invented UML-B [17], a "UML-like" graphical front end for Event-B. UML-B uses the class diagrams and state diagrams familiar to the software engineers and system engineers to model the system. System model expressed by UML-B can generate the corresponding Event-B code directly on the Rodin platform. Recently, UML-B has evolved into iUML-B (integrated UML-B), allowing 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 [3, 18].

Although UML-B was originally used as a UML profile [17] to realize the translator from UML class diagram and state diagram to "classic B", it has evolved into an independent formal modeling language since Snook defined the UML-B metamodel [19]. Since then, the decomposition and refinement process of the UML-B model corresponds to the decomposition and refinement process of the Event-B model, rather than the decomposition and refinement of the UML model defined by the object management organization (OMG).

The difference between the state decomposition of UML and that of UML-B is that the decomposition of UML-B needs to follow a lot of constraints. In addition, modelers have to prove the consistency between the abstract model and the refined model. Said [20–23] proposed many rules for state refinement of UML-B, which are the basis of various refinement studies in this paper. With the support of Said's work, the decomposition and refinement of the UML-B model must follow certain formal rules, and some proof obligations (PO) automatically generated by the iUML-B plug-in must be "discharged"(i.e., be proven) to ensure the consistency between the abstract model and the concrete refined model; otherwise, the refinement process cannot proceed.

3.2. Timed Automata and Timed Automata Patterns

3.2.1. Timed Automata and Its Definition. A timed automaton is a finite state machine with a clock. It is very suitable for modeling the behavior of real-time systems and verification of time properties. It provides a general method for annotating state transition diagrams with timing constraints using a limited number of clock variables. In the formal system of timed automata, each component or subsystem is described by a set of nodes representing the state of the system. The directed edges between nodes represent state transitions, and the marks on the directed edges represent the name of the event that triggers the transition. The state in the timed automata can express either an instantaneous state or a time period. Researchers have developed and intensively studied methods of using timed automata to check the safety and progress properties of the system. There are many analysis and verification tools that support timed automata, such as UPPAAL [24], Kronos [25], and schedulability analyzer TIMES [26].

Definition 1. A timed automata A is defined as $A = (L, Init, F, \Sigma, C, Inv, T)$, where

(i) L is a finite state set

(ii) $Init \in L$ represents the initial state

(iii) F is the terminal state set

(iv) Σ is a set of actions/events

(v) C is a limited set of clocks

(vi) Inv defines a set of local invariants; it will give constraints on the states

(vii) T represents a set of transition relationships, expressed as $L \times \Sigma \times 2^C \times \Phi(C) \times L$, where Φ(C) is a set of clock constraints, defined by the following syntax (x is a clock; c is a real number):

$$\varphi := \text{true} \mid x \le c \mid c \le x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2$$

3.2.2. Timed Automata Pattern. The timed automata pattern [4, 27, 28] is a set of reusable and composable formal models presented by Dong. Its goal is to use the composition of these simple models to model and verify real-time systems efficiently. These patterns, such as delay, timed interrupt, deadline, and timeout, are abstractions of common concepts and phenomena in concurrent and distributed real-time systems. In the timed automata patterns, a timed automaton is abstracted as a triangle. The left vertex or the circle attached to the left vertex represents the initial state of the timed automata; the right side represents its final state. Figure 1 shows the Deadline timed automata pattern.

We classify the timed automata patterns proposed by Dong and divide them into vertical refinement patterns and horizontal expansion patterns. The principle of distinguishing these two patterns is whether there are more than one timed automaton in a pattern. Those patterns that only add new states to a single timed automaton are classified as vertical refinement patterns, while those representing the relationship between multiple timed automata (i.e., there are more than one timed automaton in a pattern) are classified as a horizontal expansion pattern. In addition, we also proposed three new vertical refinement patterns to express the concept that the atomic state in the abstract timed automata is further decomposed into multiple time periods. Therefore, based on this work, we divided vertical refinement patterns into two categories further: the first

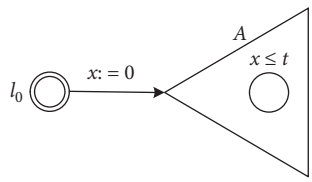

Figure 1: Deadline timed automata pattern.

category is called incremental refinement patterns, which are those that add new states or transitions to a timed automaton. The second type is called decomposition refinement patterns, which are those decomposing a certain atomic state in a timed automaton into multiple new atomic states. The classification and summary of timed automata patterns are shown in Table 1, where the three newly proposed decomposition patterns are marked in italics.

We use A or A_i to represent a single timed automaton in all the timed automata pattern definitions. Due to space limitations, we only use three typical examples, namely, the Deadline pattern, Sequence pattern, and Event Interrupt pattern to illustrate these three types of refinement patterns, respectively.

3.2.3. Vertical Refinement Patterns

Example of Incremental Refinement Patterns: Deadline Pattern. The Deadline pattern is used to model the deadline concept of real-time systems, as shown in Figure 1. The execution of timed automata A must be completed within t time units. Therefore, all states in A are marked with the invariant $x \le t$ (where x is a clock variable). When the system enters automata A, the clock x is reset to 0. The formal definition of the Deadline pattern is as follows.

Definition 2. Deadline $(A, t) \stackrel{def}{=} (L, Init, F, \Sigma, C, Inv, T)$, where
$L \triangleq L_A \cup \{l_0\}$, $Init \triangleq l_0$, $F \triangleq F_A$, $\Sigma \triangleq \Sigma_A$, $C \triangleq C_A \cup \{x\}$,
$Inv \triangleq \{(l_0, \text{urgent})\} \cup \{(l, inv) | l \in L \wedge inv = (Inv\ (l) \wedge x \le t)\}$,
$T \triangleq T_A \cup \{(l_0, \tau, \{x\}, true, l_1)\}$

Example of Decomposition Refinement Patterns: Sequence Decomposition Pattern. The Sequence decomposition pattern decomposes an atomic state in the timed automata A into multiple sequential substates. This pattern is used to decompose the abstract time interval into multiple subintervals, as shown in Figure 2. The dotted frame in the figure represents the atomic state node before decomposition, which is called the parent state; the state in the dotted frame represents the child state after decomposition. Each child state must complete its execution within a limited time, and the sum of the execution time of all child states cannot exceed the time limit of its parent state. The formal definition of the Sequence pattern is as follows.

Definition 3. Seq $(A, t, t_3, t_4, t_5) \stackrel{def}{=} (L, Init, F, \Sigma, C, Inv, T)$, where
$L \triangleq L_A \cup \{l_3, l_4, l_5\} \backslash \{l\}$, $Init \triangleq l_1$, $F \triangleq F_A$, $\Sigma \triangleq \Sigma_A \cup \{a_2, a_3\}$, $C \triangleq C_A \cup \{x_3, x_4, x_5\}$,
$Inv \triangleq \{(l_3, x_3 \le t_3), (l_4, x_4 \le t_4), (l_5, x_5 \le t_5)\} \cup Inv_a$.
$T \triangleq T_A \cup \{(l_2, a_1, \{x_3\}, \text{true}, l_3)\} \cup \{(l_3, a_2, \{x_4\}, x_3 > t_3., l_4)\}$
$\cup \{(l_4, a_3, \{x_5\}, x_4 > t_4, l_5)\} \cup \{(l_5, a_4, \varnothing, x_5 > t_5, l_6)\} \backslash \{(l_2, a_1, \{x\}, true, l)\}$
$\backslash \{(l, a_4, \varnothing, x > t, l_6)\}$

3.2.4. Horizontal Expansion Pattern

Event Interrupt Pattern. Event Interrupt pattern is used to model event-based interrupts, as shown in Figure 3. When an event a occurs, no matter what state the timed automaton A_1 is in, it will transfer to the timed automata A_2 for execution. The formal definition of the Event Interrupt pattern is as follows.

Definition 4. Interrupt $(A_1, A_2, a) \stackrel{def}{=} (L, Init, F, \Sigma, C, Inv, T)$, where
$L \triangleq L_1 \cup L_2$, $Init \triangleq l_1$, $F \triangleq F_1 \cup F_2$, $\Sigma \triangleq \Sigma_1 \cup \Sigma_2 \cup \{a\}$, $C \triangleq C_1 \cup C_2$,
$Inv \triangleq Inv_1 \cup Inv_2$,
$T \triangleq T_1 \cup T_2 \cup \{(l, a, \{x\}, true, l_2) | l \in L_1\}$

4. Methodology

4.1. Overview of Event-B's Time Refinement Framework. In this section, we used the iUML-B state machine to express the elapse of time, the decomposition of time intervals, and the behavior constraints of multiple concurrent real-time objects. We divide the control flow models of real-time systems into two categories according to the methods in the theory of timed automata:

(i) Functional model: This is the "untimed" control flow model without adding time constraints. It should be emphasized that we stipulate that each functional state machine describes the state transition of a single variable. This idea comes from the "atomic state machine" proposed in our previous work [29, 30].

(ii) Time elapse model: In a real-time system, the elapse of time is also an important factor that affects the sequence of events in the system, so it is necessary to model the elapse of time specifically.

We use the "functional state machine" and "ticker state machine" to express the two abovementioned models, respectively.

(1) Functional state machine: It is used to express the "untimed" behavior of the system. This approach is often used in the study of timed automata. It only expresses the sequence of events in the system without time constraints, which is the control flow without time constraints.

We used the "linked" capability of the iUML-B state machine of constructing the functional state machine; that is, the transition edges in the iUML-B state machine can be "linked" with the existing events in the Event-B model and automatically

Table 1: Classification of timed automata patterns.

Vertical refinement patterns		Horizontal expansion pattern
Incremental refinement pattern	Decomposition refinement pattern	
Event prefix	*Sequence*	Choice
Recursion	*Xor*	Sequential
Wait	*Loop*	Timeout
Deadline	—	Time interrupt
Wait until	—	Event interrupt
Timed-event prefix	—	Parallel composition
Periodic-repeat	—	—

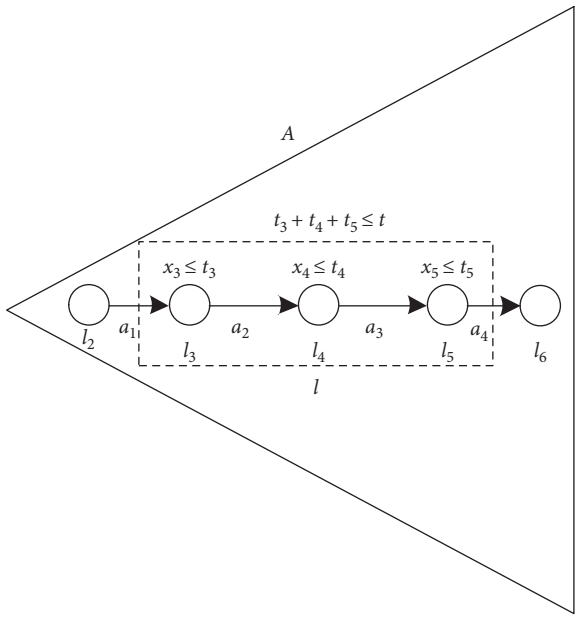

Figure 2: Sequence timed automata pattern.

