Extending Event-B with Discrete Timing Properties

M.R.Sarshogh[a,*], M.Butler[b,**]

[a]ESS Research Group ECS University of Southampton Highfield, Southampton SO17 1BJ
[b]ESS Research Group ECS University of Southampton Highfield, Southampton SO17 1BJ

Abstract

Event-B is a formal language for systems modelling, 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, modelling 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 patterns, *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 have been given a semantic in terms of the standard Event-B constructs. To ease the process of using timing properties 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.

Keywords:
Real-time System, Event-B, Event, Deadline, Delay, Expiry, Refinement Patterns, Parametrized Events, Periodic Events, Decomposition, Timing Properties

*Corresponding author
**Principal corresponding author
Email addresses: `mrs2g09@ecs.soton.ac.uk` (M.R.Sarshogh), `mjb@ecs.soton.ac.uk` (M.Butler)

1. Introduction

In Event-B [1], systems are modelled 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 patterns in control systems. These patterns are common and useful in specification of control systems. It is natural to talk about these kinds of systems in terms of possible events of the system. For example in the trigger-response patterns, trigger and response are both events of a control system.

One of the main advantages of Event-B method is its support for stepwise modelling by refinement. The other strength of this method is the mechanized proof obligation generator and the prover which makes the verification process, efficient and productive [1]. These advantages of Event-B, make it a suitable approach for formal modelling of critical systems. Besides having decomposition features is greatly beneficial in modelling large and complex systems, by dividing the model to sub-models which can be refined independently.

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 behaviour and its properties can be modelled 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 behaviours. 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 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 abstract levels.

Event-B lacks explicit support for expressing and verifying timing properties. In a time-critical system, timing properties specify timing boundaries on the system reactions and responses. Modelling time-critical systems, using Event-B has been investigated in several studies. The contribution of this paper is categorizing discrete timing properties in three groups: deadline, delay and expiry. Beside, augment Event-B with language constructs for these property groups, introducing patterns for refining timing properties and proving satisfaction of abstract timing property by their concrete ones. Having a semantics based on the standard Event-B constructs for timing properties, helped us to benefit from the existing features of Event-B such as refinement and decomposition, with no change or adaptation required.

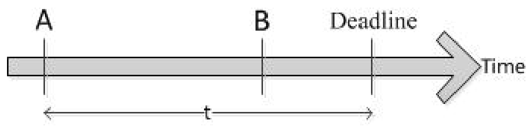
Event-B refinement allows atomic events at the abstract level to be broken down into sub-steps at the refined level. The goal of our refinement patterns is to provide an easy way to represent and correctly refine timing properties on abstract atomic events with more elaborate timing properties on the refined events.

In any system analysis, there always exists a level of granularity which will not be broken to a finer one. In a time-critical system, there are assumptions about the maximum duration required to establish each step at concrete level such as individual assignments or signal transmissions. Based on those assumptions it is possible to analyse the required time for a composite process to respond to a request or react to a change in that system. In our approach there is a top-bottom analysis. So, we start from the abstract behaviours of a system and model their properties and timing assumptions, then by each refinement, we introduce the sub-steps of the abstract behaviours and replace their timing assumptions with the timing assumptions of the sub-steps. As a result, by verifying the consistency of the refinement, we prove the consistency of the timing assumptions in different levels of abstraction. Hence, when the modeller verifies the consistency of the last refinement, he/she has verified the satisfaction of the abstract timing properties based on the timing assumptions of the most concrete behaviours.

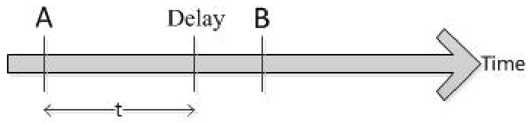
This paper presents an initial set of patterns. Then we introduced several patterns to refine some of the timing properties, and why we think they are beneficial. In the end to demonstrate the practicality of our approaches, how they have been used to model and verify an automatic gear controlling system and a message passing system, will be discussed.

2. Timing Properties Categories

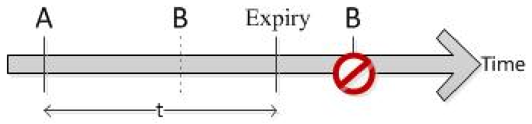
In order to explicitly represent timing properties we extend the Event-B syntax with constructs for deadlines, delays and expiries. Each of these timing properties places a discrete timing property between trigger events and their response events. A typical pattern is a trigger followed by one of several possible responses, thus each of our timing constructs specifies a constraint between a trigger event A and either a response event B or a set of



(a) Timing diagram of Deadline property.



(b) Timing diagram of Delay property.



(c) Timing diagram of Expiry property. Event B may only occur before the expiry duration (t). That is why it is represented by a dash line.

Figure 1: In these diagrams, t is the timing property duration, A is the trigger event and B is its response event and the horizontal axis is the time line.

response events $B_1..B_n$. The syntax for each of these properties is as follow:

$$\textbf{Deadline}(A, B_1 \vee .. \vee B_n, t)(Figure\ 1a), \tag{1}$$
$$\textbf{Delay}(A, B, t), (Figure\ 1b), \tag{2}$$
$$\textbf{Expiry}(A, B, t).(Figure\ 1c). \tag{3}$$

Deadline$(A, B_1 \vee .. \vee B_n, t)$ (1) means that one of the response events $(B_1..B_n)$ must occur within time t of trigger event (A) occurrence. We use a disjunction symbol between the possible responses, to indicate the alternative nature of the property, but we treat the possible responses as a set. In the case of delay (2), the response event can only happen if the delay period has passed following an occurrence of the trigger event. Finally the expiry (3) means that the response event cannot happen if the expiry period has been passed following an occurrence of the trigger event.

Beside systematising the process of specifying discrete timing properties in Event-B models, having these annotations hides the complexity of encoding timing properties in an Event-B model from the modeller. As a result, a timed-Event-B machine from modeller point of view, will be a non-timed-Event-B machine plus a list of its timing properties, declared by the introduced timing properties.

In the following sections, the semantic of these timing extensions of Event-B will be discussed.

3. Semantics of Timing Properties In Event-B

We give a semantics to our timing constructs by translating them into Event-B variables, invariants, guards and actions. In particular, these timed-Event-B elements constrain the order between trigger event, response events and the time-progression event ($Tick\_Tock$).

In each case we assume there is already a partial order between the trigger event and the corresponding response events. The assumption is that the response events are only enabled if the trigger event has already happened. This ordering assumption for a sequential control flow, is encoded by using boolean flags for unparametrized events. As shown in Figure 2a, event A sets the boolean variable A as one of its actions, so when variable A has the value of $TRUE$, this indicates that event A has already happened. Also, one of the response events' guards checks the occurrence of trigger event A, by checking its occurrence flag.

It has not been assumed 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 by an event.

3.1. Delay Semantics

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 value in the machine and an event is added to model the progress of time.

