A method to refine time constraints in event B framework

Joris Rehm

A method to refine time constraints in event B framework

Joris Rehm[1,2]

LORIA
Université Henri Poincaré Nancy 1
BP 239
54506 Vandœvre-lès-Nancy
France

Abstract

Some software or hardware system involves time constraints. When those constraints are required to express the behaviour of the system, we need to write them in the corresponding formal model. We show in this short paper the general method used to deal with time constraints with a simple application example. This applies for event B formal method which does not have specific notions for time and uses the refinement to introduce it.

Keywords: formal method, event B, refinement, time

1 Introduction

We present a abstract model of message passing without time and a refined model with time constraints and proved properties. Those constraints are introduced in a refinement which allows us to study the general and the specific properties separately.

We studied the leader election protocol from IEEE 1394 from the work of J.R. Abrial, D. Cansell and D. Méry [2]. In the final step (root contention) of this distributed algorithm, many time constraints and timers are used. Consequently, the root contention is very abstract in [2]. In order to extend this work with a detailed proved model, we wrote a prototype, presented here, with the central problem involved : representation of messages passing with real time constraints.

1.1 Event B formal method

Models of this formal method [1] have named events which modify the values of variables. An event has a guard : it is a logical expression which allows or not

[1] Thanks to D. Cansell, D. Méry and the MOSEL team of LORIA

[2] Email: joris.rehm@loria.fr

execution of the event. An invariant restricts the set of allowed system states, it must be preserved on variables by the events. Models are proved when proof obligations, given by B theory, are proved with the invariant. In the end, a model can refine one more abstract model; the refinement must be proved correct by specific proof obligations. Refinement is very important in the method. It is used to introduce specification, implementation and add details in an incremental process.

2 Message passing : abstract model

We design a system of two devices a and b. Device a send one message to b, triggers a timer and sleep until it ends. The most important element consists to prove the reception of message by b at the end of the timer. In a first step, we introduce the problem with an abstract model so the timer is writen very abstractly (without time).

Constants a and b represent the devices, the variable AB represent the content of the channel between the two devices. Three events can occur :

- sendA : a sends its message to b using connection AB
- recB : b receives it from the connection AB
- quA : when a knows that the message is received by b, it modifies one of its local variable S.

```
INVARIANT
  $A \subseteq \{a\} \wedge$
  $B \subseteq \{b\} \wedge$
  $AB \subseteq \{a\} \wedge$
  $S \subseteq \{a\} \wedge$
  $(A \neq \emptyset \implies AB \neq \emptyset)$
```

We can see the definition of variables in the 4 first lines of the invariant above. Variable A denotes the sending of the message if and only if A is not empty, similarly B denotes its reception and S denotes the state after execution of quA. All variables are empty or not. According to a distributed system, we consider that A and S are local variables for device a, B is a local variable for device b and AB is a global variable.

Next, we define the three events:

```
sendA $\widehat{=}$
  when
    $A = \emptyset$
  then
    $A := \{a\} \mid\mid AB := \{a\}$
  end
```

```
recB $\widehat{=}$
  when
    $AB = \{a\}$
  then
    $B := \{b\} \mid\mid AB := \emptyset$
  end
```

```
quA $\widehat{=}$
  when
    $B = \{b\}$
  then
    $S := \{a\}$
  end
```

In the guard of event quA we explicitly ask the message to be received. In the abstract model, we are "cheating" because this event is intended to be local to device a but it uses the variable B which is intended to be local to device b. It is as if device a can use local information of device b.

3 Refining time constraints in the model

There are no specific elements in B method to deal with time. However, its language contains first order set theory and arithmetic, and is therefore sufficiently expressive to represent the state of a real-time system. For doing so, we loosely follow M. Abadi and L. Lamport in [3] and we represent time and timers as additional variables of the system.

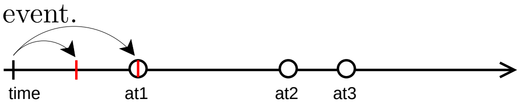
The main idea is to guard events with a time constraint, therefore those events can be observed only when the system reaches a specific time. We call those times **active times** and we say that events will "process" it. As we want to model dynamic system, we need a way to "post" new active times in the future. If posting events do not was guarded by time constraints they can initiate timed events else if they was also constrained by time it's a way to denote a exact timing between two events. These two variables are the most important ones :

- $time$ in $\mathbb{N}$ models the current time value.
- $at \subseteq \mathbb{N}$ is the known future active times of the system. Each active time stands for one future event activation.

We need two constants : $prop$ is the propagation time needed by the message to transit from a to b and st is the sleeping time used in the timer. We need also two new variables: stm is the "send time message" and slp the time when a will stop sleeping at the end of the timer.

Now we can refine the abstract model. I would explain the event "*tick_tock*" first, because it is independent of any concrete system model, and it captures the model of time.