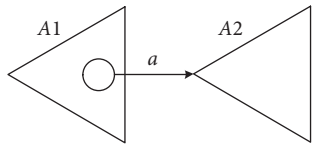

Figure 3: Event Interrupt timed automata pattern.

embed the generated code into the context (set, constant, and axiom) and machine (variables, events, and invariants) to control the order of these events. For example, suppose that we have already written five events: "INITIALISATION," "e_1," "e_2," "e_3," and "e_4." If we want to control the order of these events like this,

INITIALISATION- > e_1- > e_2- > e_3- > e_4,

where "->" means that the event on its left must occur before the event on its right, then we can create an iUML-B state machine and "link" these five events to the transition edges of the state machine, as shown in Figure 4. The italic codes in Figure 4 are the codes automatically generated by the iUML-B state machine to control the event order.

(2) Ticker state machine: It is used to express the elapse of time, including elapse of the global clock as well as the increment and reset of the local clock. An example of the ticker state machine is shown in the upper left corner of Figure 5. It has only one event *tick*, which is used to increase the value of global clock variables and local clock variables.

In this way, we can "mimic" the timed automata with a composition of a functional state machine and a ticker state machine. Suppose that there is a ticker state machine *TickerStm* and a functional state machine *FunctionStm* and "⊗" is used to represent the composition of state machines; then *TickerStm*⊗*FunctionStm* can be used to represent the Event-B code generated by these two state machines. Using the composition of a functional state machine and a ticker state machine, we can get an equivalent model similar to a timed automaton in Event-B (we named it "atomic time state machine," although it is not visible), as shown in Figure 5.

We use Figure 6 to explain in detail the construction process of the functional state machine and the clock state machine. In order to express the meaning of our method clearly, in the several large figures in this section, we just use a solid circular symbol with numbers or letters to express the state node of the iUML-B state machine and use a dotted rounded rectangle to express superstate of the iUML-B state machine, that is, those states that contain substates. For example, the state "0" and state "1" in TAtom_{0_0} in Figure 6 actually represent the two nodes of the iUML-B state machine. Similarly, the dashed rounded rectangle in TAtom_{0_3} in Figure 7 actually represents a superstate of the iUML-B state machine, which contains two substates D1 and D2. In the Nth layer model M_N, the function state machine TAtom_{0_0} is first used to describe the system's event sequence, and the Event-B code is automatically generated. At this time, the functional state machine model has no time constraints and time-related invariants. It is a completely "untimed" state machine, and the M_N model does not have the constraints of time. At the $N+1$-th layer, the clock state machine Ticker_{0_1} is added. This state machine will execute an event cyclically forever after initialization, that is, the tick_tock event. The effect of the clock tick_tock event is to add a minimum time unit to the values of all global clock variables and local clock variables. The code generated by Ticker_{0_1} is marked in italics at the bottom right of the M_{N+1} layer model. In order to turn the functional state machine into a time state machine, the Nth functional state machine

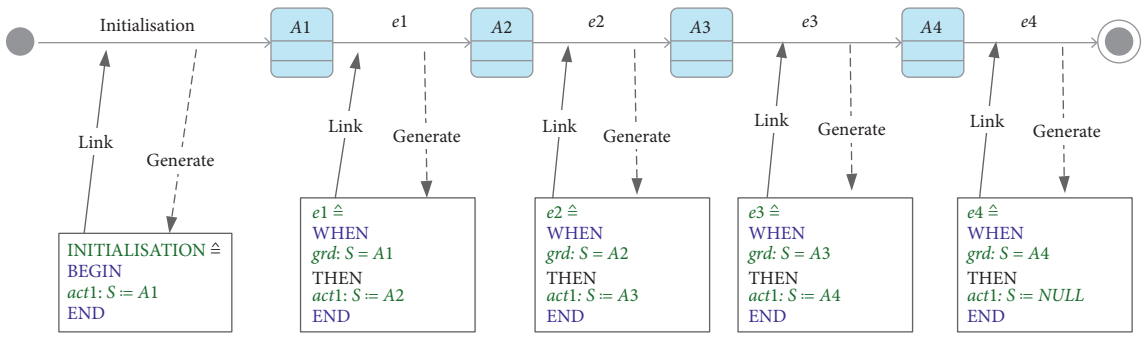

Figure 4: The construction of functional state machine.

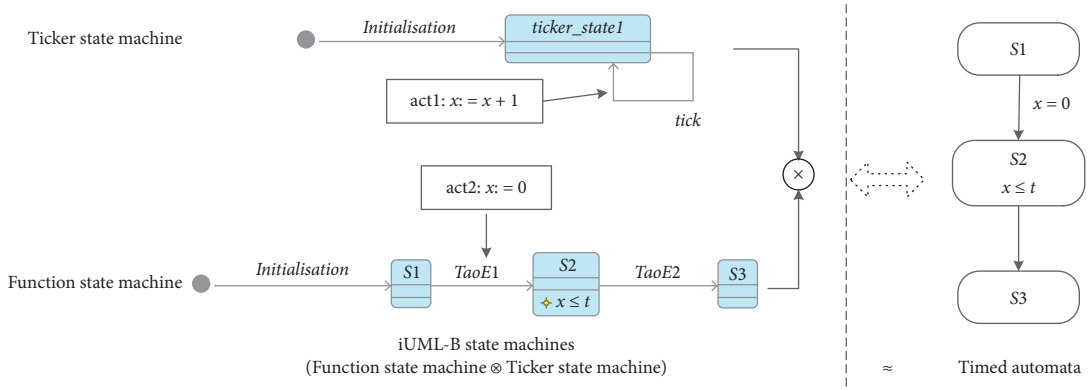

Figure 5: Composition of functional state machine and ticker state machine.

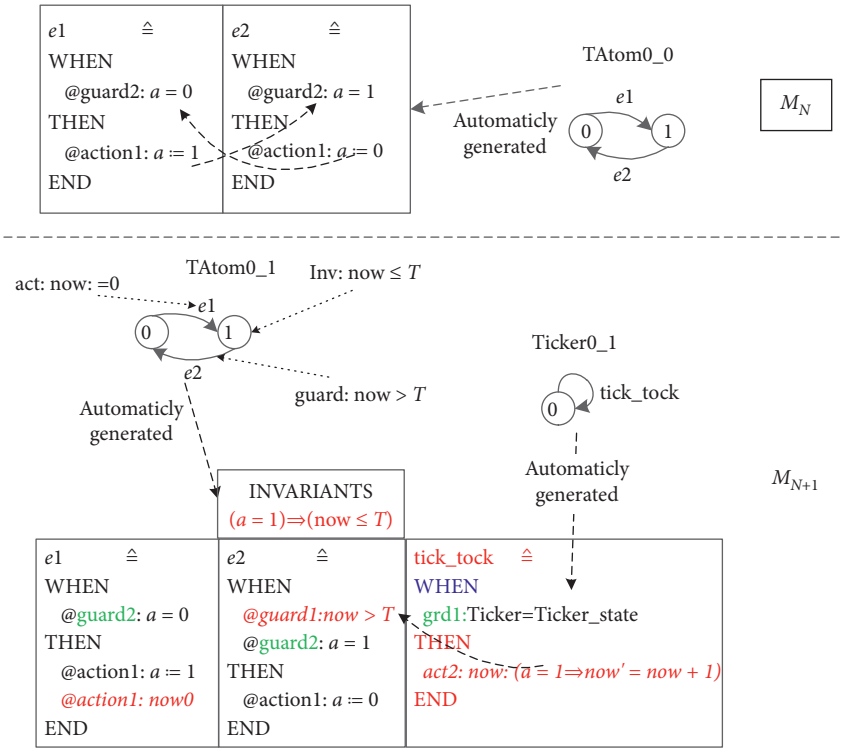

Figure 6: Details of functional state machine and clock state machine.

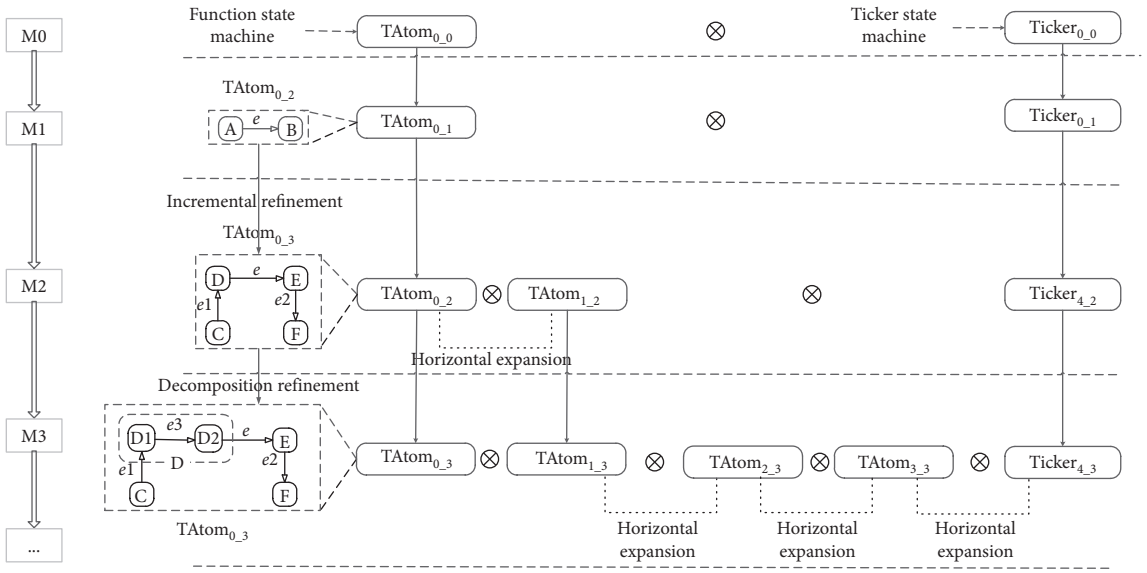

Figure 7: Time refinement framework based on functional state machine and ticker state machine.

$TAtom_{0\_0}$ is refined into the $N+1$ layer $TAtom_{0\_1}$. The refined method is as follows: first, a local clock variable (for example: now) is added, and a clock reset action (such as now: = 0) is added to the input edge of the state machine node that needs to add time constraints; secondly, an invariant is added to the node (such as state node $a = 1$) that needs to add a time constraint; for example,

Inv: now $\leq T$

where T is a constant of natural number type.

Then, a guard is added on the transition edge where time constraints need to be added; for example,

guard: now $> T$

Finally, an action is added to the tick_tock event:

act: now:$|(a = 1 \Rightarrow \text{now}' = \text{now} + 1)$

which can ensure that only when a is in the state $a = 1$, the value of the clock variable will (and must) increase a minimum time unit.

Next, we can use the functional state machine and the ticker state machine to gradually build an Event-B model of a real-time system, which is used as a time refinement framework as shown in Figure 7.

In this time refinement framework, the system model starts from the initial abstract model M0. The M0 layer has the simplest functional state machine $TAtom_{0\_0}$ and ticker state machine $Ticker_{0\_0}$. Then, in the M1 layer, we add a new state or transition (incremental vertical refinement) to $TAtom_{0\_0}$, making it $TAtom_{0\_1}$. Of course, we may also add a new clock variable to the ticker state machine $Ticker_{0\_0}$ to make it become $TAtom_{0\_1}$. At the M2 layer, we continue to perform incremental vertical refinement of $TAtom_{0\_1}$ and add a new functional state machine $TAtom_{1\_2}$; then we add the constraint relationship between $TAtom_{0\_2}$ and $TAtom_{1\_2}$ to perform a horizontal expansion. Through continuous use of this method, we can not only refine a single real-time object model but also continuously add new real-time objects to the system model, until the final real-time system model is obtained.