In order to explain the semantics of delay in Event-B, we assume a generic trigger event A and a generic response event B. In the Patterns of this section, let $G_X(c, v)$ be the set of guards and Act_X be the actions of event X, where c denotes the constants and v the variables.

```
TIMING
    $Delay(A, B, t)$
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Event A and B plus a delay

```
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $tA := time$
        $Act_A$
    END
EVENT $B \widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $time \geq tA + t$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END

EVENT $Tick\_Tock \widehat{=}$
    any
        $tick$
    WHERE
        $tick > 0$
    THEN
        $time := time + tick$
    END
```

(b) Encoded delay for events A and B.

Figure 2: Semantic of a delay property in Event-B.

A delay property is structured as follows:

$$Delay(A, B, t). \tag{4}$$

There are two parts to the Event-B semantics of a delay property. First the occurrence time of the trigger event is recorded in a variable (tA). Second, in the response event (B), a guard is needed which prevents the trigger event from being enabled before the stated delay duration has passed from the occurrence of the trigger event. In Figure 2 a general pattern of delayed trigger-response in an Event-B model, plus the $Tick\_Tock$ event has been shown.

Figure 2a shows the generic trigger-response pattern plus a delay property, and Figure 2b shows how the delay is represented in terms of standard Event-B elements.

As shown in Figure 2b, the $Tick\_Tock$ event is a part of the semantics of timing properties which models the progress of time based on the *tick* duration.

3.2. Expiry Semantics

Expiry is given an Event-B semantics similar to modelling delay, guarding the response events according to the recorded occurrence time of the trigger event and the specified expiry period. In order to explain how expiry is represented in Event-B, we assume a generic trigger event A and a generic response event B with an expiry as shown in Figure 3a.

```
TIMING
    $Expiry(A, B, t)$
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Events A and B plus an expiry

```
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $tA := time$
        $Act_A$
    END
EVENT $B \widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $time \leq tA + t$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END

EVENT $Tick\_Tock \widehat{=}$
    any
        $tick$
    WHERE
        $tick > 0$
    THEN
        $time := time + tick$
    END
```

(b) Encoded expiry for events A and B.

Figure 3: Semantics of an expiry property in Event-B.

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

3.3. Deadline Semantics

As explained in the introduction section, in time-critical systems, there are assumptions about maximum duration required to stablish different processes. These assumption can be modelled by deadline in our approach. In order to give Event-B semantics to expiry and delay, just the trigger and the response events are guarded. However, based on the deadline semantics, the *Tick_Tock* event is guarded instead. If the trigger event has happened, we want to force the response event to occur, before passing the deadline. Guarding the *Tick_Tock* event is a way of enforcing one of the $B_1..B_n$ events to occur before passing the deadline. So, by guarding the *Tick_Tock*, the upper bounds assumption can be enforced to a model.

A deadline property is structured as follow:

$$Deadline(A, B_1 \vee .. \vee B_n, t). \tag{5}$$

The guard on the $Tick\_Tock$ event to enforce deadline (5), prevents reaching the deadline ($time+1 \leq tA+t$) if trigger event A has occurred but a response event B_x has yet to occur (Figure 4b).

```
TIMING
    $Deadline(A, B_1 \vee .. \vee B_n, t)$
EVENT $A \mathrel{\widehat{=}}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B_x \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_x = FALSE$
        $G_{B_x}(c, v)$
    THEN
        $B_x := TRUE$
        $Act_{B_x}$
    END
```

(a) Event A and B_x plus a deadline

```
EVENT $A \mathrel{\widehat{=}}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $tA := time$
        $Act_A$
    END
EVENT $B_x \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_x = FALSE$
        $G_{B_x}(c, v)$
    THEN
        $B_x := TRUE$
        $Act_{B_x}$
    END

EVENT $Tick\_Tock \mathrel{\widehat{=}}$
    any
        $tick$
    WHERE
        $tick > 0$

        $A = TRUE \wedge$
        $(B_1 = FALSE \wedge .. \wedge$
        $B_n = FALSE) \Rightarrow$
        $time + tick \leq tA + t$
    THEN
        $time := time + tick$
    END
```

(b) Encoded deadline for Event A and B_x.

Figure 4: Semantics of a deadline property in Event-B.

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

4. Parametrized Timing Properties

Events may have parameters in Event-B. If either of the trigger or response events are parametrized, we need to specify occurrence of the parametrized event for which parameter is the subject of the property. The syntax for each of these properties for parametrized trigger and response events is as follows:

$$\forall\, a \cdot P_a \;\mid\; \textbf{Deadline}\ (A(a), B_1(f_{B_1}(a)) \vee \cdots \vee B_n(f_{B_n}(a)), t), \tag{6}$$

$$\forall\ a \cdot P_a\ \mid \textbf{Delay}\ (A(a), B(f_B(a)), t), \tag{7}$$

$$\forall\ a \cdot P_a\ \mid \textbf{Expiry}\ (A(a), B(f_B(a)), t). \tag{8}$$

Deadline (6) means that within a duration t time units from occurrence of event A for a parameter a, which satisfies predicate P_a, one of the response events B_x ($x \in 1..n$) has to happen for a parameter $f_{B_x}(a)$. $f_{B_x}(a)$ is an injection from the parameters of the trigger event to the parameters of a response event B_x. This relation is based on the exciting order between trigger event and its responses. Predicate P_a specifies the range of parameters, the timing property is based on. It is also possible that we have a mixture of parametrized and unparametrized events in the response set, In the case of unparametrized trigger event and parametrized response events, The timing constraint will be based on the response events' parameters. The other way around will be similar as presented in (6).

In the case of delay (7), it means that from occurrence of the trigger event (A) for a parameter a, which satisfies predicate P_a, response event B can only happen for a parameter $f_B(a)$, if t time units have already passed. Same as the deadline $f_B(a)$ is an injection from trigger event's parameters to its response's parameters based on their order. Finally, expiry (8) means that the response event cannot happen for a parameter $f_B(a)$ after the expiry period has been passed following an occurrence of trigger event A for a parameter a, if a captures predicate P_a.

In the case of parametrized trigger-response events, in order to model the order between a trigger event occurrence and its possible responses, a set of occurrence values is used. So when a trigger event A happens for a parameter a, the parameter will be added to a set A and that set is used to enable the response events. Same as the unparametrized order, it has not been assumed that the trigger and response events will occur only once for each parameter. So, by the end of each iteration, the occurrence sets will be emptied by an event.

By using parametrized timing properties, it is possible to model iterative events, such as sending a message part by part, or synchronize several iterative events such as sending and receiving events in a message passing system.

