Electronic Communications of the EASST
Volume 46 (2011)

Proceedings of the 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011)

Specification and refinement of discrete timing properties in Event-B

Mohammad Reza Sarshogh, Michael Butler

15 pages

Guest Editors: Jens Bendisposto, Cliff Jones, Michael Leuschel, Alexander Romanovsky
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Specification and refinement of discrete timing properties in Event-B

Mohammad Reza Sarshogh[1], **Michael Butler**[2]

[1] mrs2g09@ecs.soton.ac.uk, http://www.ecs.soton.ac.uk/people/mrs2g09
University of Southampton, Southampton, UK

[2] mjb@ecs.soton.ac.uk, http://users.ecs.soton.ac.uk/mjb/
University of Southampton, Southampton, UK

Abstract: Event-B is a formal language for systems modeling, based on set theory and predicate logic. It has the advantage of mechanized proof, and it is possible to model a system in several levels of abstraction by using refinement. Discrete timing properties are important in many critical systems. However, modeling of timing properties is not directly supported in Event-B. In this paper we identify three main categories of discrete timing properties for trigger-response pattern, *deadline*, *delay* and *expiry*. We introduce language constructs for each of these timing properties that augment the Event-B language. We describe how these constructs can be mapped to standard Event-B constructs. To ease the process of using the timing constructs in a refinement-based development, we introduce patterns for refining the timing constructs that allow timing properties on abstract models to be replaced by timing properties on refined models. The language constructs and refinement patterns are illustrated through some generic examples. Event-B refinement allows atomic events at the abstract level to be broken down into sub-steps at the refined level. The goal of our refinement patterns is to provide an easy way to represent and correctly refine timing constraints on abstract atomic events with more elaborate timing constraints on the refined events. This paper presents an initial set of patterns.

Keywords: Real-time System, Event-B, Event, Deadline, Delay, Expiry, Refinement Patterns

1 Introduction

In Event-B [Abr10], systems are modeled formally by a collection of events (i.e. guarded actions) that act on abstract variables. The aim in this work is to introduce an approach to formally model the timing properties for the trigger-response pattern in control systems. This pattern is common and useful in specification of control systems. It is natural to talk about these kinds of systems in term of possible events of the system. For example in the trigger-response pattern, trigger and response are both events of the control system.

One of the main advantages of Event-B method is its support for stepwise modeling by refinement. The other strength of this method is the mechanized proof obligation generator and the prover which make the verification process, efficient and productive. These advantages of Event-B, make it a suitable approach for formal modeling of critical systems.

An Event-B model has two main parts, context and machine. The context specifies the static

part of the system and the machine models the dynamic part. In the machine, system behavior and its properties can be modeled by using states variables, invariants and events. Variables represent the current state of the system. Invariants specify the global specifications of the state variables and system behaviors. Finally, events represent the transition of the system from a state to another. Events are guarded atomic actions where guards specify the state of the machine where the event can occur in, and actions indicate how the that event modifies the state variables. By refining a machine it is possible to introduce new state variables and events, strengthen the guards of the abstract events or introduce new actions on new state variables. Standard refinement techniques are used to verify the refinement between models at different abstraction levels.

Event-B lacks explicit support for expressing and verifying timing properties. Modeling time-critical systems, using Event-B has been investigated in several studies. What distinguish our work, is categorizing timing constraints in three groups, introducing a systematic way of encoding each of them in an Event-B model, introducing patterns for refining timing constraints and proving satisfaction of abstract timing constraint by their concrete ones. In this way, the consistency of the system timing properties in the system specification can be proved by using refinement feature of the language.

2 Timing Properties Categories

In order to formalize the process of adding time properties to an Event-B model, it has been decided to categorize the mostly used time related specifications in time-critical system descriptions. Hence, several time-critical system specifications like a car gear-controlling system, a message passing algorithm in a network, a water tank level controller, etc., had been studied to extract their timing properties. The next step was to categorize them in several groups according to the nature of their restriction. The result was three groups of timing properties; *Deadline*, *Delay* and *Expiry*. These three will be explained in more details in the following. As mentioned before, these timing properties are essentially trigger-response patterns, and trigger and response are naturally modeled as events. As a result, all the definitions in this work are event based, where A is the trigger event and B is the response event.

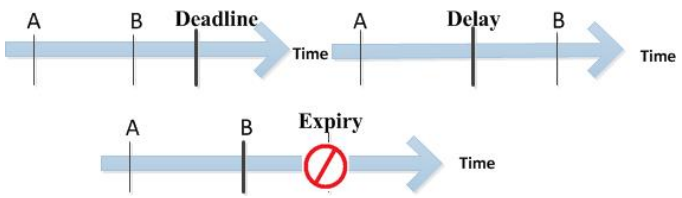


Figure 1: Time Boundary Diagrams

Imagine a system with two events, A and B where first event A has to happen to make event B possible to occur. The three types of timing boundaries which may be declared between event A and event B are as follow:

- **Deadline:** Event B must occur within time D of event A occurring,

- **Delay:** Event B cannot occur within time W of event A occurring,
- **Expiry:** Event B cannot occur after time E of event A occurring.