The rest of this section shows how to implement the various refinement patterns in Table 1 using iUML-B state machine. Due to space limitations, we only use three examples to illustrate the construction method and process.

4.2. Incremental Refinement

4.2.1. General Construction Method. We use the construction process of the Deadline timed automata pattern to illustrate the principle of incremental refinement, as shown in Figure 8. The principle of the Deadline pattern is to constrain all events in a state machine to be completed before a certain deadline. Therefore, we follow the steps below to complete the refinement process:

(1) In untimed refinement, the Nth layer functional state machine $TAtom_{0\_1}$ is refined into the $N+1$-th layer functional state machine $TAtom_{0\_2}$. A new node is added at the $N+1$-th layer and marked as state 2. The newly added node is then changed into a new initial state node, and a new event e_3 is added that connected the new node with the original initial state of $TAtom_{0\_1}$ $(a = 0)$.

(2) A new clock variable x and a new clock constant T_1 are added in the $TAtom_{0\_2}$ machine.

(3) The following operations are performed on $TAtom_{0\_2}$:

(i) An action is added to event e_3:

act: $x := 0$

(ii) An invariant is added on all nodes of the original state machine that need to add deadline constraints:

Inv: $x \leq T_1$

(4) A new action is added to the tick_tock event of $Ticker_{0\_2}$, the clock state machine of the $N+1$ layer:

act_3: $x:|(a = 0 \Rightarrow x' = x + 1)$

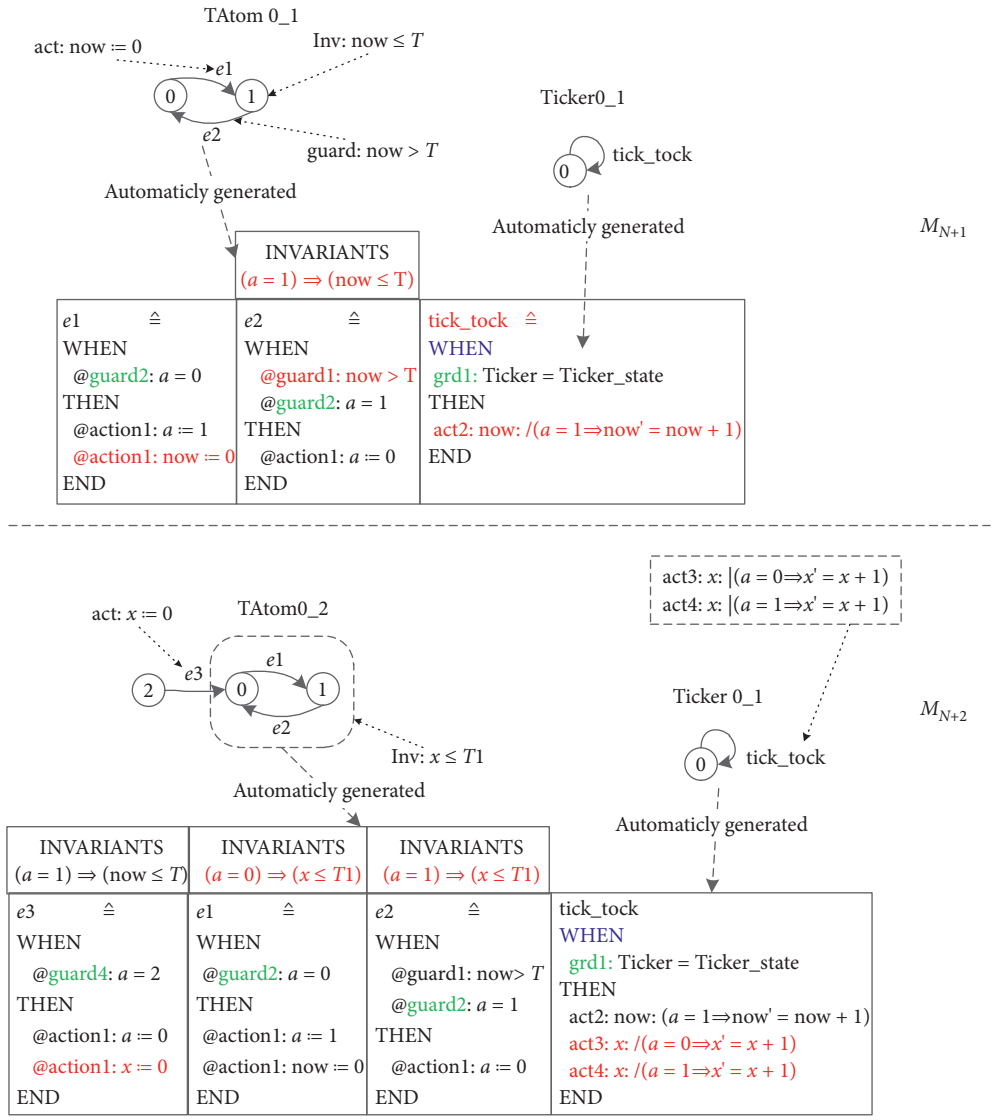

Figure 8: General method of incremental refinement.

$$\text{act}_4\text{: } x\text{:}|(a = 1 \Rightarrow x' = x + 1)$$

After the above steps, the requirements of the Deadline pattern can be achieved; that is, the behavior of the TAtom_{0_1} state machine is limited to be completed within T_1 time units.

4.2.2. Performing Incremental Refinement Using iUML-B State Machine. We use iUML-B state machine to perform various incremental refinements under the guidance of general methods. Only an example of Deadline pattern is given here. In Figure 9 and the rest of this paper, we use the superstate of the iUML-B state machine to represent the atomic time state machine before refinement. For example, in Figure 9, B1 is used to represent the atomic time state machine before refinement. If a pattern includes two iUML-B time state machines, we will use B1 and B2 to represent them, respectively.

According to the definition of the Deadline pattern in Figure 1 and Definition 3.2, the state node B1 and the initial state B0 are constructed to simulate the l_0 node and l_1 node in Figure 1, and the system will enter the B0 state after initialization. Since B0 (that is, l_0) is in an emergency state, a conditional constraint is added to the ticker state machine so that the clock x will not start until it enters the B1 state. In addition, the invariant $x \le t$ is added to the state of B1 to constrain it to be completed within t time units. This also means that all transitions in B1 must be completed within t time units. When the clock x reaches t, the time will be "stopped," and the tick_tock event will be disabled.

4.3. Decomposition Refinement. What needs to be emphasized is that, according to the definition in UML, state decomposition and composition can be performed based on the UML state machine diagram itself, but, as we said in

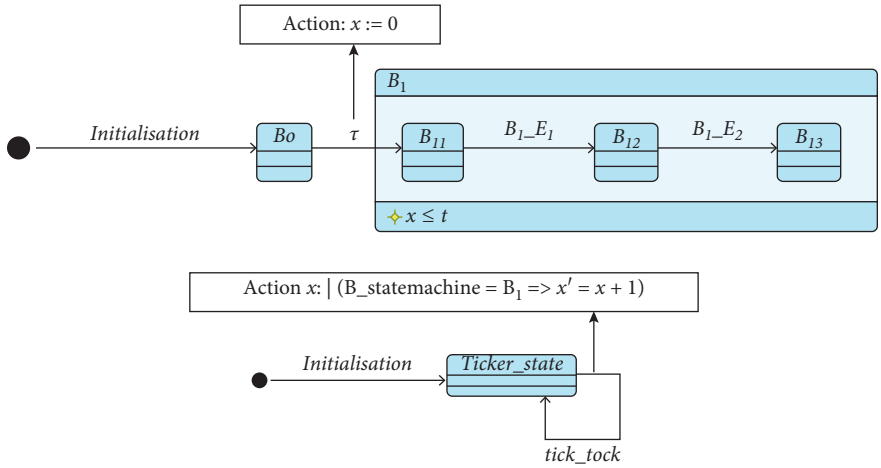

Figure 9: Performing incremental refinement using iUML-B state machine.

Section 3.1.2, the refinement of the iUML-B state machine is not as informal as the refinement of UML's state machine. In short, first of all, the iUML-B state machine is just the front end of Event-B. Therefore, its decomposition and refinement are based on the decomposition and refinement of Event B. Secondly, the refinement of the iUML-B state machine must follow more rules to ensure that the refinement is correct. Third, compared with the state refinement of iUML-B, the state refinement of UML does not have any formal semantic guarantee. Regarding the difference between the two modeling languages, Said discussed this in detail in her PhD thesis.

4.3.1. General Construction Method. The principle of time interval decomposition refinement is to decompose a time interval in the abstract model into multiple time intervals that will be executed sequentially (or selection, cycle) in the refined model. These intervals are named Subinterval, the shorthand of subinterval. We use sequential decomposition to clarify the principle of time interval decomposition refinement, as shown in Figure 10.

The Nth layer model in Figure 10 is refined into the $N+1$-th layer model. Its main effect is to decompose the time-constrained node $a=1$ in the Nth layer model into three subintervals in the $N+1$-th layer model, namely, SubInter1, SubInter2, and SubInter3. The detailed refinement process is as follows.

(1) In untimed refinement, the Nth layer functional state machine TAtom_{0_1} is refined into the $N+1$-th layer functional state machine TAtom_{0_2}. The state machine node 1 of TAtom_{0_1} is decomposed into three nodes of TAtom_{0_2}, which are marked as states 1, 2, and 3. Events e_3 and e_4 are added to connect these nodes.

(2) Clock variables x_1, x_2, and x_3 as well as clock constants t_1, t_2, and t_3 are added to the $N+1$ layer model.

(3) The following operations are performed on TAtom_{0_2}:

(i) Action "act: $x_1 := 0$" is added on the transition edge of event e_1, and invariant $x_1 \le t_1$is added on e_1's target node (node 1);

(ii) A guard grd: $x_1 > t_1$ and an action "act: $x_2 := 0$" are added on the transition edge of event e_3, and invariant $x_2 \le t_2$is added on e_3's target node (node 2);

(iii) A guard grd: $x_2 > t_2$ and an action "act: $x_3 := 0$" are added on the transition edge of event e_4, and invariant $x_3 \le t_3$ is added on e_4's target node (node 3);

(iv) A guard grd: $x_3 > t_3$is added to the transition of event e_2 .

(4) The ticker state machine Ticker_{0_1} is refined into Ticker_{0_2}. The detailed refinement process is that, for each node with clock constraints if the current system state stays at the node, the clock variable corresponding to the node is increased by a minimum time unit. For example, for node $a=1$, just add action: $(a=1 \Rightarrow x_1' = x_1 + 1)$.

(5) A global invariants is added: *Inv*: $t_1 + t_2 + t_3 \le T$.

In the above process, step (1) is untimed refinement and does not involve any time information; the other steps are time refinement. These refinement steps can all be completed in the iUML-B state machine, and the Event-B code shown in Figure 10 is automatically generated. Codes related to time control flow are marked in italics.

4.3.2. Performing the Decomposition Refinements Using the iUML-B State Machine. We use the Sequence refinement pattern to illustrate how to use the iUML-B state machine to perform the decomposition refinement. In the Sequence refinement pattern, one part is the decomposition of the state. The original state l is decomposed into three substates l_3, l_4, and l_5. The new states are connected by the added events a_2 and a_3. The other part is the decomposition of time. Each substate has its own local clock. The prerequisite for state transition is to satisfy the constraints of the local invariant of the state, as shown in Figure 11.