4.1. Semantics of Parametrized Delay and Expiry

To give an Event-B semantics to a delay on a parametrized trigger-response pattern, the variable which records the trigger event occurrence time is a function from the event's parameter to its occurrence time, and trigger and response events each have a occurrence set, which are used to order their occurrence for different parameters. A generic parametrized trigger-response pattern with a delay property is shown in Figure 5a and Figure 5b presents how the delay property is enforced by using standard Event-B construct.

```
TIMING
    $\forall a \cdot P_a \mid$
    $Delay(A(a), B(f_B(a)), t)$
EVENT $A \mathrel{\widehat{=}}$
    any
        $a$
    WHERE
        $a \in X$
        $a \notin A$
        $G_A(c, v, a)$
    THEN
        $A := A \cup \{a\}$
        $Act_A$
    END
EVENT $B \mathrel{\widehat{=}}$
    any
        $a, b$
    WHERE
        $a \in A$
        $b = f_B(a)$
        $b \notin B$
        $G_B(c, v, b)$
    THEN
        $B := B \cup \{b\}$
        $Act_B$
    END
```

(a) Event A and B plus a delay

```
EVENT $A \mathrel{\widehat{=}}$
    any
        $a$
    WHERE
        $a \in X$
        $a \notin A$
        $G_A(c, v, a)$
    THEN
        $A := A \cup \{a\}$
        $tA(a) := time$
        $Act_A$
    END
EVENT $B \mathrel{\widehat{=}}$
    any
        $a$
        $b$
    WHERE
        $a \in A$
        $b = f_B(a)$
        $b \notin B$
        $P_a \Rightarrow time \geq tA(a)+t$
        $G_B(c, v, b)$
    THEN
        $B := B \cup \{b\}$
        $Act_B$
    END

EVENT $Tick\_Tock \mathrel{\widehat{=}}$
    any
        $tick$
    WHERE
        $tick > 0$
    THEN
        $time := time + tick$
    END
```

(b) Encoded delay for events A and B.

Figure 5: Semantics of a parametrized delay property in Event-B.

In Figure 5, X specifies the range of the parameters of A, and P_a specifies the range of its parameters, the timing property is concerned with. As can be seen from Figure 5b, event B can only occur for a parameter $f_B(a)$, if event A has already happened for a parameter a.

$$\forall\ a\ \cdot\ P_a\ \mid\ Expiry(A(a), B(f_B(a)), t) \tag{9}$$

Semantics of the expiry is similar to the delay semantics. On a same generic trigger-response example as shown in Figure 5a, if we want to have expiry (9) instead of the delay, the timing guard in the response event, in Figure 5b, will be changed to the following predicate:

$$P_a \Rightarrow time \leq tA(a) + t \tag{10}$$

By having guard (10), if the trigger parameter satisfies predicate P_a, then the response event is only enabled, if the current time is still within the expiry duration.

4.2. Semantics of Parametrized Deadline

To explain the semantics of deadlines on the parametrized trigger-response pattern, a generic example with n alternative responses will be explained in the following.

```
TIMING
    $\forall a \cdot P_a \mid$
    $Deadline(A, B_1(f_{B_1}(a)) \vee .. \vee$
    $B_n(f_{B_n}(a)), t)$
EVENT $A \;\widehat{=}$
    any
        $a$
    WHERE
        $a \in X$
        $a \notin A$
        $G_A(c, v, a)$
    THEN
        $A := A \cup \{a\}$
        $Act_A$
    END
EVENT $B_x \;\widehat{=}$
    any
        $a, b$
    WHERE
        $a \in A$
        $b = f_{B_x}(a)$
        $b \notin B_x$
        $G_{B_x}(c, v, b)$
    THEN
        $B_x := B_x \cup \{b\}$
        $Act_{B_x}$
    END
```

(a) Event A and B_x ($x \in 1..n$) plus a deadline. X is the range of events A parameters.

```
EVENT $A \;\widehat{=}$
    any
        $a$
    WHERE
        $a \in X$
        $a \notin A$
        $G_A(c, v, a)$
    THEN
        $A := A \cup \{a\}$
        $Act_A$
    END
EVENT $B_x \;\widehat{=}$
    any
        $a, b$
    WHERE
        $a \in A$
        $b = f_{B_x}(a)$
        $b \notin B_x$
        $G_{B_x}(c, v, b)$
    THEN
        $B_x := B_x \cup \{b\}$
        $Act_{B_x}$
    END

EVENT $Tick\_Tock \;\widehat{=}$
    any
        $tick$
    WHERE
        $tick > 0$

        $\forall\ a\ \cdot P_a \wedge a \in A \wedge$
        $f_{B_1}(a) \notin B_1 \wedge .. \wedge$
        $f_{B_n}(a) \notin B_n \Rightarrow$
        $time + tick \leq tA(a) + t$
    THEN
        $time := time + tick$
    END
```

(b) Encoded deadline for Event A and B_x.

Figure 6: Pattern for encoding a deadline property in Event-B.

As shown in Figure 6a, a response event B_x can only happen for a parameter $f_{B_x}(a)$, if A has already happened for parameter a. Deadline is based on occurrence of the trigger event (A) for a parameter a , and occurrence of at least one of the response events such as event B_x, for parameter $f_{B_x}(a)$, before reaching the deadline. The guard on the *Tick_Tock* event allows a progress of time for a duration which does not violate the deadline.

5. Some Patterns to Refine Deadline, Delay and Expiry

In this section, some patterns to refine an abstract deadline to more detailed timing properties will be explained. It should be mentioned that these are not modelling patterns, rather they are refinement patterns; the aim of our patterns is to explain how timing properties can be refined based on some specific control flow refinement patterns. In each of the following sections, the corresponding control flow refinements which make the timing refinement necessary will be explained and then the timing refinement will be discussed. Besides, invariants required to discharged refinement proof obligations of the timing properties will be discussed for each pattern. Each refinement pattern will be explained by applying it to a generic control flow refinement pattern. In the rest of this report timed-Event-B models will be shown from a modeller point of view. So, each timed-machine will be a list of its timing properties specified by introduced constructs in Section 3 plus the non-timed Event-B machine. In each refinement pattern, for the constants c and set of variables v of a machine, $G_X(c, v)$ presents the guards of event X in that machine and Act_X presents the actions of that event.

Besides, in this section, refinement diagrams [2] have been used to present the relations of events in different levels of abstraction. Refinement diagrams are tree form structures, in which the abstract events are placed as the root on the top, and their concrete events are presented as their leaves on the bottom of the structure. Those events that refine the abstract events are connected to their abstract events by solid lines, whereas the new events which refine skip and represent the required pre/post steps of the abstract events, in a more concrete behaviour of a system, are connected with dash lines to their corresponding abstract events. Besides, in each level, the order of execution is from left to right.

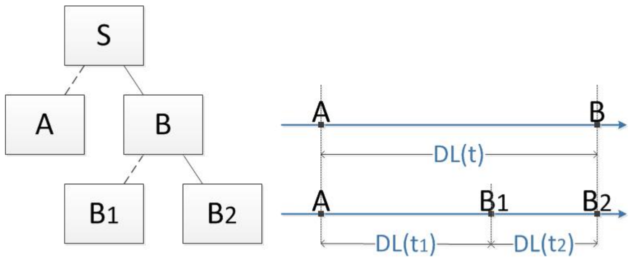


Figure 7: Refining an abstract deadline to two sub-deadlines is presented by the refinement diagram on the left. $DL(x)$ presents a deadline property with a period of x in the timing diagrams.

