# A Duration Pattern for Event-B Method

Joris Rehm

# A Duration Pattern for Event-B Method

Joris Rehm
Université Henri Poincaré Nancy 1 - LORIA
BP 239 - 54506 Vandœuvre-lès-Nancy - France

E-mail: joris.rehm@loria.fr

## Abstract

*Event-B is a formal method used to do Model Driven Engineering certified by theorem proving. We propose a pattern to handle duration over a predicate in this method which originally does not have particular tool to specify and reason about real-time properties.*

**Keywords:** Formal method, Real-time, Event-B method, Pattern

## 1 Introduction

Our goal is to be able to work on system with real-time aspects with "Event-B" formal method.

This formal method allows us to describe models of dynamic systems (software or hardware) as set of events modifying the value of model's variables under some conditions (called guard). In order to verify the model, we can prove (by automatic and interactive theorem proving) an invariant under the variable. We also use a relation of refinement between two models: from an abstraction to a more concrete version of the model. If the proof obligations of the refinement are done, the invariant of the abstract model is also valid for the concrete model. The logic used is a first order classical logic together with a set theory. For more details of Event-B method see [4, 3], and for the B notation see [1]. Tools can be found at Event-B website[1].

Event-B does not handle explicitly the real-time problems, hence we propose here a pattern to handle duration time over a predicate.

The next section introduces this notion and defines how it should behave. Section 3 shows the pattern, for Event-B, derived from this notion. Section 4 illustrates this by an example and we sketch a case-study in 5. Finally we conclude in 6.

## 2 Definition

Let $P(X)$ be a predicate over the variables $X$.

We want to define the duration $D(P)$, the last duration while the predicate $P$ was true. If $P$ is currently true, $D(P)$ is the time since $P$ is true. The duration $D(P)$ is defined by:

1. For the initial states: $D(P) = 0$.

2. We consider events which can either change value of the variables $X$, or make the time progress:

   (a) for a transition which makes $X$ change: if we have $\neg P(X)$ before the event and $P(X)$ after we reset the duration $D(P)$ to 0; for all the other case the value of the duration remains the same.

   (b) for a transition which makes the time progress of $d$ unit, two possible case: if $P(X)$ then the new value of the duration is $D(P) + d$; else (we have $\neg P(X)$) the value of the duration remains the same.

## 3 Pattern

To encode the counter $D$ of the definition as variable of a model, we also need to encode the predicate $P$ as boolean function, with $S$ a set of elements representing the predicates. A predicate cannot be used directly in a B model, because we use a first order logic. This is present only in the pattern, because in the final model it will be advisable to replace the predicates by the real properties which we wish to study. Below we find variables, invariant and initialisation of the pattern:

[1] http://www.event-b.org

```
VARIABLES $D$, $P$
INVARIANTS
  $inv1 : D \in S \rightarrow \mathbb{N}$
  $inv2 : P \in S \rightarrow BOOL$
EVENTS
```

```
Initialisation $\widehat{=}$
begin
  $act1 : D := \{x \mapsto 0 | x \in S\}$
  $act2 : P :\in S \rightarrow BOOL$
end
```

The variable $D$ is a function ($D \in S \rightarrow \mathbb{N}$), which return the associated duration. The variable $P$ is the encoding of the predicates as boolean function ($P \in S \rightarrow BOOL$). The initialisation applies the value zero for all the durations and place indifferently the value true or false in all the predicates. The elements of $S$ will serve as identifiers to represent the studied predicates.

We can now define the event *bt* (Begin True) who must be used every time a studied predicate becomes true:

```
Event bt $\widehat{=}$
any $x$ where
  $grd1 : x \in S$
  $grd2 : P(x) = FALSE$
then
  $act1 : D(x) := 0$
  $act2 : P(x) := TRUE$
end
```

We thus have, for an identifier $x$ of predicate, the value $P(X)$ which passes from "truth" to "false", and in that case we have to reset the associated duration.

The following event represent the time progression:

```
Event tic $\widehat{=}$
any $s$ where
  $grd1 : s > 0$
then
  $act1 : D :| \forall x \cdot x \in S \Rightarrow ($
    $(P(x) = TRUE \Rightarrow$
      $D'(x) = D(x) + s) \wedge$
    $(P(x) = FALSE \Rightarrow$
      $D'(x) = D(x)))$
end
```

Where $D :| Q(D, D')$ denote a substitution where the new value of $D$ is $D'$ and where the predicate $Q$ is true. In *tic* the time progresses of $s$ units and if a value $P(x)$ is true then it is necessary to increment of $s$ the corresponding duration $D(x)$.