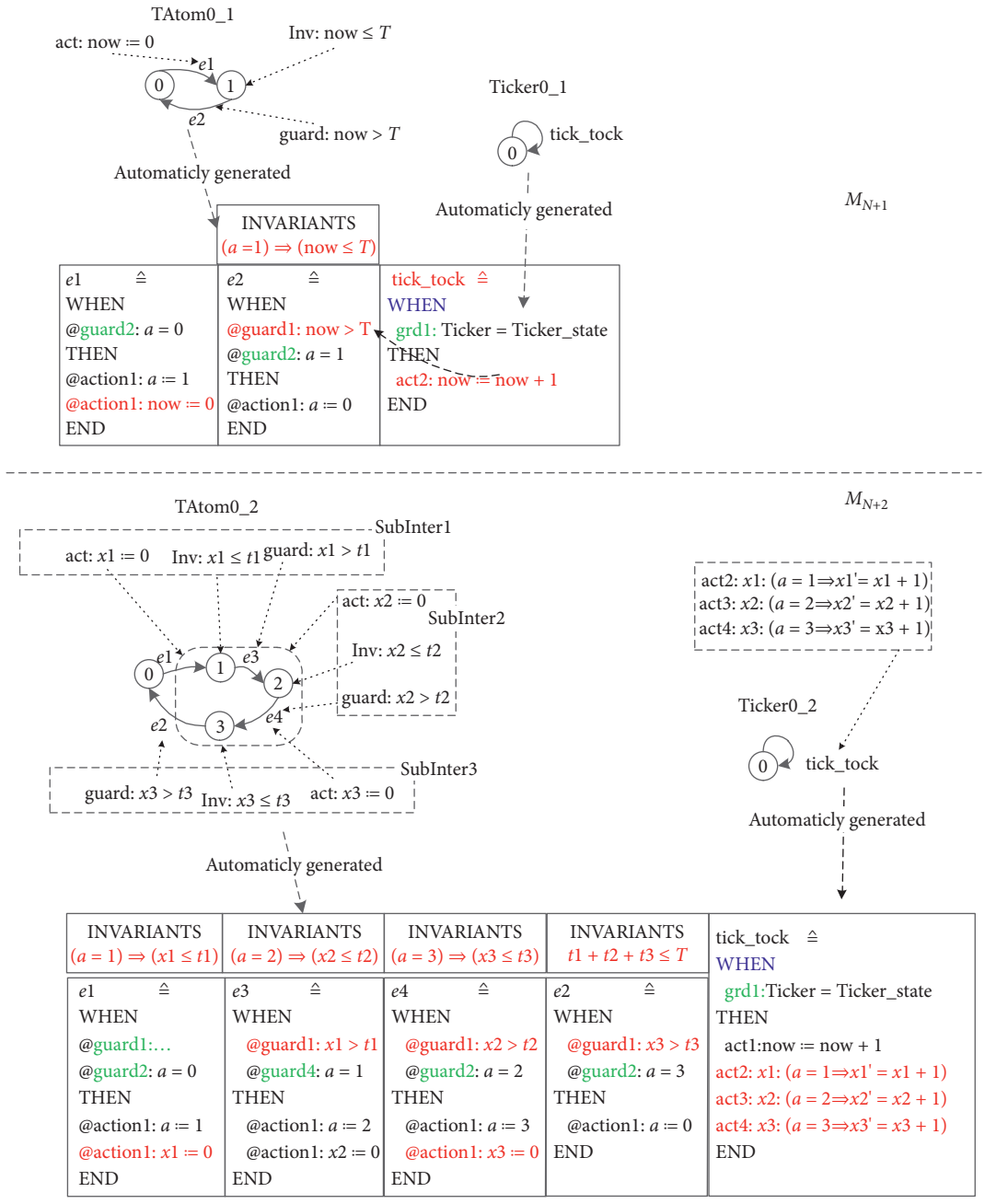

Figure 10: General method of decomposition refinement.

The invariant on the l_3 state means that when the system is in this state, the value of the clock x_3 is not greater than t_3, so the following code is added to the tick_tock event in the Ticker state machine:

Guard: l_statemachine = $l_3 \Rightarrow x_3 + 1 \leq t$

Similar invariants have been added to the l_4 and l_5 states. It should be noted that since the three substates are decomposed from the original state, the maximum time they consume cannot exceed the upper limit of the time specified by the original state. In other words, the following global invariants must be added:

Invariant: $t3+t4+t5 \leq t$

4.4. Horizontal Expansion

4.4.1. General Construction Method. In Dong's work, the horizontal expansion pattern is mainly used to describe the relationship between two real-time objects. The refined framework proposed in this chapter intends to gradually add new concurrent objects while imposing constraints on the relationship between new objects and existing objects. We model a total of 6 horizontally expansion patterns using the iUML-B state machine. In this section, only the construction process of the Event Interrupt pattern is presented to clarify the principle of horizontal expansion, as shown in Figure 12.

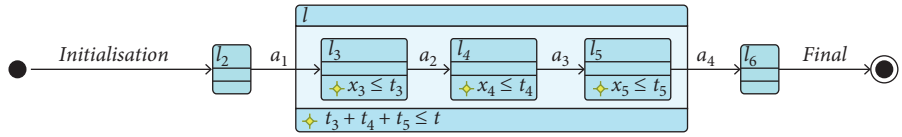

Figure 11: Performing decomposition refinement using iUML-B state machine.

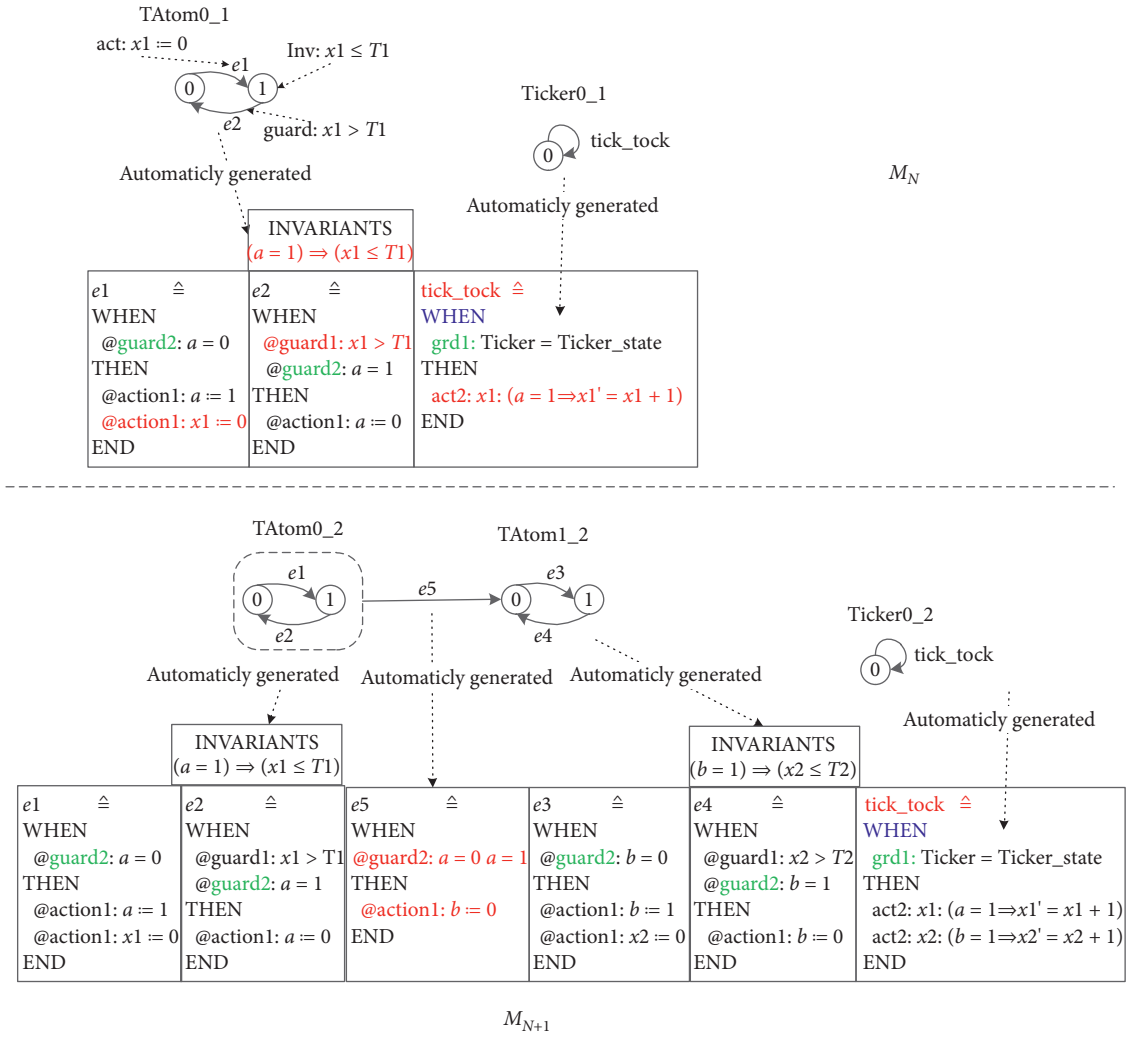

Figure 12: General method of horizontal expansion.

The principle of Event Interrupt pattern is that when an external event occurs, no matter what state a state machine is in, it must immediately switch to the initial state of another state machine and start execution. The premise is that there is already a state machine whose clock variable is x_1. The resulting model of this expansion is a new model of the composited system of two state machines. Therefore, we perform the following refined process:

(1) A new state machine TAtom_{1_2} is added at the $N+1$-th layer, whose state variable name is different from TAtom_{0_1}. For example, if the state variable name of TAtom_{0_1} is a, the state variable name of TAtom_{1_2} must not be a. It is marked as b in Figure 12.

(2) At the $N+1$-th layer, TAtom_{0_2} inherits all the states of TAtom_{0_1}, and a transition edge marked e_5 from all states of TAtom_{0_2} to TAtom_{1_2} is added. It should be noted that this step only needs to add a transition edge from the superstate of TAtom_{0_2} to the initial state of TAtom_{0_1} in the Rodin tool.

(3) A new clock variable x_2 and a new clock constant T_2 are added.

(4) A new action is added to the tick_tock event of the clock state machine (Ticker_{0_2}) of the $N+1$ layer:

$$\text{act}_2\colon x_2 :\mid (b = 1 \Rightarrow x_2' = x_2 + 1)$$

4.4.2. Performing Horizontal Expansion Using the iUML-B State Machine. We use the iUML-B state machine to perform the event interrupt horizontal expansion, as shown in Figure 13.

5. Case Study

In this section, we use the time refinement framework proposed in this paper to construct the Event-B model of a typical real-time system-pacemaker control system.

5.1. Principles of the Pacemaker System. A pacemaker is a medical device that regulates the heart rate. It can sense the activity of the heart and use electrical impulses to control the myocardium so that the heart beats at a prescribed heart rate. Pacemakers usually use a two-wire device to sense and control the myocardium of the right atrium and right ventricle, as shown in Figure 14.

The core idea of the pacemaker is shown in Figure 15. If the contraction activity of the atrium or ventricle is not sensed within a specified period of time, an electrical pulse is sent to stimulate the atrium or ventricle's myocardium so that they can beat at a specified heart rate. For example, on the atrium line in Figure 15, the pacemaker did not sense the arrival of the next same event within the specified interval after the first atrial beat event, so it sent a pacing signal (pace).