Based on the definition of these three restrictions, the deadline forces an event to happen before a specific time, delay prevents an event from happening before a specific time and expiry prevents an event from happening after a specific time. Accordingly, by having a deadline between two events, it is guaranteed that by the deadline the deadline event has already occurred, by having delay it is guaranteed that there will be a minimum gap between occurrence of two events, and by having expiry it is guaranteed that if the restricted event has happened it was before a specific time. In order to have a better understanding of these constraints, Figure 1 illustrates how these boundaries restrict events.

In this section delay, deadline and expiry have been introduced informally. In the following we explain how they are formalized.

3 Modeling Timing Properties In Event-B

In order to explicitly represent timing properties we extend the Event-B syntax with constructs for deadlines, delays and expiries. These timing properties place a discrete timing constraint between trigger events and response events. A typical pattern is a trigger followed by one of a choice of responses thus our timing constructs specify a constraint between a trigger event A and a set of response events Bx. The syntax for each of these constructs is as follows:

- **Deadline**(A, {B1,..,Bn}, t),
- **Delay**(A, {B1,..,Bn}, t),
- **Expiry**(A, {B1,..,Bn}, t).

The property **Deadline**(A, {B1,..,Bn}, t) means that one of the response events Bx must occur within the time t of trigger event A occurring. In the case of delay, if any of the events in the response set happens it has to happen after its declared delay. Finally in the case of expiry, if any of the events in the expiry set happens it has to happen before the specified expiry time.

Now a specification consists of an Event-B machine consisting of variables, invariants and events, together with a list of timing properties using the above syntax. Having the annotations standardizes the process of specifying discrete timing properties in Event-B models and allows us to define patterns for refining timing properties as we show in Section 4.

We give a semantic to our timing constructs by translating them into Event-B variables, invariants, guards and actions that are added to the machine to which the timing properties belong. The effect of additions to the Event-B machine will be to add clock increment event and constrain further the order between events. In particular they constrain the order between trigger, response and clock increment events. For example, the additional Event-B elements that a deadline property give rise to will prevent more than t clock increments occurring in between a trigger event and a corresponding response event.

We define rules for encoding each of the three timing constructs in Event-B in turn. In each case we assume there is already a partial order between the trigger event and the corresponding

response events, that is, we assume that the response events are only enabled after the corresponding trigger event has occurred. This ordering assumption is encoded using boolean flags as shown in Figure 2(a). As shown in Figure 2(a), event A sets the boolean variable A as one of its actions, so when variable A has the value of $TRUE$, it shows event A has happened. Also, in event Bx the flag of event A will be checked to see if event A has already happened. Other than checking the flag and setting the flag, in their guard, $Xgrds$ represents the other possible guards of the event and in the action section $Xacts$ represent the other possible actions of the event.

Note we do not assume that the trigger and response events will occur only once. Typically the trigger and response events will be part of an iterative loop and the ordering flags will be reset at the end of each iteration of the loop by an appropriate event.

3.1 Modeling Delay

In this section we explain how delay is encoded in an Event-B model. As mentioned before, in order to have discrete time in Event-B a natural number variable is declared to represent the current time in the machine and an event is added to model the progress of time.

In order to explain how delay is encoded in Event-B, we will go through the process, for a generic trigger event A and some generic alternative response events $B1 \ldots Bn$.

EVENT $A \mathrel{\widehat{=}}$
  WHERE
    $A = FALSE$
    $Agrds$
  THEN
    $A := TRUE$
    $Aacts$
  END
EVENT $Bx \mathrel{\widehat{=}}$
  WHERE
    $A = TRUE$
    $Bx = FALSE$
    $Bxgrds$
  THEN
    $Bx := TRUE$
    $Bxacts$
  END

(a) Event A and Bx without delay

EVENT $A \mathrel{\widehat{=}}$
  WHERE
    $A = FALSE$
    $Agrds$
  THEN
    $A := TRUE$
    $tA := time$
    $Aacts$
  END

EVENT $Bx \mathrel{\widehat{=}}$
  WHERE
    $A = TRUE$
    $Bx = FALSE$
    $time \geq tA + t$
    $Bxgrds$
  THEN
    $Bx := TRUE$
    $Bxacts$
  END
EVENT $Tick\_Tock \mathrel{\widehat{=}}$
  THEN
    $time := time + 1$
  END

(b) Event A and Bx with delay

Figure 2: Events A and Bx in 2(a) along with the Delay property will implicitly define the model in 2(b)

There are two steps in order to add a delay constraint which is defined as follow to an Event-B model:

$$Delay(A, \{B1, .., Bn\}, t). \tag{1}$$

First the occurrence time of the trigger event is recorded in a variable (tA). Then in the event which should be delayed (event A), a guard is needed which forces the event to be eligible to occur after the stated delay period has been passed from the occurrence of the trigger event. In Figure 2 a general pattern of delayed trigger-response and the event which progress the time

(Tick_Tock), in an Event-B model, has been shown. As explained in this section, it is possible to add a delay to a standard Event-B model.