In the use of this pattern, it is necessary to be careful to insert the event *bt* every time the predicate $P$ change from false to true. For that purpose, it is possible to prove, at first, a model which establishes when this predicate becomes true. And then we can refine this model to insert *bt* in the good events.

Indeed it is not possible to quantify on a predicate (second-order logic) in the notation of the method B. It is thus advisable to encode this predicate in a variable of boolean type as it is shown in the pattern. In this way, we can effectively use the concept of counter of duration on a predicate in a Event-B model.

## 4 Example

In order to illustrate our pattern, we show here a short example of a light switch.

Consider a system of three events: the event *push* represents someone pushing a button in order to switch on the light; the event *on* actually switches on the light; and the event *off* represent the automatic switch off of the light after a fixed delay $c$. The variables of the system are: the boolean $lo$ (Light On) which denotes the light; the boolean $p$ which denotes whether someone has pressed the button.

Now we want to study the duration of the predicate:

$$P \widehat{=} (p = FALSE \wedge lo = TRUE)$$

This predicate means: "the light is on and nobody has pressed the button". Thus we add the variable $D$ with $D(P)$ being the duration of $P$.

At the initialisation, the duration is zero and the other variables are set to FALSE:

```
Initialisation $\widehat{=}$
begin
  $act1 : lo := FALSE$
  $act2 : p := FALSE$
  $act3 : D(P) := 0$
end
```

The event *push* just assign $TRUE$ on $p$ in any case:

```
Event push $\widehat{=}$
begin
  $act1 : p := TRUE$
end
```

And the event *on* can be triggered if someone has pushed the button (see $grd1$). In this case, we switch the light on ($act1$); accept the user request ($act2$); and reset the duration to zero ($act3$) as specified in the pattern because $P$ changes from $FALSE$ to $TRUE$.

```
Event on $\widehat{=}$
when
  $grd1 : p = TRUE$
then
  $act1 : lo := TRUE$
  $act2 : p := FALSE$
  $act3 : D(P) := 0$
end
```

Now, with the event *off* the system switches off (*act*1) the light if: the light is on (*grd*1); nobody pushed the button (*grd*2) (otherwise *on* must take place before); and the duration while the light was on (without button pressed) is equal to the constant *c*.

```
Event off $\widehat{=}$
when
  $grd1 : lo = TRUE$
  $grd2 : p = FALSE$
  $grd3 : D(P) = c$
then
  $act1 : lo := FALSE$
end
```

Finally, the event *tic* represents the time progression. We find here a standard part (*grd*1 to *grd*3, and *act*1) from the pattern. In addition, the guard *grd*4 makes the event *on* occurring instantaneously after the event *push*. And *grd*5 forces the event *off* to occur when $D(P) = c$.

```
Event $\widehat{=}$ tic
any $x$ where
  $grd1 : x \in \mathbb{N}$
  $grd2 : P \Rightarrow x = D(P) + 1$
  $grd3 : \neg P \Rightarrow x = D(P)$
  $grd4 : \neg(p = TRUE)$
  $grd5 : \neg(P \land D(P) = c)$
then
  $act1 : D(P) := x$
end
```

## 5 Case study

As a case study for our proposition, we verified an algorithm of asynchronous communication. It is an algorithm from H. R. Simpson [8] in a version with a two slots memory. This version is not totally asynchronous, but it requires less space memory than the full version with four slots memory. The full version has been studied [2] also with Event-B Method and the first model of our case study (the more abstract specification) is equivalent to the first and second models of this case study. As the asynchronism is not total, some behaviour of the communicating processes will be forbidden, to specify it we used real-time constraints. To take into account those constraint, we used our pattern to specify the durations of some properties of the system formed by the algorithm. And with this specification, we have verified that this particular version with real-time constraint is still correct.

In this paper, because of the lack of space, we are not going to report the full proved development but we show the most interesting points.

The purpose of the algorithm is to allow a one-way asynchronous communication between two entities. As the communication is made in a one way, we name one of the entities the “writer” and the other one the “reader”. Furthermore, the direction of communication goes from the writer towards the reader. At any time, the writer can send a new value, and the reader can obtain it, or not, in an (almost) independent way. This is implemented with variables (a memory) shared between both entities.

As an illustration, we can imagine that the writer is an electronic thermometer regularly updating the temperature and that the reader is another device reading the current value of the temperature when needed.