In this paper, the events of sensing heart activity are collectively referred to as sensing events and the pacing actions on the heart are collectively referred to as pacing events, such as ventricular sensing events or atrial pacing events. Atrial sensing and atrial pacing events are collectively referred to as atrial events, and the definition of ventricular events is similar. In the functional model of Figure 15, the entire timeline is composed of a series of cardiac cycles. The cardiac cycle is interpreted as the interval between two consecutive ventricular events. $A_1 \sim A_n$ specify some time intervals after the occurrence of atrial events, and $V_1 \sim V_k$ are the time intervals after the occurrence of ventricular events. When the system is in a certain time interval, we say that this interval is "active." For example, after the atrial sensing event in Figure 15, $A_1 \sim A_n$ are active, and, after the first ventricular sensing activity, $A_1 \sim A_n$ become inactive, and $V_1 \sim V_k$ become active. Table 2 gives the professional terms in the field of pacemakers.

The meaning of various time intervals is as follows:

(i) LRI: it is the longest time interval allowed between two consecutive ventricular activities. If no ventricular activity occurs during this interval, the pacemaker must issue a ventricular pacing event (VP) at the end of LRI. The ventricular sensing event (VS) can stop and reset the LRI interval. That is, after a ventricular sensing event (VS) or a ventricular pacing event (VP) occurs, the LRI interval will restart.

(ii) URI: the URI interval is triggered by any ventricular activity (VS or VP). When the URI is active, the pacemaker cannot send out a ventricular pacing event; that is, it cannot trigger a VP event.

(iii) AVI: it is designed to maintain atrial-ventricular synchronization. AVI is the interval between consecutive atrial and ventricular events. The AVI interval is triggered by an atrial activity (AS or AP). If no ventricular activity is sensed during the AVI interval, a ventricular pacing event is issued after the interval ends. When AVI is active, the interval will not be reset by any atrial activity (AS or AP) and can only be interrupted by ventricular activity (VS or VP). It should be noted that the duration of the AVI interval is shorter than the LRI interval.

(iv) VRP: it is defined as the interval where the ventricular channel is insensitive to any input signal. The VRP interval is triggered by VS or VP and can be stopped by inner atrial activity. In addition, the VRP interval is shorter than the LRI interval.

(v) PVARP: the PVARP interval is to prevent the atrial channel from sensing ventricular pacing events due to crosstalk; therefore it is not allowed to start a new AVI interval during PVARP. PAVRP is triggered by any ventricular activity (VS or VP) and can be stopped at any time by ventricular sensing activity (VS). Similarly, the duration of the PVARP interval is shorter than the LRI interval. In order to prevent the crosstalk of ventricular activity, the duration of the PVARP interval must always be longer than the VRP interval.

(vi) VABP: in order to prevent ventricular activity from causing unwanted crosstalk behavior on the atrial channel and filter potential noise, there is a blanking period (VABP) after each ventricular event (VS, VP). During the VABP interval, atrial events are ignored. VABP is triggered by ventricular pacing activity (VP) and overlaps with PVARP, so the duration of VABP must be shorter than the duration of PVARP.

(vii) VAI: it is also called the atrial escape interval. This interval represents the greatest possible delay in the occurrence of atrial activity after the ventricular activity. VAI is triggered by arbitrary ventricular activity. If the VAI interval ends, the pacemaker must provide an atrial pacing event; that is, an AP event must be triggered. If atrial activity is detected while the VAI interval is active, the VAI interval will be interrupted. The sum of the duration of the VAI interval and the duration of the AVI is equal to the duration of the LRI interval. The VAI interval and AVI interval do not overlap.

The relationship between events and various time intervals in Table 2 is shown in Figure 16. The events and time intervals are arranged from left to right in the order of their occurrence. An AS event initiates a new AVI interval and a subsequent VP event terminates it and starts a new cardiac

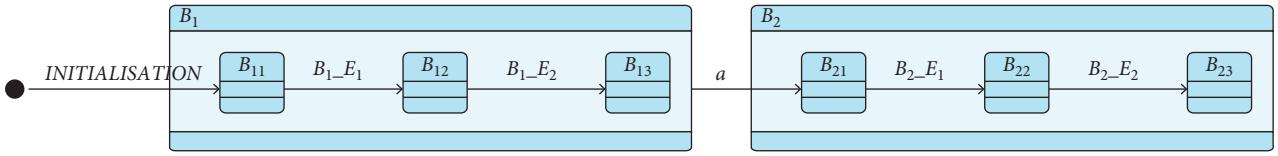

Figure 13: Performing horizontal expansion using iUML-B state machine.

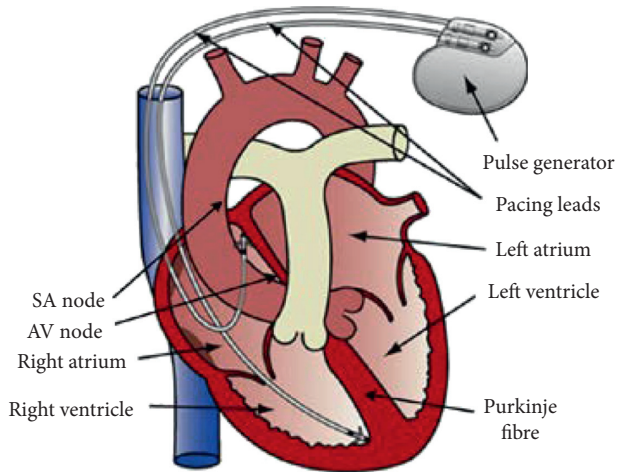

Figure 14: The principle of pacemaker.

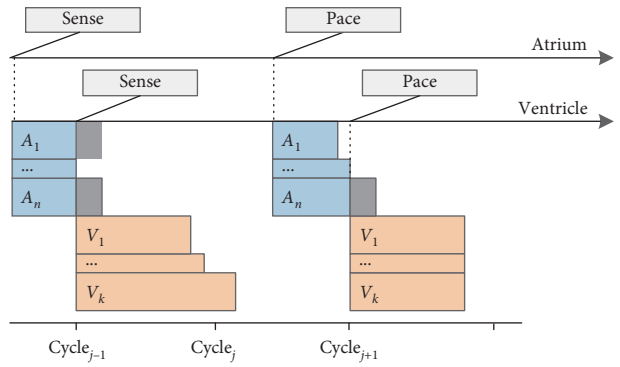

Figure 15: The core idea of pacemaker.

Table 2: Professional terms of pacemaker.

Short term	Meaning	Short term	Meaning
AS	Atrial sensing event	LRI	Lower rate interval
AP	Atrial pacing event	URI	Upper rate interval
AX	Atrial event	VAI	Ventricular-atrial interval
VS	Ventricular sensing event	AVI	Atrial-ventricular interval
VP	Ventricular pacing event	VRP	Ventricular refractory period
VX	Ventricular event	VABP	Postventricular-atrial blanking period
		PVARP	Postventricular-atrial refractory period

cycle. The intervals that are simultaneously activated by the VP event within this cardiac cycle are PVARP, VRP, LRI, URI, and VAI. In the first cardiac cycle, due to the end of VAI, the pacemaker sent out an AP event; thus the AVI interval is started. After that, although the AVI interval is still in the active phase, AVI is terminated and the system enters the second cardiac cycle because the pacemaker senses a ventricular event VS. In this cardiac cycle, although both LRI and URI are active, they are interrupted by atrial perception events and the system enters the AVI interval

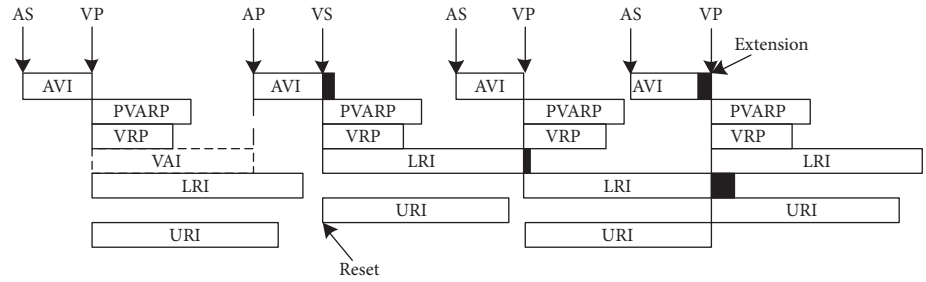

Figure 16: The trigger relationship between the pacemaker's events and time intervals.

again. Because this AVI interval started earlier, the LRI was still active when AVI ended, but, according to requirements, there must be a VP event to issue ventricular pacing after the AVI ends. Therefore, the LRI interval is terminated and the system enters the next cardiac cycle. In the third cardiac cycle triggered by the new VP event, the AVI interval arrives so earlier that its end time is even earlier than the URI. But because the URI is the minimum constraint value for a cardiac cycle, the system must wait for the ends of the URI interval before sending a VP event.

The principle of cardiac pacemaker is very complicated, which is explained in detail in the literature [31]. We will not repeat it here but only give the modeling requirements of each event and time interval as shown in Table 3.

5.2. Pacemaker Modeling Based on iUML-B Time State Machine. We use the iUML-B state machine to establish the Event-B model of the pacemaker under the guidance of the framework proposed in Section 4. Our basic idea is to model all time intervals as state nodes of the iUML-B state machine. Each interval has its own clock variables, clock constants, and invariants. The clock variables of the interval are reset when the system is entering this interval. In an interval, time can elapse only when the clock variable is within the allowable range of the clock invariants of this interval. When the value of the clock variable exceeds the boundary of the clock invariant, an event that leaves that state is triggered. According to the relationship between the time intervals, we first model the longest time interval, that is, LRI and URI as the top-most time interval, which includes the rest of all intervals. Then, the time intervals that occur sequentially are modeled as a sequential decomposition pattern of an abstract model, and the time intervals that occur at the same time are modeled as a parallel composition pattern.

In order to show that our method can make it easier for modelers to convert the Event-B model to the timed automata model, we give the mapping rule from iUML-B state machine to UTA:

(i) The state node pointed to by the "initialization" transition edge of the iUML-B state machine is mapped to the initial state node of UTA.

(ii) The states of the iUML-B state machine without time constraints are mapped to the "Committed" nodes or "Urgent" nodes of UTA.

(iii) The states of the iUML-B state machine marked with the time constraint (i.e., an interval) are mapped to the nodes of UTA marked with the same time constraint, that is, those nodes that are neither "Committed" nor "Urgent;" the invariants of these states can be directly mapped to the invariant of the corresponding node of UTA.

(iv) The "guard" and "action" on the transition edge of the iUML-B state machine are mapped to the corresponding positions of the "guard" and "update" part of the UPPAAL timed automata, respectively.

(v) The pseudostate nodes of the iUML-B state machine (such as branch and aggregation nodes) can be mapped to the pseudostate nodes in the UPPAAL timed automata or to a "Committed" position or "Urgent" position of the UTA.

We use these rules to translate the Event-B model to the corresponding UTA model of each layer. The timed automata model is added to the overall system model step by step according to the stepwise refinement process of the Event-B model.

5.2.1. Abstract Model (Layers 0 Model). The LRI and URI intervals represent the maximum time interval allowed between two ventricular events and at least the minimum time interval to elapse, respectively. Through the analysis of Figure 16, it can be found that LRI and URI are usually triggered by the same event. As for the response event VP after the completion of these two intervals, the following facts are also true: first, VP must occur after the end of the URI; secondly, if the URI has ended but the LRI has not ended, there will be two situations: one is the end of the LRI that triggers the VP event and the second is that the AVI is ended before LRI is ended; then the VP event is triggered by the end of AVI. In either case, the clock variable of URI and LRI will be reset to zero. Moreover, before the next VP or VS event arrives, even if the URI has ended, it cannot be reset. Therefore, LRI and URI are modeled as the same time interval, as shown in Figure 17(a). We present the corresponding UPPAAL timed automata model in Figure 17(b). It should be noted that, in Figure 17 and the remained figures in this section, there is no complete semantic equivalence between the iUML-B state machine and the

Table 3: Modeling requirements for pacemakers.