3.2 Modeling Expiry

Modeling expiry is similar to the delay. Again the first step is to record the occurrence time of the trigger event and the next step is to guard the restricted event according to the recorded time and the specified expiry period. Suppose, we want to force timing property 2 to the trigger-response pattern which is shown in Figure 3(a), how the model should be changed to contain the timing property is shown in Figure 3(a).

$$Expiry(A, \{B1, .., Bn\}, t) \tag{2}$$

EVENT $A \widehat{=}$
  WHERE
    $A = FALSE$
    $Agrds$
  THEN
    $A := TRUE$
    $Aacts$
  END
EVENT $Bx \widehat{=}$
  WHERE
    $A = TRUE$
    $Bx = FALSE$
    $Bxgrds$
  THEN
    $Bx := TRUE$
    $Bxacts$
  END

(a) Event A and Bx without expiry

EVENT $A \widehat{=}$
  WHERE
    $A = FALSE$
    $Agrds$
  THEN
    $A := TRUE$
    $tA := time$
    $Aacts$
  END

EVENT $Bx \widehat{=}$
  WHERE
    $A = TRUE$
    $Bx = FALSE$
    $time \leq tA + t$
    $Bxgrds$
  THEN
    $Bx := TRUE$
    $Bxacts$
  END
EVENT $Tick\_Tock \widehat{=}$
  THEN
    $time := time + 1$
  END

(b) Event A and Bx with expiry

Figure 3: Events A and Bx in 3(a) along with the expiry property will implicitly define the model in 3(b)

As shown in Figure 3, in order to have expiry for an event, an action is needed to record the occurrence time in the trigger event (event A), and a guard in the restricted event to prevent it from happening if the expiry period has been passed.

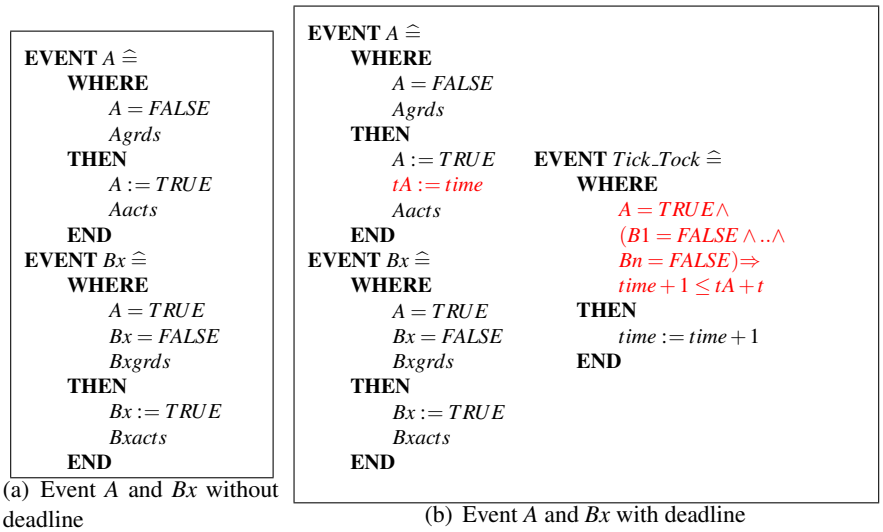
3.3 Modeling Deadline

In order to encode expiry and delay, just the trigger and the response events are involved. But, this is not the case for modeling deadline. In order to model a deadline the $Tick\_Tock$ event is involved as well, because if the trigger event has happened, we want to force the response event to occur, before passing the deadline. Guardin the $Tick\_Tock$ event is a possible way to enforce one of the events $B1$ to Bn to occur before the deadline passes. As it will be explained in Section 6 guarding the clock in order to model deadline has been used in several timed specifications theories and tools.

Suppose, a deadline has been declared by our timing annotation as follow:

$$Deadline(A, \{B1, .., Bn\}, t). \tag{3}$$

In order to model this restriction in an Event-B model, first the occurrence time of event A should be recorded by adding a new action. Then a guard on the $Tick\_Tock$ event is needed, to enforce the deadline. In Figure 4 how deadline 3 can be added to a standard Event-B model is shown in detail.



(a) Event A and Bx without deadline

(b) Event A and Bx with deadline

Figure 4: Events A and Bx in 4(a) along with the deadline property will implicitly define the model in 4(b)

Multiple deadline constraints may be added to a model. In this case, a deadline guard similar to what has been shown in Figure 4(b) should be added to the $Tick\_Tock$ event for each deadline constraint.