As usual in Event-B development, we have a chain of model which refine each other. The first model is the most abstract specification, and we consider two atomic events named *read* and *write*. But in the implementation and in the last model, the reading and writing operations are not atomic (the size of the communicated data is not limited). To represent this peculiarity in the first model, there will be a gap between the read value and the written value. But the algorithm gives guarantees onto the level of freshness of the read value and we have formalised the value of the gap. Informally, we can say that the read value is as recent at least as the last value written at the time of the previous reading.

We skip here most of the content unrelated to real-time of the study and we abstract it with the major point: The key for this algorithm is not to write twice in a row during the same reading, this would provoke two actions on the same slot of the (2 slots) memory, which is undesirable.

To implement this, this version of the algorithm uses real-time properties. One of the real-time constraints is the limitation of reading duration, we thus put the invariant $cr < minbw$, with *cr* being the duration of the reading operation and *minbw* a constant used as limit. Another important constraint is that the duration between two consecutive writing is equal to *minbw*, at least. Thus, when the system is writing, the writer spent at least *minbw* unit of time to not write, which corresponds to the invariant $writing \neq \varnothing \Rightarrow cnw \geq minbw$, where $writing \neq \varnothing$ means "there are no operation of writing", and $cnw$ is the duration of the non-writing.

In this case study, we applied our pattern in order to obtain (among other) $cr$, the duration of the reading operation, and $cnw$ the duration between two operation of writing.

The proved development was conceived on the Rodin software tool (from the European project of the same name) with the prover B4Free of the ClearSy company. All the proof obligations (PO) were cleared. The following table gives the details of the number of proof obligation by models:

| Model | Total | Auto | Manual | To do |
|---|---|---|---|---|
| m0 | 34 | 25 | 9 | 0 |
| m1 | 12 | 12 | 0 | 0 |
| m2 | 42 | 39 | 3 | 0 |
| m3 | 33 | 28 | 5 | 0 |
| m4 | 33 | 30 | 3 | 0 |

The PO labelled "Auto" are done without user intervention, while the label "Manual" express an interactive session of proving. We found the interactive proof quite easy and short.

## 6 Conclusion

We propose a new pattern which improves Event-B method in order to reason about duration on predicate. We have already proposed another pattern [5, 7] which focuses on the future time of execution of event.

Those two concepts which we have defined are dual, the previous pattern allows us to force the progress of a system in the time whereas the counter of duration allows us to measure the behaviour of a system in the time.

The concept of duration was defined here as properties which it has to verify on a system of transition, then we showed how to obtain a Event-B model which will serve as pattern of refinement to use these concepts, in practise, in a proved development.

On the topic of the duration, many research work was carried out on the "duration calculus" [9] and a way to combine this approach with the (classical) B Method [1] was done in [6].

We applied our pattern of duration on a case study, this allowed us to replace the abstract specification by real-time constraints. This case study is only partially described here because of the lack of space, but it still shows an application of our method and shows the reuse of a generic model of a real-time aspect for modeling and proving.

## References

[1] J.-R. Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, Aug. 1996.

[2] J.-R. Abrial and D. Cansell. Formal development of simpson's 4-slot algorithm. Technical report, Private communication, March 2006.

[3] J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inform.*, 77(1-2):1–28, 2007.

[4] D. Cansell and D. Méry. Foundations of the b method. *Computers and Artificial Intelligence*, 22(3):221–256, 2003.

[5] D. Cansell, D. Mèry, and J. Rehm. Time constraint patterns for event B development. In *B 2007: Formal Specification and Development in B*, volume 4355/2006, pages 140–154. Springer, January 17-19 2007.

[6] S. Colin, G. Mariano, and V. Poirriez. Duration calculus: A real-time semantic for b. In Z. Liu and K. Araki, editors, *ICTAC*, volume 3407 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 2004.

[7] J. Rehm and D. Cansell. Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In F. B. Yamine Aït Ameur and Virginie Wiels, editors, *RNTI ISoLA 2007 Workshop On Leveraging Applications of Formal Methods, Verification and Validation*, volume RNTI-SM-1, pages 179–190, Poitiers-Futuroscope France, 12 2007. Cépaduès.

[8] H. Simpson. Four-slot fully asynchronous communication mechanism. *Computers and Digital Techniques, IEE Proceedings* -, 137(1):17–30, Jan 1990.

[9] C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems.* EATCS: Monographs in Theoretical Computer Science. Springer, 2004.