	No. of requirements	Description of requirements
Environmental requirements	ENV_1	The pacemaker interacts with the heart only through pacing and sensing
Functional requirements	FUN_1	AP can occur only after VX, and VP can occur only after AX
LRI	TIME_1	The VP event is triggered at the end of the LRI
	TIME_2	VS can terminate LRI
	TIME_3	LRI is triggered by VX
VRP	TIME_4	When VRP is active, VX cannot occur
	TIME_5	VRP is triggered by VX
	TIME_7	VRP is shorter than LRI
AVI	TIME_8	AVI is triggered by AX
	TIME_9	VP event is triggered at the end of the AVI
	TIME_10	When AVI is active, AX cannot occur
	TIME_11	VS can terminate AVI
	TIME_12	AVI is shorter than LRI
PVARP	TIME_13	When PVARP is active, AX cannot occur
	TIME_14	PVARP is triggered by VX
	TIME_15	VS can terminate PVARP
	TIME_16	PVARP is shorter than VAI
URI	TIME_17	URI is reset by VX
	TIME_18	When URI is active, VP cannot occur
VAI	TIME_19	VAI is triggered by VX
	TIME_20	The sum of VAI and AVI equal to LRI
	TIME_21	AP is triggered at the end of the VAI
	TIME_22	AS and VS can terminate VAI
	TIME_23	VAI and AVI do not overlap
VABP	TIME_24	VABP is triggered by VP
	TIME_25	VABP starts by PVARP and overlaps with it
	TIME_26	VABP is shorter than PVARP

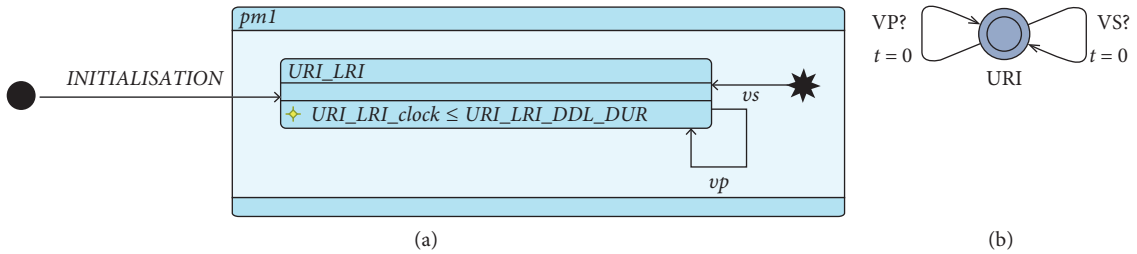

Figure 17: Abstract models URI_LRI and the corresponding timed automata. (a) URI_LRI state machine; (b) URI timed automata.

corresponding UPPAAL timed automata. This equivalence needs to be proved in subsequent work.

There are three trigger events in the URI_LRI interval: VS, VP, and *INITIALISATION*. *INITIALISATION* means that the time interval is activated after the model is initialized. At any time, the URI_LRI interval can be aborted by VS. Therefore, according to the Time-Event Prefix pattern, we construct an "any" state and let it connect to the URI_LRI state through the VS event. Then, we reset the clock URI_LRI_clock in the action of the VS event. In addition, the event VP can occur only after the time interval URI_LRI_DLY_DUR has ended and must occur before the end of the URI_LRI_DDL_DUR. According to the related definition of the Delay pattern, we add a guard to the VP event:

Guard: URI_LRI_clock ≥ URI_LRI_DLY_DUR

Then, we add a guard to the tick_tock event of the ticker state machine according to the definition of Deadline pattern:

Guard: URI_LRI_clock+1 ≤ URI_LRI_DDL_DUR

which will stop the clock when it reaches URI_LRI_DDL_DUR.

5.2.2. First Level Refined Model. In this level of refinement, we divide the URI_LRI interval into two subintervals, namely, the ventricular-atrial interval (VAI) and the ASensed interval which represents the time interval after receiving an atrial event. We have AVI ≤ ASensed and VAI + ASensed ≤ URI_LRI. These two intervals occur sequentially in the timeline. Therefore, we use the sequential

decomposition pattern to model the subintervals as separate states VAI and ASensed, as shown in Figure 18(a). Events AS and AP represent atrial sensing events and atrial pacing events, respectively. We add a UPPAAL timed automata model "VAI" in Figure 18(b), where the clock "t" is the local clock of VAI timed automata and the constant "TVAI" corresponds to the VAI_DDL_DUR constant in Figure 18(a). Unless otherwise specified, the remaining diagrams in this section will follow this rename rule. If "||" represents the composition of timed automata, then the current global timed automata model (named system) can be expressed as

System = URI||VAI

The ASensed interval follows the VAI interval and is triggered by the event AP or AS. The AS event is located at the end of the VAI interval and the beginning of the ASensed interval and can occur within the VAI_DDL_DUR time. The event AP, as the response of VAI's ending and the trigger of ASensed, must occur after VAI_DLY_DUR and before VAI_DDL_DUR. According to these requirements, we reset ASensed's local clock ASensed_clock when AS and AP events occur. In addition, we add a guard to AS events according to the definition of Deadline patterns:

Guard: VAI_clock + 1 ≤ VAI_DDL_DUR

and add guards to AP events according to the definition of Delay pattern and Deadline pattern:

Guard1: VAI_clock + 1 ≥ VAI_DLY_DUR

Guard2: VAI_clock + 1 ≤ VAI_DDL_DUR

5.2.3. Second Level Refined Model. In this level of refinement, we continue to refine the ASensed interval. According to the principle in Figure 16 in Section 5.1, atrial events AS and AP will trigger the AVI interval. Therefore, in this level of refinement, we connect the AS and AP events to the AVI interval. In addition, the end of the AVI interval can be divided into three situations: the first situation is that the AVI is interrupted by the VS event, and the system enters the VAI interval; the second situation is that the URI has already become inactive after the end of the AVI interval; then a VP event needs to be triggered to perform a ventricular pace; in the third situation, the URI is still active after the AVI interval ends, so the pacemaker needs to wait for the end of the URI to enter the next cardiac cycle. We use XOR pattern and sequential pattern to model these three situations, as shown in Figure 19(a).We add a UPPAAL timed automata model "AVI" in Figure 19(b), where the clock "clk" is the local clock of this timed automata and the clock "t" is the global clock of the overall system. Now, the current global timed automata model can be expressed as

System = URI||VAI||AVI

5.2.4. Third Level Refined Model. In the third level of refinement, we further refine the VAI interval. First, according to the principle described in Section 5.1, the VRP interval and PVARP interval overlap in the timeline, and their relationship conforms to the parallel composition pattern. Therefore, we decompose the VAI interval into a parallel composition of VRP and PVARP. Second, according to Jiang's work [32], we put VABP before PVARP (this will make our model and the timeline in Figure 16 inconsistent. Although this may seem strange, we think Jiang's analysis is more professional), as shown in Figure 20(a). We add two UPPAAL timed automata models "VABP" and "VRP" in Figure 20(b) and Figure 20(c). Now, the current global timed automata model can be expressed as

System = URI||VAI||AVI||VABP||VRP

We can continue to use the horizontal expansion patterns and vertical refinement patterns proposed in this paper to get a more complex iUML-B state machine model of the pacemaker. The case study in this section is just to prove the practicality of our proposed modeling framework and iUML-B time state machine pattern. In addition, as we have shown, the iUML-B time state machine can be easily converted to the UPPAAL timed automata. This is because we have used the iUML-B state machine to imitate UPPAAL timed automata from the beginning.

6. Discussions

In this section, we compared the existing real-time Event-B method with the method proposed in this paper. We gave six evaluation criteria, as shown in Table 4.

The principle of the "time constraint" pattern is shown in Figure 21(a). An event named tick_tock is used to progress the time, that is, to increment the value of the clock variable. An event named post_time is used to add one or more time constraints, and an event named process_time is used to delete one or more time constraints. Under the limitation of the "time constraint" pattern, if the clock variable is increased to the minimum time constraint point (e.g., at1 in Figure 21(a)), and the process_time event has not deleted this constraint, the system will "stop" the clock, which is what "constraint" means. Time constraint pattern is only a design method. It does not provide a mechanism to express the order of events as well as time duration. In other words, people can only manually add Event-B code in order to control the order of events and describe the time duration. It cannot express concurrent real-time objects, so a lot of manual codings are required when modeling a slightly more complex real-time system. In addition, it does not provide any plug-ins to automatically generate the proof obligations, nor does it support the refinement (including horizontal expansion and vertical refinement) process of the real-time system.

The principle of "duration pattern" is shown in Figure 21(b). It uses an event bt (which means "begin to true") to change a predicate P from FALSE to TRUE and at the same time reset the value of timer D to 0. An event named tic will increment the value of D only when P is TRUE. When the value of D reaches a certain constant c, another event is triggered. Except that a variable D is used to express the duration, the disadvantages of the "duration pattern" are the same as the time constraint pattern.

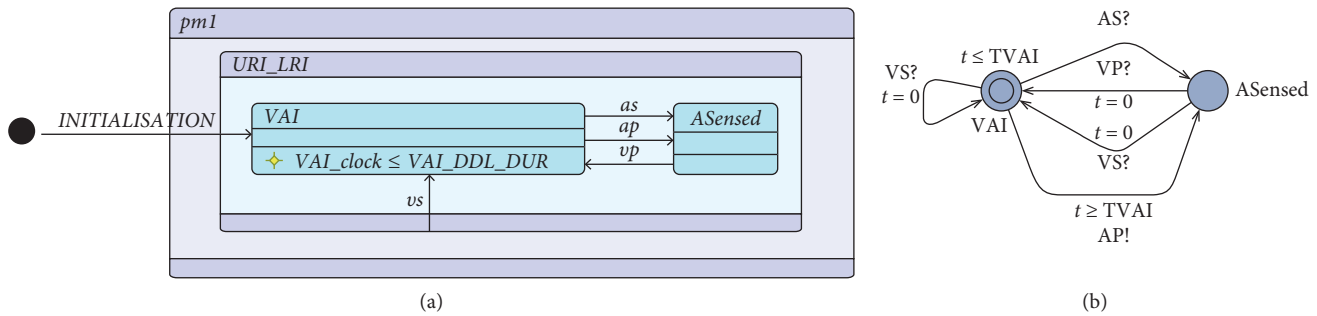

Figure 18: First level refined model and the corresponding timed automata. (a) First level refined state machine; (b) VAI timed automata.