It is possible to cause a deadlock by declaring a longer delay between two events than an existing deadline between them. There are two approaches to detect this kind of deadlock, either by running a model checker (e.g. ProB) and then check the uncovered events or by declaring an invariant which implies if a deadline guard is not true (current time is equal to the deadline and none of the restricted event has yet occurred) then one of the restricted events should be eligible to occur. As a result, if there is a deadlock, the invariant will not be proved for the $Tick\_Tock$ event.

4 Some Patterns to Refine Deadline, Delay and Expiry and Their Uses

In this section, some patterns of refining the introduced types of timing boundaries will be explained and their uses in order to synchronize different events will be shown by explaining some

general examples. In this section Event Refinement Diagrams [But09] are used to present the order between events of a refinement and also their relations with their abstract events. As a result the Event Refinement Diagram notation will be explained briefly in the next section.

4.1 Refinement and Event Refinement Diagram

Usually, a real world system has a complex specification with a lot of details. If we want to model all the details of a system specification in a single stage, the complexity and the size of the model can cause a lot of difficulties. One solution is to model systems, step by step by using refinement. The system specification should be broken to different levels of abstraction. Then, the first step will be the modeling of the most abstract specification of the system. Then by each refinement more details of the system specification will be added to the model. By this approach, the model will be a more explicit representation of the target system by each refinement.

In Event-B refinement process, it is possible to introduce new events which do not exist in the abstract machine. Other events extend abstract events or refine them. Those events which do not exist in the abstraction, refine *skip*. They model the pre-steps or post-steps of abstract events which are not visible in the abstraction in order to reduce the complexity. Although they do not refine any abstract event, they are related to abstract events.

In order to simplify tracking the relations between abstract and concrete events, refinement diagrams have been introduced by Butler in [But09]. In a refinement diagram there is a tree structure in which the abstract event is positioned as the root of the tree, and its concrete events or events which are new but model the pre/post-steps of the abstract events are represented as leaves. The other characteristic of this notation is that the concrete events which exist in the abstract machine and refine abstract events, are connected to their corresponding abstract event by solid lines and the new events which model the pre/post-steps of abstract events are connected by to their related abstract events by dash lines. Figure 5 is an event refinement diagram, illustrating

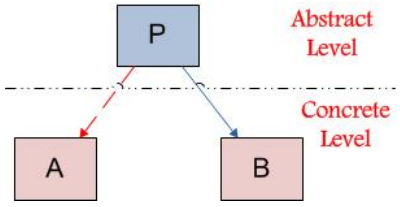


Figure 5: Refinement Diagram Example

that abstract event P is refined by a combination of concrete event A followed by concrete event B. Event A is a pre-step of event B that refines skip, while event B refines event P.

By this introduction to the Refinement Diagram, how the elicited timing properties can be refined, will be explained in the following.

4.2 Refining a Deadline to Sequential Sub-Deadlines

Consider an abstract model of a system where there is a deadline between event A and event B. As shown in Figure 7, event B can only occur if event A has already happened. The deadline

properties for this level of abstraction, is shown in Figure 7(a). In the next refinement event B will be broken to two steps, as shown in Figure 6. By breaking event B to $B1$ followed by $B2$, its related deadline needs to be broken too. Also the other important issue is that, the abstract event has been refined by the second step, because the accomplishment of the second step is equivalent to accomplishment of abstract event(B). So the first step should refine *skip*.

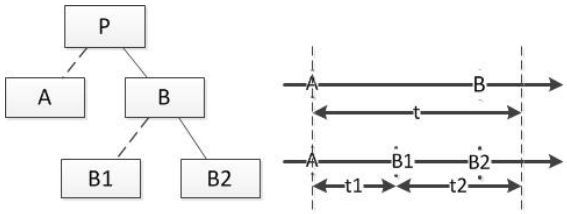


Figure 6: Refining an abstract deadline to two sub-deadlines

$Deadline(A,\{B\},t)$.

EVENT $A \widehat{=}$
WHERE
$A = FALSE$
$Agrds$
THEN
$A := TRUE$
$Aacts$
END

EVENT $B \widehat{=}$
WHERE
$A = TRUE$
$B = FALSE$
$Bgrds$
THEN
$B := TRUE$
$Bacts$
END

(a) Event A and B

$Deadline(A,\{B1\},t1)$
$Deadline(B1,\{B2\},t2)$
Where
$t1 + t2 \leq t$

EVENT $B1 \widehat{=}$
WHERE
$A = TRUE$
$B1 = FALSE$
$B1grds$
THEN
$B1 := TRUE$
$B1acts$
END

EVENT $B2$ *refines* $B \widehat{=}$
WHERE
$B1 = TRUE$
$B2 = FALSE$
$B2grds$
THEN
$B2 := TRUE$
$B2acts$
END

(b) Event $B1$ and $B2$

Figure 7: Events A and B in abstract Machine in 7(a) and events $B1$ and $B2$ in the concrete machine in 7(b)

Now, in order to respond to the trigger, two steps have to be accomplished where each of them has its own deadline. In the concrete level, the trigger event of deadline constraint for event $B1$ is event A and the trigger event for the deadline of event $B2$ is event $B1$. Hence, the abstract deadline should be broken as shown in Figure 7(b) where the sum of new deadlines does not violate the abstract deadline.