5.1. 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 property of this level of abstraction, is shown in Figure 8a. In the next refinement, event B has been broken to two sequential steps, as shown in Figure 7. By breaking event B to B_1 followed by B_2, its related deadline needs to be broken too. The other important issue in this pattern is that, the abstract event has been refined by the second step, because the accomplishment of the second step is equivalent to accomplishment of the abstract event(B). So the first step should refine *skip*. Now, in order to respond to the trigger event, two steps have to be accomplished, where each of them has its own deadline. In the concrete level, the trigger event of the deadline property for event B_1 is event A and the trigger event for the deadline of event B_2 is event B_1. Hence, the abstract deadline should be broken to two new deadlines as shown in Figure 8b, in a way that their combination based on the control flow does not violate the abstract deadline ($t_1 + t_2 \leq t$).

```
TIMING
    $Deadline(A, B, t)$
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Event A and B

```
TIMING
    $Deadline(A, B_1, t_1)$
    $Deadline(B_1, B_2, t_2)$
EVENT $A \widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B_1 \widehat{=}$
    WHERE
        $A = TRUE$
        $B_1 = FALSE$
        $G_{B_1}(c, v)$
    THEN
        $B_1 := TRUE$
        $Act_{B_1}$
    END

EVENT $B_2$ refines $B \widehat{=}$
    WHERE
        $B_1 = TRUE$
        $B_2 = FALSE$
        $G_{B_2}(c, v)$
    THEN
        $B_2 := TRUE$
        $Act_{B_2}$
    END
```

(b) Event B_1 and B_2

Figure 8: Events A and B plus their deadline property in the abstract Machine in 8a followed by event A , events B_1 and B_2 in the concrete machine plus their concrete timing properties in 8b.

We need to prove that the concrete machine refines the behaviour of its abstract one. For the refinement pattern, presented in Figure 7, it is necessary to prove the abstract deadline holds in the concrete machine.

As shown in Figure 8, in the concrete machine, the abstract deadline between event A and event B is refined by the following deadlines:

$$Deadline\ (A,\ B_1,\ t_1), \tag{11}$$

$$Deadline\ (B_1,\ B_2,\ t_2). \tag{12}$$

Based on deadline (12) if event B_1 has happened and event B_2 has not happened yet, then the current value of time should be less than or equal to the occurrence time of event B_1 plus the deadline period (t_1). By having this timing property the relation between occurrence time of event B_2 and event B_1 has been specified. But we are interested on the relation of the occurrence time of event B_2 and event A. So it is enough to specify the relation of the occurrence time of event B_1 and event A.

The relation between the concrete states and the abstract ones is expressed by *gluing invariants* [1] 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 abstract. The first type is needed to clarify the relation between the order in the abstract machine and the order in the concrete machine, which has not been modelled by explicit guards. For example based on the guards in the presented refinement pattern (Figure 8), B_2 can only happen after B_1 occurrence and B_1 can only happen after A occurrence, accordingly B_2 can only happen after A, but it has not been mentioned explicitly in the guards of B_2. The other type of gluing invariants is needed to specify the relation between the new deadlines in the concrete machine and the abstract deadline. In the refinement pattern presented in Figure 8, these invariants should be as follow:

- The relation between abstract event and its refining event (B_2 and B are the boolean variables which act as the occurrence flags of events B_2 and B):

$$B_2 = B, \tag{13}$$

- The order between concrete events:

$$B_1 = TRUE \Rightarrow A = TRUE, \tag{14}$$

- 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 tB_1 does the same thing for event B_1):

$$B_1 = TRUE \Rightarrow tB_1 \leq tA + t_1, \tag{15}$$
$$A = TRUE \wedge B_1 = FALSE \Rightarrow time \leq tA + t_1. \tag{16}$$

Invariant (13) specifies that the occurrence of event B_2 is equivalent to the occurrence of event B.

The relation of the occurrence time of event B_1 and event A has been specified by the gluing invariant (15) based on deadline (11), since in event B_1 there is no information about the occurrence time of event A. Based on invariant (15) we know that if B_1 has occurred, then the duration between A and B_1 does not exceed t_1.

Invariant (16) which is equivalent to the required guard on the $Tick\_Tock$ event for deadline (11) provides the required information to discharge the proof obligation of invariant (15) for event B_1.

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, and for their deadline refinement it is possible to follow a similar approach.

Refining a parametrized abstract deadline to sequential sub-deadlines can be done in a same way, the only difference will be in the occurrence flags which will be replaced by occurrence sets and occurrence times will be a relation from a parameters to their occurrence time.

5.2. Refining An Abstract Deadline to An Iterative Sub-Deadline

Sometimes there is an iterative event in a system which happens for a finite number of times and complete a task gradually. It usually causes the appearance of some properties of the overall effect of the iterative event and some properties of each occurrence of that event in the system specifications. For example, transferring a message part by part in a system, may have some timing properties on the overall transferring process and some timing properties on transferring each part.

In a stepwise modelling and reasoning manner, one possible way of dealing with these types of behaviour is to abstract it by two events, start and end, and prove the properties of the overall effect for them, and then refine them to an iterative event and verify the properties of each occurrence for the iterative event.

From timing point of view, we will be able to verify the consistency of each occurrence's deadline with the overall deadline of the task accomplishment. For example, consider a case where in the abstraction we have a trigger event A and its response event B which has to happen within a duration of t time units from occurrence of A. Then, a new iterative event B_1 which happens n times will be added which represents the required pre-steps of event B in the refinement and the response event will be refined based on the new order by event B_2 (Figure 9). In the refinement diagram presented in Figure 9, n in the circle represents the fact that event B_1 has to happen n times before event B can happen.

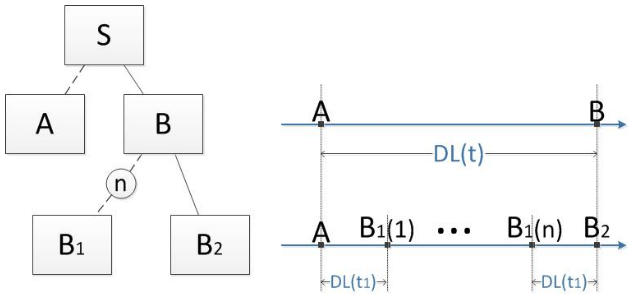


Figure 9: Refining a deadline to an iterative deadline.

So, the abstract deadline will be refined by a deadline of t_1 time units between the trigger event and the first occurrence of the iterative event, a deadline with a same duration between each occurrence of the iterative event and the next one, and a deadline with t_1 duration between the last occurrence of the iterative event and occurrence of event B_2 where $t = (n + 1) * t_1$ (Figure 10) .