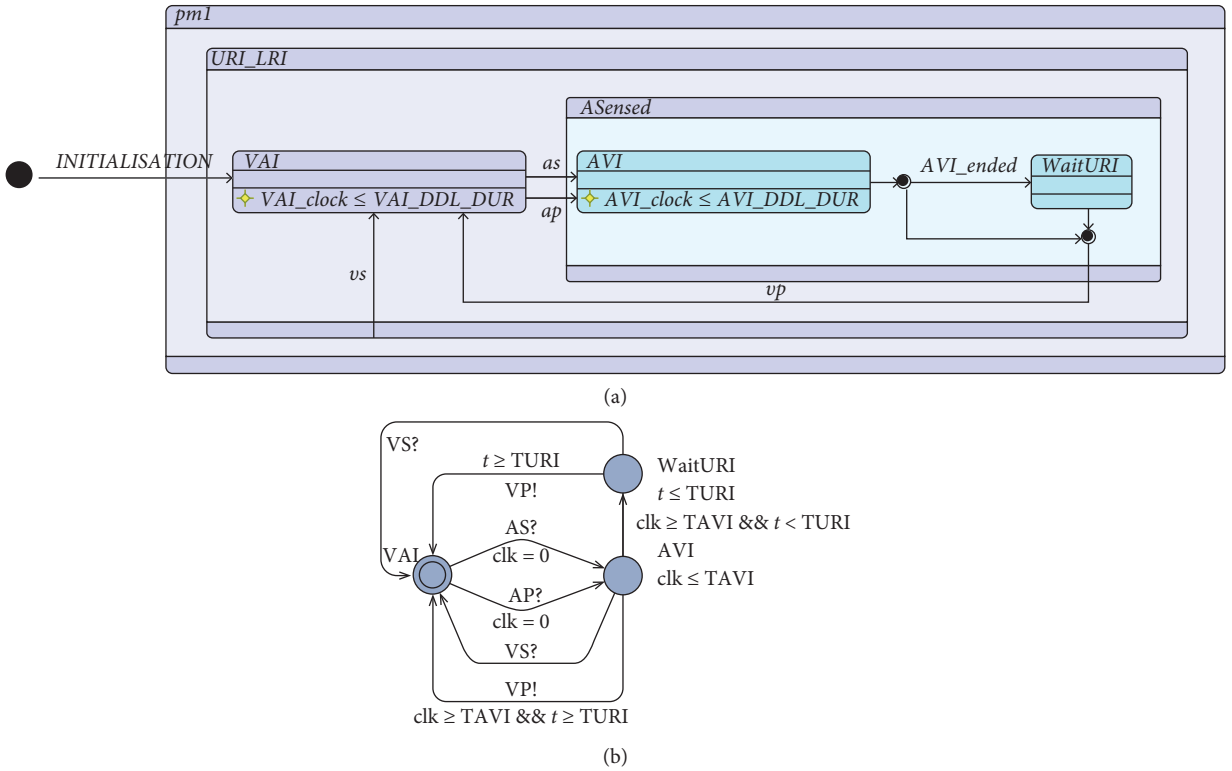

Figure 19: Second level refined models and the corresponding timed automata. (a) Second level refined state machine; (b) AVI timed automata.

The "Timing Properties" method is a mature and complete real-time Event-B modeling method. It uses three patterns named Deadline (A; B; *t*), Delay (A; B; *t*), and Expiry (A; B; *t*) to represent deadline, delay, and timeout scenes in real-time systems and provides strict Event-B semantics for these time patterns. The Timing Properties method uses a tree-like ERS diagram (event refined structure diagram [33]) to express the order and refined structure of events, thereby supporting the refinement and decomposition of the real-time Event-B model. In addition, Sarshogh also composites three patterns to form various complex real-time Event-B refined patterns. The principle of the Timing Properties method is shown in Figure 22. Modelers can use the timing plug-in to add time constraints, automatically generate proof obligations, and automatically discharge these proof obligations. Sarshogh claims that most of the proof obligations (95%) can be discharged with an automatic theorem prover. Therefore, the time properties method can well support Event-B modeling of large and complex real-time systems as well as automatic proof of consistency. But the Time Properties method does not support the expression of concurrent real-time objects. Although the ERS diagram can express the vertical event decomposition relationship well, it is not intuitive enough to express the order of events at a particular refinement level. People need to understand the actual order of events through analysis.

The Timing Interval method proposed by Sulskus uses "Timing Interval Notation" to express various time constraints. It adds a higher-level concept of the interval, extends the timing notation, and introduces the abort event. In

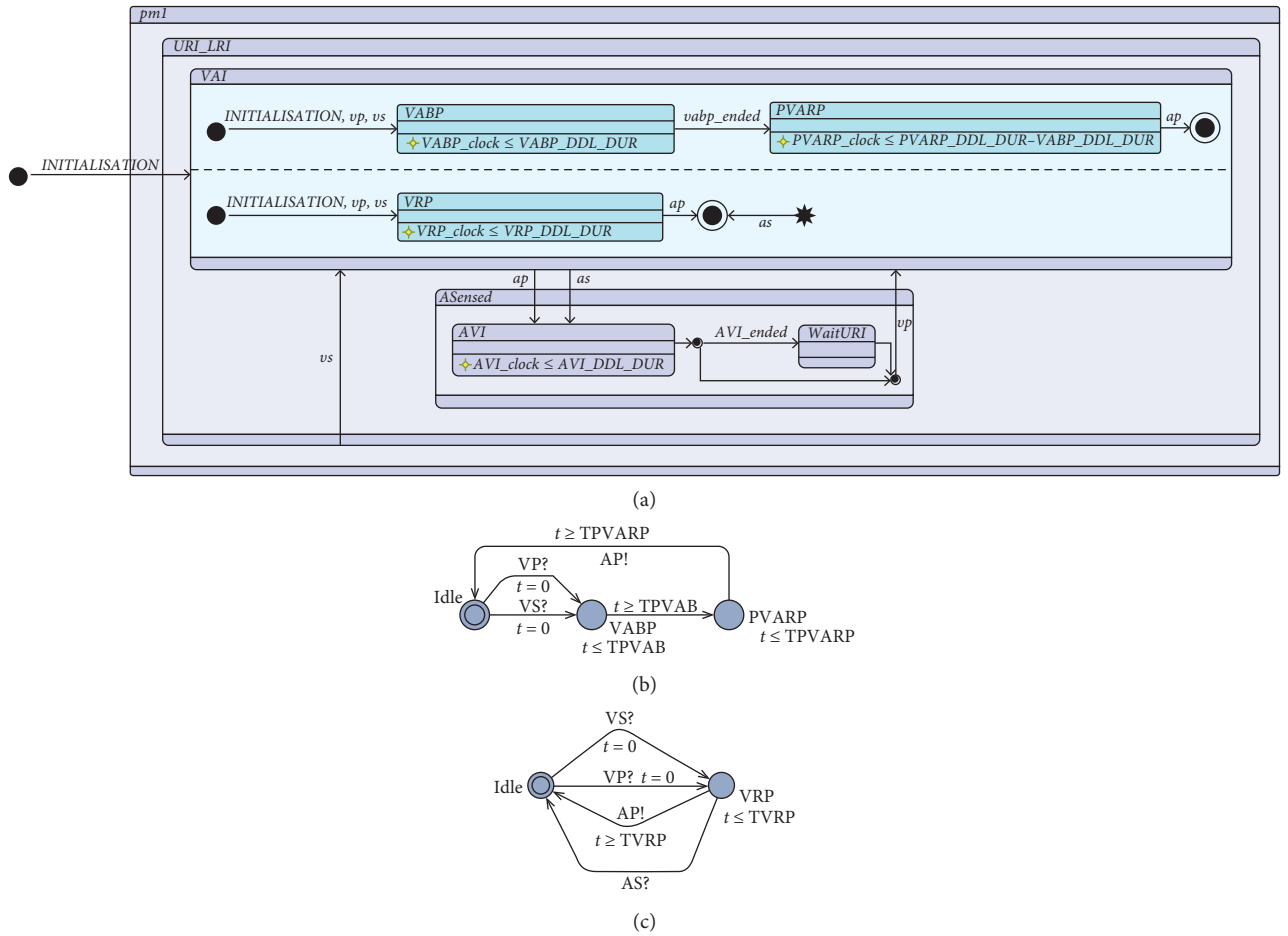

Figure 20: Third level refined model and the corresponding timed automata. (a) Third level refined state machine; (b) VABP timed automata; (c) VRP timed automata.

Table 4: The evaluation standard of real-time Event-B methods.

Number of standards	Description of standard
S1	How does it express the order of events in a real-time system?
S2	How does it express the duration of time?
S3	How does it express the vertical refinement process of the Event-B model of the real-time system?
S4	How does it express the concurrent real-time objects?
S5	How does it support the automatic generation of the proof obligations in the model refinement process?
S6	How does it support the automatic proof of time-related properties?

addition, the time interval method uses an improved ERS diagram to make the expression of the event decomposition relationship in the vertical refinement process clearer. For example, a square marked with "Sub-Int" is used to express the subinterval, and a square marked with the letter "*P*" is used to express the parallel composition relationship of the subintervals. The time interval method also uses the iUML-B state machine to express the order of events on a particular refinement level as well as the parallel composition of real-time concurrent objects. An example of the Timing Interval method is shown in Figure 23. Sulskus also proposes various compositions of real-time Event-B refinement patterns and uses the "tiGen" plug-in to express time intervals as well as automatically generating proof obligations to ensure refinement consistency. In the Event-B model of three complex real-time systems constructed by using the Timing Interval method, 100% of the proof obligation is automatically proved.

It can be seen that the method of using various automated plug-ins to support real-time system modeling within the Event-B framework has become more and more powerful. But none of these methods can avoid the disadvantages of Event-B itself; that is, neither can they support the expression and verification of various time properties (such as LTL properties such as fairness and liveness and similar TCTL properties), nor can they "naturally" preserve these

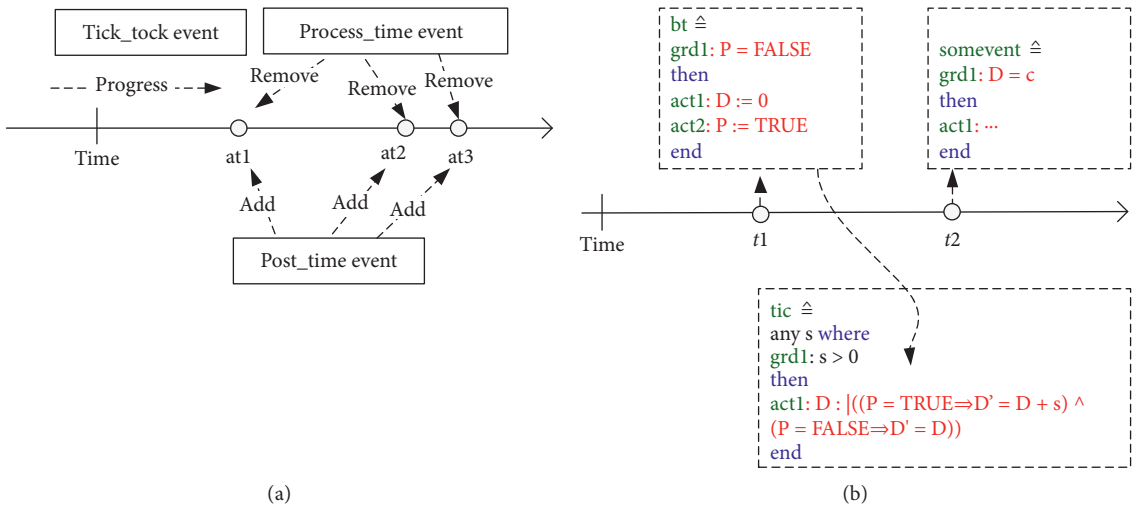

Figure 21: Time constraint pattern and Duration pattern. (a) Time constraint pattern and (b) Duration pattern.