The relation between the concrete states and the abstract ones is expressed by a *gluing invariant* [ABH+10] in Event-B, in order to verify the refinement. Two kinds of gluing invariants are needed in order to prove that the concrete deadlines satisfy their abstraction. The first type is required to clarify the relation between the order of the abstract and concrete events which are involved in the deadline. The other type is needed to specify the relation between the new deadlines in the concrete machine and the abstract deadline. In the explained pattern these invariants should be as follow:

- The relation between abstract events and its refining events ($B2$ and B are the boolean

variables which act as the occurrence flag of events $B2$ and B):

$$B2 = B, \tag{4}$$

- The order between concrete events:

$$B1 = TRUE \Rightarrow A = TRUE, \tag{5}$$

- The relation between the abstract deadline trigger time and its concrete one (tA is an integer variable which records the occurrence time of event A and $tB1$ does the same thing for event $B1$):

$$B1 = TRUE \Rightarrow tB1 \leq tA + t1, \tag{6}$$

$$A = TRUE \wedge time > tA + t1 \Rightarrow B1 = TRUE. \tag{7}$$

Invariant (4) specifies that the occurrence of event $B2$ is equivalent to the occurrence of event B. Invariant (5) specifies that event $B1$ must occur after event A. Invariant (6) shows the relation between occurrence time of even $B1$ and the trigger time of abstract deadline and Invariant (7) specifies the deadline for occurrence of event $B1$ which is the trigger for occurrence of event $B2$. Invariant (7) is required in order to prove Invariant (6) for event $B1$, because it specifies that $B1$ must occur before $tA + t1$.

It should be mentioned that the abstract deadline can be broken into more than two sub-deadlines either by successive refinement steps or by refining the abstract event with more than two sub sequential events in one refinement step.

4.3 Refining An Abstract Deadline to Alternative Sub-deadlines

Often, when a process has to finish by a specific time there is a recovery scenario which will guarantee that by the deadline either the desired response or some recovery response will be achieved. So by the deadline either the normal or the recovery scenario has been accomplished. For example, consider, instead of refining event B in the example of Section 4.2, by two sequential sub steps, it has been refined by breaking it into two alternative events, $B1$ and $B2$. So, after occurrence of event A either event $B1$ or event $B2$ should happen. How event B and the abstract timing property are refined is shown Figure 8.

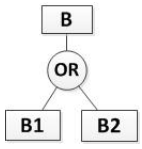
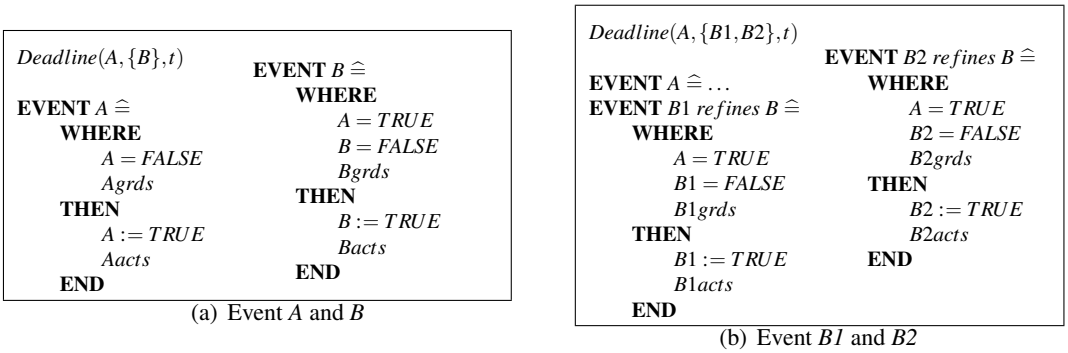


Figure 8: Refining an event to two possible events

As shown in Figure 9, in the refinement event B has been broken to event $B1$ (normal response) and event $B2$ (recovery response) and both of them refine the abstract event. As a result the deadline will be refined as shown in Figure 9(b). As explained in Section 3.3 by declaring a

deadline which has more than one member in its deadline set, we specify the behavior where, after occurrence of event A, before passing the deadline time, either process will be accomplished by occurrence of event $B1$ or event $B2$.



(a) Event A and B

(b) Event $B1$ and $B2$

Figure 9: Events A and B in abstract Machine in 9(a) and events $B1$ and $B2$ in the concrete machine in 9(b)

In this case the only kind of invariant which is required is the one which connected the concrete events occurrences to their abstract one. In the above example the required invariants will be as follow:

$$B2 = TRUE \vee B1 = TRUE \Leftrightarrow B = TRUE. \tag{8}$$

Based on invariant 8 occurrence of event B is equivalet to the occurrence of event $B1$ or event $B2$.