```
TIMING
    $Deadline(A, B, t)$
EVENT $A \mathrel{\hat=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \mathrel{\hat=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Event A and B

```
TIMING
    $\forall b \cdot b = 1 \mid Deadline(A, B_1(b), t_1)$
    $\forall b \cdot b \in 1..n-1$
     $\mid Deadline(B_1(b), B_1(b+1), t_1)$
    $\forall b \cdot b = n \mid Deadline(B_1(b), B_2, t_1)$
EVENT $A$ $extends$ $A$
EVENT $B_1 \mathrel{\hat=}$
    any
        $b$
    WHERE
        $b \in 1..n$
        $b = 1 \Rightarrow A = TRUE$
        $b > 1 \Rightarrow b - 1 \in B_1$
        $b \notin B_1$
        $G_{B_1}(c, v, b)$
    THEN
        $B_1 := B_1 \cup b$
        $tB_1(b) := time$
        $Act_{B_1}$
    END

EVENT $B_2$ $refines$ $B \mathrel{\hat=}$
    WHERE
        $n \in B_1$
        $B_2 = FALSE$
        $G_{B_2}(c, v)$
    THEN
        $B_2 := TRUE$
        $Act_{B_2}$
    END
```

(b) Event B_1 and B_2

Figure 10: Events A and B plus their deadline property in the abstract Machine in 10a followed by events A , events B_1 and B_2 in the concrete machine with their concrete timing properties in 10b.

As shown in Figure 10b, in the first deadline, the only parametrized event is the response event, so the parameter in the deadline property will be based on the parameter of the response event, despite the parametrized deadline syntax introduced in Section 4. The syntax introduced in that section is based on the cases where both groups of the trigger and response events are parametrized.

To prove the consistency of the timing properties in these two levels of abstraction, we need to show that the overall deadline of the iterative event is $(n-1) * t_1$. To prove this, we need to prove the following invariants for the occurrence of event B_1 for a parameter $b \in 1..n$:

$$\forall\, b \cdot b \;\in\; B_1 \;\wedge\; b \;>\; 1 \;\Rightarrow\; b-1 \;\in\; B_1 \tag{17}$$

$$\forall\, b \cdot b \;\in\; B_1 \;\Rightarrow\; 1..b \;\subseteq\; B_1 \tag{18}$$

Invariants (17) and (18) show the sequential order of the iterative occurrences. So each step can happen if the previous ones have already happened. By having these, now we can talk about the effect of sequential order on the timing properties.

$$\forall\, b \cdot b \;< n \;\wedge\; b \;\in\; B_1 \;\wedge\; b+1 \notin\; B_1 \Rightarrow\; time \leq tB_1(b) + t_1 \tag{19}$$

$$\forall\, b \cdot b \;\geq\; 1 \;\wedge\; b \;\in\; B_1 \;\Rightarrow\; tB_1(b) \leq tB_1(1) + ((b-1) * t_1) \tag{20}$$

Invariant (19) is the iterative deadline and invarinat (20) is what can be proved based on invarinats (19) and (18). By having invariant (20) we can prove that the concrete deadlines are consistent with their abstract one by breaking possible cases in three groups as follow:

1. A happened but B_1 has not happened yet,
2. B_1 happened for a parameter $x < n$ but has not happened for $x+1$,
3. B_1 happened for n but B_2 has not yet happened.

Invariant (20) is required for case 2 and 3. So, by using this approach, it is possible to model an iterative event and reason about its timing properties.

5.3. Refining An Abstract Deadline to Alternative Sub-deadlines Case One

It is often the case in a system specification, that we require a response to a request within a specific period of time. This response may be satisfaction of the request or an error in the case that the request cannot be met. This kind of behaviour can be modelled by a deadline between the request and the possible response scenarios.

For instance, consider the case where instead of refining event B, in the refinement pattern of Section 5.1, by two sequential sub-steps, it has been refined by two alternative events, B_1 and B_2 as shown in Figure 11, where event B_1 represents the main response scenario and event B_2 represents the alternative one.

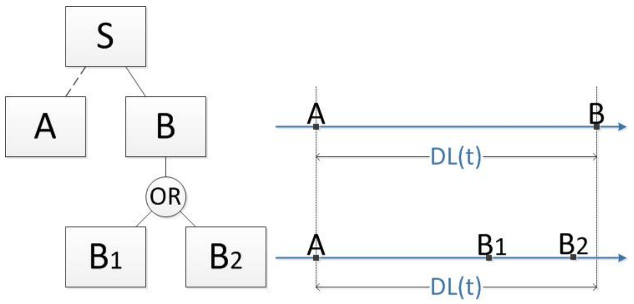


Figure 11: Refining an event to two possible events. OR in the refinement diagram represents the fact that either of B_1 occurrence or B_2 occurrence in the refinement is equivalent to the occurrence of event B in the abstract.

Based on the refinement, either of event B_1 occurrence or event B_2 occurrence is equivalent to occurrence of the abstract event (B). So the abstract deadline between event A and event B will be satisfied by occurrence of either of the refining events before the deadline. As shown in Figure 12b, the concrete deadline is between occurrence of event A and either the occurrence of event B_1 or event B_2, and its duration is the same as the abstract deadline.

```
TIMING
    $Deadline(A, B, t)$
EVENT $A \hat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \hat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Events A and B and their timing property.

```
TIMING
    $Deadline(A, B_1 \vee B_2, t)$
EVENT $A \hat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B_1$ refines $B \hat{=}$
    WHERE
        $A = TRUE$
        $B_1 = FALSE$
        $G_{B_1}(c, v)$
    THEN
        $B_1 := TRUE$
        $Act_{B_1}$
    END
EVENT $B_2$ refines $B \hat{=}$
    WHERE
        $A = TRUE$
        $B_2 = FALSE$
        $G_{B_2}(c, v)$
    THEN
        $B_2 := TRUE$
        $Act_{B_2}$
    END
```

(b) Events A, B_1 and B_2 and their timing property

Figure 12: Refining a trigger-response pattern and its timing properties to two alternative responses plus the concrete timing property.

The only kind of invariant required to discharge refinement proof obligations of the timing property in this pattern, specifies the relation between the occurrences of the abstract and the concrete events. In the this generic refinement pattern this invariant will be as follow:

$$B_1 = TRUE \vee B_2 = TRUE \Leftrightarrow B = TRUE. \tag{21}$$

Based on invariant (21) either of event B_1 or event B_2 occurrence, is equivalent to the occurrence of event B. Same as the previous refinement pattern, a parametrized abstract deadline can be refined to two alternative parametrized sub-deadline in a same way by changing the occurrence flags and the occurrence time variables based on the parametrized timing properties semantics.

5.4. Refining An Abstract Deadline to Alternative Sub-deadlines Case Two

Sometimes a request-response sequence in the abstract machine should be refined by several alternative cases of request-response. For example, consider the case where in the abstract model, the car direction is given by the driver and the car will go to that direction. Then it will be refined

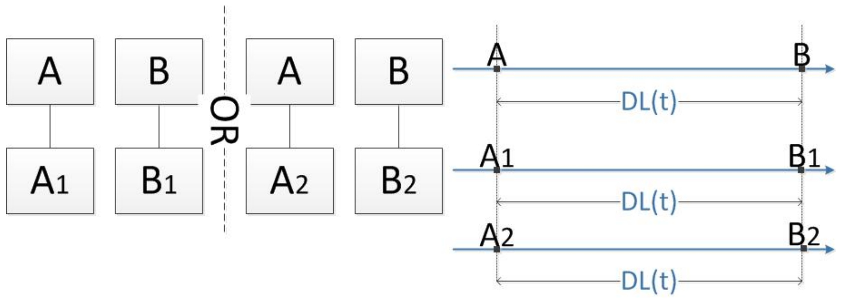


Figure 13: How a single request-response sequence can be refined to several request-response cases. OR in the refinement diagram represents the fact that occurrence of either of the two sequences in the concrete level is equivalent to the occurrence of the abstract sequence.

by two possible cases where either driver sets the direction to the left and car goes to the left, or driver sets the direction to the right and car goes to the right. The difference between this case and the refinement pattern, explained in Section 5.3, is just about the trigger event. In Section 5.3 the trigger event has not been refined, but in this case the trigger event has been refined to several alternative cases and each concrete trigger event is related to one of the alternative responses. Consider the generic refinement pattern of Section 5.3 where event A is refined by two alternative events A_1 and A_2 as shown in Figure 13. In this refinement, event A_1 triggers event B_1 and event A_2 triggers event B_2. As a result, the abstract deadline between event A and event B will be refined as shown in Figure 14b.

How the partial order between each response and its corresponding request is enforced, using boolean flags, is shown in Figure 14b.