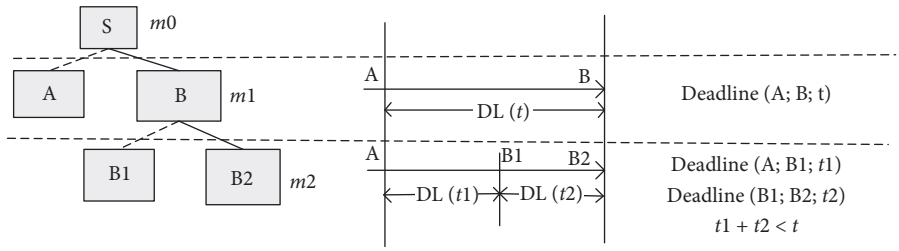

Figure 22: Timing Properties method.

properties in the refinement process. As Hoang and Schneider said [34], "The challenge is to identify more natural ways of integrating Event-B and LTL so that LTL properties can be preserved by Event-B refinement, which is not the case in general." Our point is that it is better and easier to adopt an integrated formal method than to integrate Event-B and LTL properties verification by using the syntax and semantics of Event-B just like the work done by Iliasov and Vain et al.

The method proposed in this paper uses the iUML-B functional state machine to express the order of events at a particular refinement level and uses the ticker state machine to express the elapse of time. We use the state node of the iUML-B state machine to express time intervals like Sulskus did except that our method needs to add various time invariants to the state node manually, while the invariants of the Timing Interval method are automatically generated. In terms of expressing the event decomposition relationship between the vertical refinement levels, our method is better than the time constraint method and the duration pattern, but it is not as clear as the ERS diagram of the Time Properties method and the Time Interval method. One must observe and compare the state machines of two refinement levels to understand how the Nth level events are decomposed into the $N+1$-th level events. In terms of the expression of concurrent real-time objects, our method is optimal. We borrow from the idea of Dong's timed automata patterns and proposed various horizontal expansion patterns, which are more helpful for modeling the Event-B model of a complex system with a large number of concurrent real-time objects.

The limitation of our method is that we do not provide any plug-ins to support the automatic generation of proof obligation and automatic theorem proving. This is because the motivation of our work is to eliminate the grammatical and semantic gap between the event-based formal system and the automata-based formal system so that the Event-B model can be more easily converted to a timed automata model. Thereby, the time properties of the Event-B model can be verified easier.

Unlike the various existing methods that directly convert the Event-B model to UPPAAL timed automata, we only want to convert the "time aspect" of the Event-B model to timed automata. The reason is that, first of all, there are many unmappable parts between the syntax elements of the Event-B model and those of the UPPAAL timed automata. For example, a large number of discrete mathematical expressions in the Event-B model cannot be converted into the UPPAAL model directly. Secondly, when the Event-B model is very complex, this translation process is very time-consuming, especially when these Event-B codes are generated by automatic code generation tools (such as tiGen).

The evaluation of various real-time Event-B modeling methods including our method is given in Table 5.

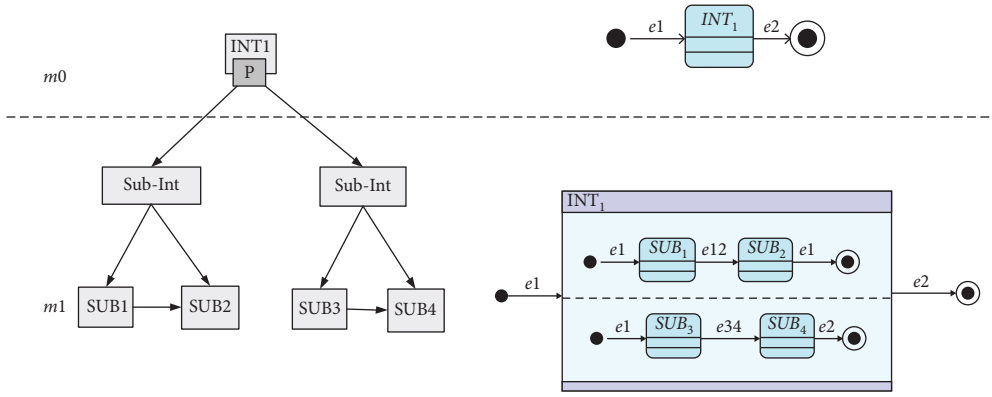

Figure 23: Timing Interval method.

Table 5: The evaluation of various real-time Event-B modeling methods.

Methods	Evaluation standards					
	S1	S2	S3	S4	S5	S6
M1	Event-B code	Event-B code	N/A	N/A	N/A	N/A
M2	Event-B code	Duration	N/A	N/A	N/A	N/A
M3	ERS diagram	Three patterns	ERS diagram	N/A	TP plug-in	Mostly automated
M4	ERS diagram	iUML-B state machine	ERS diagram and iUML-B state machine	ERS Diagram and iUML-B state machine	tiGen plug-in	Fully automated
M5	iUML-B state machine	iUML-B state machine	iUML-B state machine	iUML-B state machine	N/A	N/A

M1: time constraints method; M2: duration pattern method; M3: Time Properties method; M4: Time Interval method; M5: iUML-B time state machine method.

7. Conclusions

Theorem proving language usually uses first-order predicate logic as its theoretical basis, so it has great limitations in expressing time and verifying time properties. But, for software engineers, a formal system like Event-B based on theorem proving is very suitable for the development process from system abstract model to concrete model as well as software code. Therefore, using Event-B to model real-time systems is an inevitable way to verify complex software. The current real-time Event-B modeling work mainly focuses on the modeling and theorem proving methods within the Event-B framework, which brings more learning costs and barriers to understanding. This paper proposes a real-time system modeling and refinement framework based on the iUML-B state machine, in which a series of reusable Event-B design patterns for modeling real-time systems are given. Using these patterns, modelers can quickly build an Event-B model of a real-time system and convert it to a timed automata model to complete time property verification.

In the future, we will continue to study the automatic translation method from the iUML-B time state machine to UPPAAL timed automata and prove the correctness of the translation process.

Data Availability

Data supporting this study is available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by project of Natural Science Foundation of Shaanxi Province, China (Grant no. 2020JM-633) and project of the Key Research and Development Program of Shaanxi Province, China (Grant no. 2020GY-084).

References

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

[2] D. Cansell and D. M. Rehm, "Time constraint patterns for event B development," *Proceedings of the International Conference of B Users*, pp. 140–154, Springer, Berlin, Germany, 2007.

[3] 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, Linz, Austria, May 2016.

[4] J. S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi, "Timed automata patterns," *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 844–859, 2008.

[5] M. Butler and J. Falampin, "An approach to modelling and refining timing properties in B," *Proceedings of the Refinement*

of Critical Systems (RCS), 2002, https://eprints.soton.ac.uk/256235/1/ButlerFalampin.pdf.

[6] J. Rehm, "A method to refine time constraints in event B framework," in *Proceedings of the Automatic Verification of Critical Systems-AVoCS 2006*, pp. 173–177, Nancy, France, September 2006.

[7] J. Rehm, "A duration pattern for event-B method," in *Proceedings of the 2nd Junior Researcher Workshop on Real-Time Computing-JRWRTC 2008*, Rennes, France, February 2008.

[8] J. Rehm, "Proved development of the real-time properties of the IEEE 1394 root contention protocol with the event-B method," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 1, pp. 39–51, 2010.

[9] S. Mohammad Reza and B. Michael, "Extending event-B with discrete timing properties," *Special Issue on Automated Verification of Critical Systems*, 197 pages, University of Southampton, Southampton, England, 2013.

[10] S. Mohammad Reza and B. Michael, "Specification and refinement of discrete timing properties in event-B," *Automated Verification of Critical Systems*, vol. 46, pp. 1863–2122, 2011.

[11] G. Sulskus, *An Investigation into Event-B Methodologies and Timing Constraint Modelling*, 272 pages, University of Southampton, Southampton, England, 2017.

[12] G. Sulskus, M. Poppleton, and A. Rezazadeh, *An Interval-Based Approach to Modelling Time in Event-B*, pp. 292–307, Springer, Berlin, Germany, 2015.

[13] A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, "Augmenting event-B modelling with real-time verification," in *Proceedings of the 2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pp. 51–57, Zurich, Switzerland, June 2012.

[14] J. Vain, *Developing Multi-View Contracts Using Event-B and Uppaal Timed Automata*, pp. 126–134, IEEE, NY, USA, 2016.

[15] J. Vain, L. Tsiopoulos, and P. Boström, "Integrating refinement-based methods for developing timed systems," in *From Action Systems to Distributed Systems: The Refinement Approach*, L. Petre and E. Sekerinski, Eds., pp. 171–185, CRC Press. Boca Raton, Fl, USA, 2016.

[16] J. Berthing, P. Boström, K. Sere, L. Tsiopoulos, and J. Vain, *Refinement-based Development of Timed Systems*, pp. 69–83, Springer, Berlin, Germany, 2012.

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

[18] C. Snook, T. S. Hoang, and M. Butler, *Analysing Security Protocols Using Refinement in iUML-B*, pp. 84–98, Springer, Berlin, Germany, 2017.

[19] C. Snook and M. Butler, "UML-B and event-B: an integration of languages and tools," *Book UML-B and Event-B: An Integration of Languages and Tools, Series UML-B and Event-B: an Integration of Languages and Tools*, pp. 336–341, ACTA Press, Calgary, Canada, 2008.

[20] M. Y. Said, *Methodology of Refinement and Decomposition in UML-B*, Ph.D. Thesis, University of Southampton, Southampton, England, 2010.

[21] M. Y. Said, M. Butler, and C. Snook, "Class and state machine refinement in UML-B," in *Proceedings of Workshop on Integration of Model-Based Formal Methods and Tools (Associated with IFM 2009)*, Zurich, Switzerland, July 2009.

[22] M. Y. Said, M. Butler, and C. Snook, "Language and tool support for class and state machine refinement in UML-B," *Proceedings of International Symposium on Formal Methods*, pp. 579–595, Springer, Berlin, Germany, 2009.

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

[24] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[25] S. Yovine, "KRONOS: a verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 123–133, 1997.

[26] T. Amnell, *TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems*, pp. 60–72, Springer, Berlin, Germany, 2003.

[27] H. Ping, *Object-Z/TCOZ and Timed Automata; Projection and Integration*, Ph.D. thesis, Huazhong University of Science and Technology, Wuhan, China, 2008.

[28] J. S. Dong, P. Hao, S. Qin, J. Sun, and Y. Wang, "Timed patterns: TCOZ to timed automata," in *Proceedings of the 6th International Conference on Formal Engineering Methods*, pp. 483–498, ICFEM 2004, Seattle, WA, USA, November 2004.

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

[30] H. Peng, C. Du, L. Rao, and Z. Liu, "LTS semantics model of Event-B synchronization control flow design patterns," *Journal of Information Processing Systems (JIPS)*, vol. 15, no. 3, pp. 570–592, 2019.

[31] Z. Jiang and R. Mangharam, "High-confidence medical device software development," *Foundations and Trends in Electronic Design Automation*, vol. 9, no. 4, pp. 309–391, 2015.

[32] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam, "Modeling and verification of a dual chamber implantable pacemaker," in *International Conference on Tools and Algorithms for the Construction and Analysis of System*, pp. 188–203, Springer, Berlin, Germany, 2012.

[33] 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.

[34] T. S. Hoang, S. Schneider, H. Treharne, and D. M. Williams, "Foundations for using linear temporal logic in Event-B refinement," *Formal Aspects of Computing*, vol. 28, no. 6, pp. 909–935, 2016.