This event takes a new current time tm in the future which is indeterminate and must be before the first active time because we do not want to miss the right moment for observing a time-constrained event.



```
tick_tock  $\widehat{=}$
  any  $tm$  where
    $tm \in \mathbb{N}$  $\wedge$
    $tm > time$  $\wedge$
    $(at \neq \emptyset \implies tm \leq \mathsf{min}(at))$
  then
    $time := tm$
  end
```

The event $sendA$ sets up the informative variables stm and slp with the corresponding time and add two new active times in the set at :

```
sendA  $\widehat{=}$
when $A = \emptyset$ then
  $A := \{a\}$ ||
  $AB := \{a\}$ ||
  $at := at \cup \{time + prop\} \cup$
$\{time + st\}$ || /* added */
  $stm := time$ || /* added */
  $slp := time + st$ /* added */
end
```

We can see, in $sendA$, the two new active times $time+prop$ and $time+st$ which are the future arrival time of messages and the awake time ending the timer. For this event, the refinement is just a superposition, i.e. some lines have been added without change existing expressions. Superposition are usually easy to prove.

When the device b receives the message, we can observe the event *recB*.

```
recB $\widehat{=}$
when
    $AB = \{a\} \wedge$
    $time = stm + prop$ /* added */
then
    $B := \{b\}$ ||
    $AB := \emptyset$ ||
    $at := at - \{time\}$   /* added */
end
```

In the event *recB*, the guard $time = stm+prop$ ensures that *time* has reached the first active time which has been added for the message arrival.
This current active time is deleted from *at* thus enabling time to progress again.

In event *quA* we use the sleep time to wait after message reception; the event is triggered by the expiration of the timer *slp*.

```
quA $\widehat{=}$
when
    $A \neq \emptyset \ \wedge$ /* changed to a local guard */
    $time = slp$ /* added */
then
    $S := \{a\}$ ||
    $at := at - \{time\}$ /* added */
end
```

Here the refinement is not just a superposition: the abstract guard was $B = \{b\}$ and is changed to a locally available condition : $A \neq \emptyset \wedge time = slp$. The use of the non local variable B has disappeared with the use of the local variable A and of the variable *time*. Variable *time* is universal and global so we can use it to get more information from the local state of distributed devices. In this model, the variable *time* is the real time and implementation of our model have to use clocks.

In order to prove the refinement we need the following invariant:

$$\begin{array}{l}
time \in \mathbb{N} \\
stm \in \mathbb{N} \\
slp \in \mathbb{N} \\
at \subseteq \mathbb{N} \\
(A \neq \emptyset \implies stm + prop < slp) \\
(A \neq \emptyset \wedge time \geq stm + prop \wedge stm + prop \notin at \implies B = \{b\}) \\
(at \neq \emptyset \implies time \leq min(at)) \\
at \subseteq \{stm + prop, slp\} \\
(A = \emptyset \implies at = \emptyset) \\
(A \neq \emptyset \wedge at = \emptyset \implies time \geq slp) \\
(A \neq \emptyset \wedge at \neq \emptyset \implies slp \in at) \\
(A \neq \emptyset \wedge at = \{slp\} \implies time \geq stm + prop)
\end{array}$$

We give explanations of the most interesting parts of this invariant:

- $A \neq \emptyset \wedge stm + prop \notin at \wedge time \geq stm + prop \Rightarrow B = \{b\}$:

This part of the invariant is important to prove the refinement of quA. In this expression if time is beyond $stm + prop$ and if the time constraint $stm + prop$ has already been used then we are sure of the reception ($B = \{b\}$).

- $A \neq \emptyset \wedge at = \{slp\} \Rightarrow time \geq stm + prop$:

  If active times set is only $\{slp\}$ and if the message is sent then current time is after the message reception.

- $A \neq \emptyset \wedge at = \emptyset \Rightarrow time \geq slp$:

  This predicate is interesting if the message has already been sent ($A \neq \emptyset$) and if there is no more time constraints on process ($at = \emptyset$), in other words once all the events were observed. In this case, one can affirm that the current time exceeded the moment when a was awoken.

- $A \neq \emptyset \implies stm + prop < slp$:

  This invariant uses the fact that $prop < st$ because event $sendA$ provides the following proof obligation: $\{a\} \neq \emptyset \implies time + prop < time + st$. This fact is a property on constants st and $prop$ which expresses that the propagation time is less than the sleep time.

The abstract event quA is cheating, since it looks at the variable B in its guard ($B = \{b\}$); the refined version is no more cheating, since the guard is local $A \neq \emptyset$ (message is sent) and the time constraint $time = slp$. Only $time$ is a global variable shared by each participant of the global system: it is local for each participant and everyone has the same time. We assume that the time is the same for everyone.

4 Conclusion

Our work illustrates the use of a explicit time variable which interacts with a number of active times. Those active times can be put in the future in order to constrain some events with precise timing. This concept has been used in a simple example of message passing, which is a fundamental problem and occurs also in [2]. From there we intend to use this work on the root contention problem of IEEE 1394 and study properties of real time event-driven systems.

References

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

[2] J-R. Abrial, D. Cansell and D. Méry *A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol*, Formal Asp. Comput. **14** (2003), 215-227

[3] M. Abadi and L. Lamport *An old-fashioned recipe for real time*, ACM Transactions on Programming Languages and Systems, **16(5)** (1994), 1543-1571