```
TIMING
    $Deadline(A, B, t)$
EVENT $A \;\widehat{=}$
    WHERE
        $A = FALSE$
        $G_A(c, v)$
    THEN
        $A := TRUE$
        $Act_A$
    END
EVENT $B \;\widehat{=}$
    WHERE
        $A = TRUE$
        $B = FALSE$
        $G_B(c, v)$
    THEN
        $B := TRUE$
        $Act_B$
    END
```

(a) Events A and B and their timing property.

```
TIMING
    $Deadline(A_1, B_1, t)$
    $Deadline(A_2, B_2, t)$
EVENT $A_1$ $refines$ $A \widehat{=}$
    WHERE
        $A_1 = FALSE$
        $G_{A_1}(c, v)$
    THEN
        $A_1 := TRUE$
        $Act_{A_1}$
    END
EVENT $A_2$ $refines$ $A \widehat{=}$
    WHERE
        $A_2 = FALSE$
        $G_{A_2}(c, v)$
    THEN
        $A_2 := TRUE$
        $Act_{A_2}$
    END
EVENT $B_1$ $refines$ $B \;\widehat{=}$
    WHERE
        $A_1 = TRUE$
        $B_1 = FALSE$
        $G_{B_1}(c, v)$
    THEN
        $B_1 := TRUE$
        $Act_{B_1}$
    END
EVENT $B_2$ $refines$ $B \;\widehat{=}$
    WHERE
        $A_2 = TRUE$
        $B_2 = FALSE$
        $G_{B_2}(c, v)$
    THEN
        $B_2 := TRUE$
        $Act_{B_2}$
    END
```

(b) Events A_1, A_2, B_1 and B_2 and their timing properties

Figure 14: Refining a trigger-response pattern and its timing property to two alternative request-response cases and their concrete timing properties.

The only difference between the concrete deadlines and the abstract one, is the name of trigger and response events. For each case of request-response the corresponding request event and response event will be constrained by a deadline with a same duration as the abstract one. To discharge the refinement proof obligations of the presented timing properties, a gluing invariant is required for each concrete sequence of request-response, to specify the relation between occurrences of the abstract events and their concrete ones. Based on our generic refinement pattern the required invariants should be as follow:

$$A_1 = TRUE \lor A_2 = TRUE \Leftrightarrow A = TRUE \tag{22}$$

$$B_1 = TRUE \lor B_2 = TRUE \Leftrightarrow B = TRUE \tag{23}$$

A parametrized abstract deadline can be refined to alternative parametrized sub-deadlines by using the same approach and changing the invariants according to the semantic of parametrized timing properties.

5.5. *Refining Alternative Sub-Deadlines by Sequential Sub-Deadlines and Expiries*

Developing this pattern was triggered when we were working on an automatic gear controlling case study. Based on the case study, there were two conditions assumed for the system, normal and difficult. After receiving the request, controller tries to stablish a synchronized speed between engine and wheels in order to release the current gear. But in a difficult situation, it is not possible to accomplish the first step in time, so after struggling to archived the synchronized speed for some times, the controller will try to open the clutch and then release the current gear by an open clutch.

In order to explain this pattern, the refinement pattern of Section 5.3 will be continued. In the current state, we have a trigger event A and two alternative responses, events B_1 and B_2. The timing properties of these levels of abstraction are shown in Figure 12.

In the next refinement, each of event B_1 and event B_2 will be refined to two sequential steps and their deadline will be refined to two sequential deadlines, same as the pattern presented in Section 5.1 (event B_1 will be broken to events B_3 and B_4 and event B_2 will be broken to events B_5 and B_6).

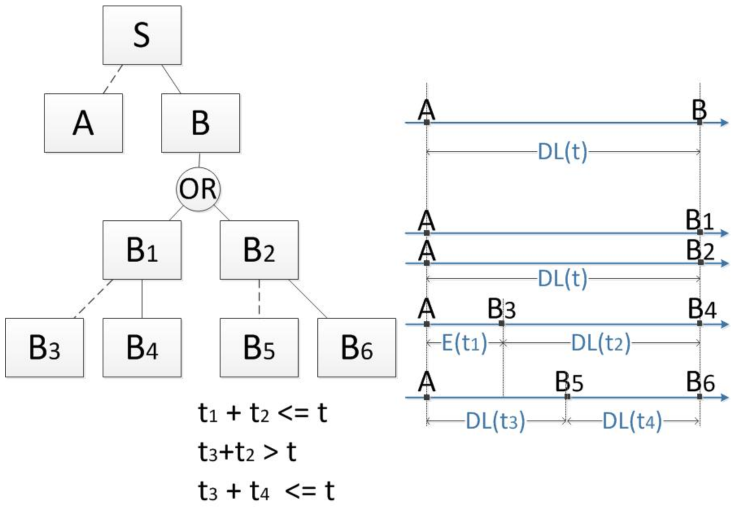


Figure 15: Refining each of the alternative scenarios to two sequential steps.

In this refinement, one of the two high level responses (B_1 or B_2) will occur as follow:

```
TIMING
    $Expiry(A, B_3, t_1)$
    $Deadline(B_3, B_4, t_2)$
    $Deadline(A, B_3 \vee B_5, t_3)$
    $Deadline(B_5, B_6, t4)$
EVENT $B_3 \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_3 = FALSE$
        $G_{B_3}(c, v)$
    THEN
        $B_3 := TRUE$
        $Act_{B_3}$
    END
EVENT $B_4$ $refines$ $B_1 \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_3 = TRUE$
        $B_4 = FALSE$
        $G_{B_4}(c, v)$
    THEN
        $B_4 := TRUE$
        $Act_{B_4}$
    END
EVENT $B_5 \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_5 = FALSE$
        $G_{B_5}(c, v)$
    THEN
        $B_5 := TRUE$
        $Act_{B_5}$
    END
EVENT $B_6$ $refines$ $B_2 \mathrel{\widehat{=}}$
    WHERE
        $A = TRUE$
        $B_5 = TRUE$
        $B_6 = FALSE$
        $G_{B_6}(c, v)$
    THEN
        $B_6 := TRUE$
        $Act_{B_6}$
    END
```

Figure 16: Events B_3, B_4, B_5 and B_6 of the final refinement and their timing properties.

- The first step of B_1, represented by B_3, must occur within time t_1 of A and then its second step, B_4 occurs within time t_2 of B_3,

- B_3 does not occur within time t_1 of A, instead the first step of B_2, represented by B_5, must occur within time t_3 of A and the second step, B_6, must occur within time t_4 of B_5.

As a result by the deadline t_3 of A in the concrete machine, either the first response case has been activated or the second one (by occurrence of their first steps).

It is assumed that the sum of the concrete deadline between event A and event B_3 (t_3), followed by the concrete deadline between event B_3 and B_4 (t_2), is not within the abstract deadline between event A and event B ($t_3 + t_2 > t$).

As mentioned above, event B_3 has an expiry property too. So after a specific time, it cannot happen any more and the only possible response will be the alternative response modelled by events B_5 and B_6.

By enforcing this property with an expiry as shown in Figure 16, the concrete timing properties will satisfy their abstract ones. By having an expiry property between events A and B_3 for the period of t_1, it will be guaranteed that if event B_3 has occurred, it was within time t_1 of A occurrence (Formulated in invariant (24)).

$$B_3 = TRUE \Rightarrow tB_3 \leq tA + t_1 \tag{24}$$

From event B_3 occurrence, event B_4 has t_2 time units to happen based on the existing deadline between them. As a result, the abstract deadline has been contained by the concrete timing properties. This pattern shows how combination of deadline and expiry can be used to model timing properties of a time-critical system. The same pattern can be used for parametrized timing properties by changing the invariants according to the semantic of the parametrized timing properties.

5.6. Using Alternative Deadlines Instead of Combining Deadline and Expiry

The reader may ask why we do not use two disjunctive deadlines instead of a deadline and an expiry for timing properties similar to events A, B_3 and B_4 in the example of Section 5.5. To explain why this would not work, we will go through the refinement process of an abstract deadline by two disjunctive deadlines on a same control flow refinement pattern. Suppose we want to encode timing properties of events A, B_3 and B_5 as follow:

$$Deadline(A, B_3, t_1) \vee Deadline(A, B_5, t_3) \quad \textbf{Where} \;\; t_1 < t_3. \tag{25}$$

To encode these properties in an Event-B model, the required guard on the *Tick_Tock* event will be as follow based on the introduced deadline semantics in Section 3.3:

$$\begin{gathered}(A = TRUE \wedge B_3 = FALSE \Rightarrow time + 1 \leq tA + t_1) \\ \vee \\ (A = TRUE \wedge B_5 = FALSE \Rightarrow time + 1 \leq tA + t_3)\end{gathered} \tag{26}$$

In the following we will prove that deadline guard (27) is equivalent to the required deadline guard for deadline (27) and as a result, it does not enforce the expiry property to the model.

$$Deadline\ (A,\ B_3\ \vee\ B_5,\ t_3) \tag{27}$$

Based on the deadline semantics, explained in Section 3.3 the required guard on the $Tick\_Tock$ event for deadline (27) is as follow:

$$A = TRUE \wedge B_3 = FALSE \wedge B_5 = FALSE \Rightarrow time + 1 \leq tA + t_3 \quad (28)$$

It is easy to prove that guard (27) is logically equivalent to guard (28) since $t_1 < t_3$.

Accordingly, alternative deadlines (25) allow event B_3 to occur after time t_1 of A occurrence. As a result, by having two disjunctive deadlines, the expiry property on event B_3 will not be enforced.

In this section some approaches have been introduced in order to refine timing properties based on several control flow refinement patterns. These patterns do not contain all the possible cases of refining timing properties and we are still working on other possible refinement patterns.

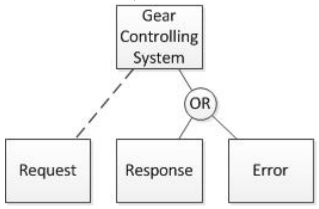
6. Application of the Approach

In order to investigate the practicality of the approach two major case studies has been modelled. The first case study was an automatic gear controlling system which has unparametrized timing properties to reach the requested gear. The other case study, was a message passing system with parametrized timing properties.

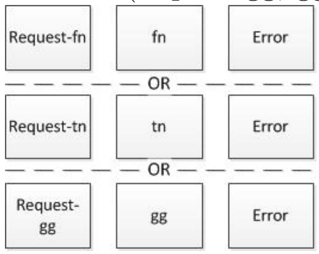
By modelling timing properties based on our approach, and refining them by using the explained refinement patterns, we managed to model these two systems in several levels of abstraction, and verify some of their important discrete timing specifications. In the automatic gear controlling system, based on its specification, it is supposed to respond a change request in 1.5 seconds or an error should be notified. We managed to model this system in 12 levels of abstraction. One of the advantage of our approach is that it is possible to do the control flow refinement first, and then do the timing properties refinement. This feature was very helpful during this case study, because after the third refinement, the model was too complex to handle the refinement of control flow and timing properties in a same level of abstraction. Some of the initial steps to model this case study are as follow:

1. In the first machine, the request event will be followed by the response event or the error event, before reaching the deadline,

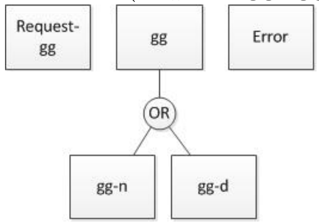
Deadline(request, response ∨ error, 1500)



2. In the next level, the request and the response are refined by three possible cases, changing from the neutral (fn), changing to the neutral (tn), changing from a gear to another(gg), and the deadline based on the pattern of Section 5.4 is replaced by
Deadline(request-fn, fn ∨ error, 1500)
Deadline(request-tn, tn ∨ error, 1500)
Deadline(request-gg, gg ∨ error, 1500)

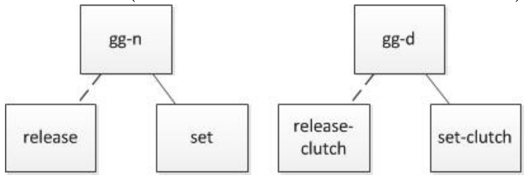


3. It is assumed that the gear changing process happens either in a normal situation (n) of a difficult (d) one, so each response case had to be refined by two alternative responses. The timing property for the gg case, based on Section 5.1 pattern, is replaced by
Deadline(request-gg, gg-n ∨ gg-d ∨ error, 1500)



4. In the next refinement the required steps to change a gear, releasing the current gear, and setting the requested one, are added to the model. In the gg case, we need to use the approach, explained in Section 5.5 to prove the timing refinement, since there is a race between the first steps of the two possible responses (gg-n or gg-d). The deadline of the previous level for the gg case is replaced by the followings:

Deadline(request-gg, release-n $\vee$ release-d $\vee$ error, 750),
Expiry(request-gg, release-n , 550),
Deadline(release-n, set-n $\vee$ error, 950),
Deadline(release-d, set-d $\vee$ error, 750).



This process continued until we added all the details of the system behaviour and its specification to the model, to verify that the timing properties of the detailed behaviour supports our assumption for the abstract behaviours.