4.4 Refining Alternative Sub-Deadlines by Sequential Sub-Deadlines and Expiries

We now present a pattern for refining an abstract deadline by some alternative deadlines and then refine these by sequential deadlines.

In order to explain this pattern, the example of Section 4.3 will be continued. So, in the current state, we have a trigger event A and two alternative responses, event $B1$ and $B2$. The deadlines of each level of abstraction, are shown in Figure 9.

In the next refinement, each of the events $B1$ and $B2$ will be refined to two sequential steps and their deadline will be refined to two sequential deadlines, same as the pattern shown in Section 4.2 (event $B1$ will be broken to events $B1\_1$ and $B1\_2$ and event $B2$ will be broken to events $B2\_1$ and $B2\_2$). In this system, the first response case is desirable (modeled by event $B1$), but if its first step (modeled by event $B1\_1$) has not been accomplished by $t4$, the second response case (modeled by event $B2$) will be activated and its first step (modeled by event $B2\_1$) has to happen before the specified deadline ($t1$). As a result by the first deadline in the concrete machine, either the first response case has been activated or the second one (by occurrence of their first steps). For the next step, event $B1\_1$ triggers event $B1\_2$, and event $B2\_1$ triggers event $B2\_2$ as shown in Figures 10 and 11. The other specification of this system is that the deadline between the first ($B1\_1$) and the second ($B1\_2$) steps of the first response case ($B1$) is greater than

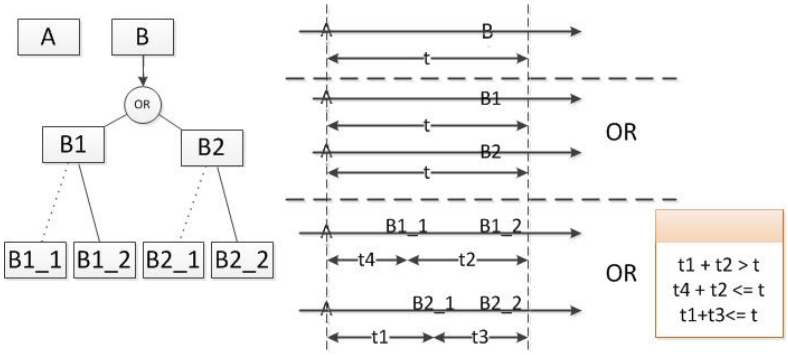


Figure 10: Refining each of the possible scenarios to two steps

$Deadline(A,\{B1\_1,B2\_1\},t1)$,
$Expiry(A,B1\_1,t4)$
$Deadline(B1\_1,\{B1\_2\},t2)$,
$Deadline(B2\_1,\{B2\_2\},t3)$
where
$t1+t2>t \wedge t4+t2 \leq t$
$t1+t3 \leq t$

EVENT $B1\_1 \widehat{=}$
WHERE
$A = TRUE$
$B1\_1 = FALSE$
$B1\_1grds$
THEN
$B1\_1 := TRUE$
$B1\_1acts$
END

EVENT $B1\_2$ *refines* $B1$ $\widehat{=}$
WHERE
$A = TRUE$
$B1\_1 = TRUE$
$B1\_2 = FALSE$
$B1\_2grds$
THEN
$B1\_2 := TRUE$
$B1\_2acts$
END

(a) Events *B1_1* and *B1_2*

EVENT $B2\_1 \widehat{=}$
WHERE
$A = TRUE$
$B2\_1 = FALSE$
$B2\_1grds$
THEN
$B2\_1 := TRUE$
$B2\_1acts$
END
EVENT $B2\_2$ *refines* $B2 \widehat{=}$
WHERE
$A = TRUE$
$B2\_1 = TRUE$
$B2\_2 = FALSE$
$B2\_2grds$
THEN
$B2\_2 := TRUE$
$B2\_2acts$
END

(b) Events *B2_1* and *B2_2*

Figure 11: Events *B1_1* and *B1_2* in 11(a) and events *B2_1* and *B2_2* in 11(b)

the equivalent deadline for the second alternative response case ($B2$). So according to the system specification the concrete deadlines should be as shown in Figure 11.

By the concrete deadlines, the abstract deadline will not be satisfied for the first response case ($t1+t2>t$). This problem is caused by the nature of deadline constraint. In the deadline we just guarantee that by passing the deadline time, at least one of the events of the deadline set has already happened.

As mentioned above, event $B1\_1$ has an expiry constraint too. So after a specific time, it cannot happen anymore and the only possible response will be the second response case. According to the system specification, event $B1\_1$ can just happen before $t4$ time units since event A occurrence. Hence, by enforcing this constraint by declaring an expiry as shown in Figure 11(a), the concrete timing properties will satisfy their abstract ones. It has been guaranteed in the model that if event $B1\_1$ happens, at most $t4$ time units have been passed from event A occurrence. From that time, event $B1\_2$ has $t2$ time units to happen. As a result, the abstract deadline is

satisfied. This pattern shows how combination of deadline and expiry can be useful in modeling and refining the timing properties of a time-critical system.