In the message passing case study, there is an assumption about the required duration to transfer a message, part by part, based on its size and the number of acceptable data loss in the channel. In the most abstract model we had three events, representing start and end of a transmission and the error event plus a deadline on the whole process. In the next refinement, the step wise transmission process was added to the model and the timing property was refined to an iterative deadline for the iterative transmission process, based on the pattern represented in Section 5.2. In the following refinements we add the detailed steps of each transmission, such as replacing a transmission with a send and a receive.

By the end of the modelling process of this case study, we verified that according to the number of acceptable drops of a channel and the timing properties of each send, the whole process will not take more than what has been assumed in the abstract.

In this section two examples of how these approach can be helpful to model and reason about the time-critical systems and their properties.

7. Future Work

Models of large systems can end up being a complex and large, and the proof process for complex models can be difficult to accomplish. One of the approaches to deal with this complexity issue is *decomposition*. Butler [2] has introduced an approach to decompose an Event-B model of a system to its components based on shared events. We are currently exploring the use of shared-event decomposition to decompose timed Event-B models, which let us refine the resulting sub-components independently. Treating both shared variables and shared events is important in timed Event-B decomposition, because *time* is a shared variable and the *Tick_Tock* event is a shared event, and it is desirable to be able to add new timing properties and refine them in each component independently.

The approach we are taking is as follows: Before decomposing a timed Event-B model, we duplicate the *time* variable for each component. Since, the *Tick_Tock* event will become a shared event between all sub-components, the deadline guards for each timing property need to be partitioned amongst the individual *Tick_Tock* events of each sub-component. As a result, the clocks of all the components will progress synchronously. Achieving this decomposition of deadline guards requires the timing properties to be refined such they are sufficiently localised. Based on these guidelines, we managed to decompose both timed-Event-B models, developed based on the case studies of Section 6. The gear controlling system was decomposed to a model of gearbox, a model of engine, a model of clutch and a model of controller. The message passing model was decomposed to a sender, a receiver and a channel.

A prototype tool has been developed which let the modeller to specify a machine timing properties based on the introduced syntaxes, then it will add them to the model based on the semantics of the timing properties. We are working on extending this tool so that it will generate the invariants required for refinement of timing properties and provide explicit support for decomposition of timed models.

8. 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 [3] there is a delay construct $\epsilon(d) \cdot P$, which enforce the model to wait for d time units and then behave as process P and time cannot proceed if d time-units have passed and process P has yet to happen. Same mechanism has been used in Timed Modal Specification of Cerans et al. in [4] to model maximal progress assumptions, where there is a must modality which enforces the maximum delay to a model. Delay in TCCS and maximal progress in Timed Modal Specification present the same property as deadline in our work. Also, what is called a loose delay in Timed Modal Specification forces the same behaviour as a delay does in our work. Besides, Urgent Event in Evans and Schneider work [5] has been encoded by preventing the time proceeding, if an urgent event is eligible to occur. This behaviour of urgent event is the same as response events of a deadline in our work, when the current time is equal to the deadline and none of the responses have occurred yet. In Timed CSP [6] time-out presents the same property as expiry does in our work and a delay in Timed CSP causes a similar behaviour to what can be modelled by combining a delay and a deadline in our work.

Modelling time-critical systems by using Event-B has been investigated in several studies. Butler et al. in [7] explained how it is possible to model discrete time in B (had a considerable influence on Event-B), by having a natural number variable which represents the current time and an operation which forwards the time. In that study a deadline has been modelled by disabling the time progress, if the current time is equal to the deadline. This work does not investigate different kinds of timing properties and timing properties refinement have not been discussed. Cansell and Rehm in [8] have modelled a message passing algorithm in Event-B by using similar principles, 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 properties have not been mentioned, but more importantly, it is not possible to refine a timing properties to finer ones based 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 [9] 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 based on it. In their study, the deadline has not been modelled. Similar to the previous approach, refining the timing property will be an issue because the set which tracks the timing boundaries.

9. Conclusion

We believe the main contribution of this work is the introduced semantics for timing properties, which supports refinement, decomposition, and mechanized process of extending an untimed Event-B model by timing properties. Besides, in this work, several types of timing properties have been covered. By benefiting from refinement, it is possible to verify the consistency of timing properties in different levels of abstraction, and decomposition helps the modeller to independently refines the timing properties of a component in a large system.

On the other hand, still all the proofs of the timing properties refinements cannot be done automatically and needs the modeller interaction, since the automatic prover is poor in proving by cases. Besides, modelling and reasoning about the control flow is still an annoying process, and timing properties are completely dependent on them.

References

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

[2] M. Butler, Decomposition Structures for Event-B, in: Integrated Formal Methods iFM2009, Springer, LNCS 5423, Vol. LNCS, Springer, 2009. URL `http://eprints.ecs.soton.ac.uk/16965/`

[3] W. Yi, Real-time behaviour of asynchronous agents, in: CONCUR, 1990, pp. 502–520.

[4] K. Cerans, J. C. Godskesen, K. G. Larsen, Timed modal specification - theory and tools, in: CAV, 1993, pp. 253–267.

[5] N. Evans, S. Schneider, Analysing Time Dependent Security Properties in CSP Using PVS, in: ESORICS, 2000, pp. 222–237.

[6] S. Schneider, Concurrent and Real Time Systems: The CSP Approach, 1st Edition, John Wiley &amp; Sons, Inc., New York, NY, USA, 1999. URL `http://portal.acm.org/citation.cfm?id=555233`

[7] M. Butler, J. Falampin, An Approach to Modelling and Refining Timing Properties in B, in: Refinement of Critical Systems (RCS), 2002. URL `http://eprints.soton.ac.uk/256235/`

[8] D. Cansell, D. Méry, J. Rehm, Time Constraint Patterns for Event B Development, in: B, 2007, pp. 140–154.

[9] J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, A. Roth, Patterns for modelling time and consistency in business information systems, in: ICECCS, 2010, pp. 105–114.