One question that can be raised here is why we do not use two disjunctive deadlines instead of a deadline and an expiry for this case. To explain it, we will apply this approach on a similar pattern. Suppose we want to encode these timing properties by two disjunctive deadlines, and we were to allow those to be declared as follow:

$$Deadline(A,\{B1\},t1) \vee Deadline(A,\{B2\},t2) \quad \textbf{Where } t2 > t1. \tag{9}$$

To encode this in an Event-B model, the guard on *Tick_Tock* event could be as follow:

$$\begin{aligned} A = TRUE \wedge B1 = FALSE \Rightarrow time + 1 \leq tA + t1 \qquad \vee \\ A = TRUE \wedge B2 = FALSE \Rightarrow time + 1 \leq tA + t2, \end{aligned} \tag{10}$$

Since $t2 > t1$ is easy to show that Guard (11) is equivalent to the guard we would use for the following deadline specification:

Deadline(A, { B1, B2 }, t2).

Intuitively, this is because guard (11) can be satisfied if the $B1$ event occurs after $t1$ time unit since occurrence of event A. As a result by having two disjunctive deadlines, the expiry constraint on event $B1$ will not be enforced.

In this section some approaches have been introduced in order to refine our three groups of timing properties. These patterns do not contain all the possible cases of refining timing properties and we are still working on other possible refinement patterns. In the following section how this research can be improved will be discussed briefly.

5 Future Work

We would like to develop a Plug-in for the Rodin tool-set in order to add time to a standard Event-B model automatically, based on the timing constraints declared in the form of deadline, delay and expiry which are expressed by the introduced annotations.

One of the features which has been added to Rodin [SPHB10], is decomposition. By using this feature, it is possible to break a machine in an Event-B model, to several independent machines where each machine represents one of the sub-components of the system. In this way it is possible to refine each sub-component independently. As a result, the complexity of the model will be decreased. Based on that, the other possible area to improve our pattern to add timing properties to an Event-B model, is to investigate the effect of decomposition on timing specification. It will strengthen the approach by eliciting the possible issues and challenges in decomposing a timed Event-B machine. What can be studied more about the decomposition are how the universal clock should be handled after the decomposition, how the time passing event should be decomposed, or how time related guards and actions will be separated between different machines in a decomposition.

Also, we would like to investigate the possibility of generalizing the introduced timing constraints in order to decrease the redundancy during the timing properties specification process in future.

6 Related Work

Many studies have been dedicated to formalizing and verifying timing properties of real-time systems. Delay, deadline and expiry can be seen in many of those works, sometimes with different names. In real-time calculus TCCS of Wang [Yi90] there is a delay construct $\varepsilon(d) \cdot P$, which enforce the model to wait for d time units and then behave as process P and time cannot proceed if d time-units passed and process P has not started yet. Same mechanism has been used in Timed Modal Specification of Cerans et al [CGL97] to model maximal progress assumption where there is a must modality which enforces the maximum delay to the model. Delay in TCCS and maximal progress in Timed Modal Specification present the same constraint as deadline in our work. Also, what is called a loose delay in Timed Modal Specification forces the same behavior as a delay does in our work. Besides, Urgent Event in Evans and Schneider work [ES00] has been encoded by preventing the time proceeding, if an urgent event is eligible to occur. This behavior of urgent event is the same as deadline events when current time is equal to the deadline and none of the deadline events have occurred yet. In Timed CSP [Sch99] time-out presents the same constraint as expiry does in our work and a delay in Timed CSP causes a similar behavior to what can be enforced by combining introduced delay and deadline in our work.

Modeling time-critical systems by using Event-B has been investigated in several studies. Butler et al. in [BF02] explained how it is possible to model discrete time in B (which is the root of Event-B), by having a natural number variable which represents the current time and an operation which increases it. In that study a deadline has been modeled by preventing the progress of time if the current time is equal to the deadline. This work does not investigate different kinds of timing constraint and timing constraint refinement have not been investigated. Cansell and Rehm in [CMR07] have modeled a message passing algorithm in Event-B by using similar principle, having a natural number variable, represents the current time, and an event which forwards the time, guarded by a set of activation times. Again in here, other kinds of timing constraints have not been mentioned, but more importantly, it is not possible to refine a timing constraint to several sub-timing constraints by this approach. Because, in order to do that, some new values should be added to the activation set in the refinement which is not possible without declaring a new activation set. The problem will be specifying the relation between the new activation set and its abstract one. Bryans et al. in [BFRR10] has introduced an approach to keep track of timing boundaries between different events in a model by adding them to a set and guarding events by them. In their study, deadline cannot be modeled. Similar to the previous approach, refining the timing constraint will be an issue because of tracking the timing constraints by a set for this approach too.

7 Conclusion

According to the gear controller case study[LPY01] which has been done in Even-B, and some other experiences, it seems that these three kinds of timing constraints can be used to model most of the timing properties of a time-critical system. In our case study we managed to first model the system without time, then declare the required timing constraints by using introduced annotations. In the end, according to the declared timing constraints, we added time to the

model. All the refinements' proof obligations which have been generated for relation between the concrete timing constraints and their abstractions have been discharged. If we manage to develop a plug-in which can add the required guards, actions, invariants and *Tick_Tock* event in order to add time to an Event-B model, based on declared timing properties, the process of modeling time-critical systems will be the same as modeling a non-time-critical system by using Event-B.

There are some similarities between our approach and the existing approaches to model time-critical systems. How we encoded deadline, delay and expiry is similar to the approach that timed automata [Alu99] verifiers use, like guarding state transitions (system events) or forcing timing properties to the global clock of the model. In timed automata, it is possible to check the temporal properties [Eme95] of a system. In this approach, the temporal properties can be checked by using refinements where a temporal property is modeled by an abstract invariant and by refinement how it will be gained by detailed behavior of the system will be modeled and verified. For example, in the abstraction we say, a gear-change request should be responded by an error message or a successful change, then by several refinements how system will manage to satisfy it will be modeled and verified.

Our approach is based on modeling discrete timing properties of reactive systems according to their events and through several levels of refinement. But it was not an isolated work, and we tried to develop an approach to add time to an Event-B model by learning from the existing works.

Acknowledgements: This work is partly supported by the EU research project ICT 214158 DEPLOY (Industrial deployment of system engineering methods providing high dependability and productivity) www.deploy-project.eu.

Bibliography

[ABH+10] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT* 12(6):447–466, 2010.

[ABHV06] J.-R. Abrial, M. J. Butler, S. Hallerstede, L. Voisin. An Open Extensible Tool Environment for Event-B. In *ICFEM*. Pp. 588–605. 2006.

[Abr10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Alu99] R. Alur. Timed Automata. *Theoretical Computer Science* 126:183–235, 1999.

[BF02] M. Butler, J. Falampin. An Approach to Modelling and Refining Timing Properties in B. In *Refinement of Critical Systems (RCS)*. January 2002.

[BFRR10] J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, A. Roth. Patterns for Modelling Time and Consistency in Business Information Systems. In Calinescu et al. (eds.), *15th IEEE International inproceedings on Engineering of Complex Computer Systems,*

ICECCS 2010, Oxford, United Kingdom, 22-26 March 2010. Pp. 105–114. IEEE Computer Society, 2010.

[But09] M. Butler. Decomposition Structures for Event-B. In *Integrated Formal Methods IFM2009, Springer, LNCS 5423*. Volume LNCS(5423). Springer, February 2009.

[CGL97] K. Cerans, J. C. Godskesen, K. G. Larsen. Timed Modal Specification – Theory and Tools. In *IN PROC. OF THE 5TH INT. CONF. ON COMPUTER AIDED VERIFICATION, VOLUME 697 OF LECTURE NOTES IN COMPUTER SCIENCE (LNCS*. Pp. 253–267. Springer–Verlag, 1997.

[CMR07] D. Cansell, D. Méry, J. Rehm. Time Constraint Patterns for Event B Development. In Julliand (ed.), *B 2007: Formal Specification and Development in B 7th International Conference of B Users, January 17-19, 2007*. Lecture Notes in Computer Science 4355, pp. 140–154. Springer-Verlag, Besançon France, 2007. ISSN : 0302-9743 (Print) ; 1611-3349 (Online) ; ISBN : 978-3-540-68760-3.

[DHQ+08] J. S. Dong, P. Hao, S. Qin, J. Sun, W. Yi. Timed Automata Patterns. *IEEE Trans. Softw. Eng.* 34(6):844–859, 2008.

[Eme95] E. A. Emerson. Temporal and modal logic. In *HANDBOOK OF THEORETICAL COMPUTER SCIENCE*. Pp. 995–1072. Elsevier, 1995.

[ES00] N. Evans, S. Schneider. Analysing Time Dependent Security Properties in CSP Using PVS. In *ESORICS*. Pp. 222–237. 2000.

[HLM+08] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, A. Skou. Testing Real-Time Systems Using UPPAAL. 2008.

[Jac83] M. Jackson. *Michael Jackson System Development*. Englewood Cliffs, N.J. : Prentice/Hall., New York, NY, USA, 1983.

[Kop97] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[LPY01] M. Lindahl, P. Pettersson, W. Yi. Formal design and analysis of a gear controller. *STTT* 3(3):353–368, 2001.

[Sch99] S. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley Sons, Inc., New York, NY, USA, 1999.

[SPHB10] R. Silva, C. Pascal, T. S. Hoang, M. Butler. Decomposition Tool for Event-B. In *Workshop on Tool Building in Formal Methods - ABZ Conference*. January 2010.

[Yi90] W. Yi. Real-Time Behaviour of Asynchronous Agents. In *CONCUR*. Pp. 502–520. 1990.