# From Absolute-Timer to Relative-Countdown: Patterns for Model-Checking

Joris Rehm

# From Absolute-Timer to Relative-Countdown: Patterns for Model-Checking

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

E-mail: joris.rehm@loria.fr

## Abstract

*Many specialised formal methods exist for specifying and verifying real-time systems. We propose extending a traditional method in order to model time with a pattern. In order to carry out verification by model-checking, we demonstrate a new instance of a pattern for real-time modelling. The former pattern is useful to carry out verification by theorem proving. The equivalence with the previous version is studied, and interesting properties for model-checking are reviewed. Finally we report on an experimental case-study.*

**Keywords.** formal method, real-time, model-checking, Event-B method, pattern

## 1 Introduction

Despite numerous work on timed systems and sophisticated tools for verifying them, many people still use non-specialised formal methods to work on their timed specifications and systems. This is not because the results in this area are not recognised but because it is complex to mix together different theories and tools. In many cases real-time systems arise in specialised areas like distributed computing, for this reason it is a promising approach to extend an existing specification with some timing properties. The work presented in this paper starts with this situation: we can see in [6] a distributed election algorithm, where most of the problem studied does not require time to be taken into account. But the final phase of this algorithm uses timing constraints, so we found it useful to find a practical way to verify those timing aspects using the same formal method: the Event-B method. We have already proposed a way to model the time as a pattern for this method in [10] and we succeeded in verifying our case-study by theorem proving in [19]. In this work, here a pattern is an element of methodology which explains how it is possible to handle time constraints within the Event-B method.

Section 2 of this paper describes the initial real-time pattern from [10] with a light improvement. In addition to theorem proving we used model-checking on the case-study [19], we report here the experience gained in this work. Models created with this pattern cannot be model-checked easily because the pattern contains unbounded variables. For example, it contains the variable $now$ for the current time and of course the time progresses indefinitely.

We propose in section 3 an equivalent version of this pattern for real-time modelling and checking. This new version uses only relative values about time and timing. We demonstrate the equivalence of the two patterns through a proof of bi-refinement (mutual refinement between the two model). And we show the properties of this pattern which allow the finiteness of the states reachable by model-checking. Finally section 4 reports on the application of our technique on a case-study, and we conclude.

### 1.1 Related works

In [1] Abadi and Lamport show a "recipe" to model real-time specifications without a specialised formal method. They call this: explicit-time specification. They focus on worst-case upper and lower bounds on real-time delays. They use the variable $now$ which is a real number that never decreases to represent the current time. Transition of the system can be make the time progress or not. Time-progression transitions are done by the action called "tick", and timing constraints can block the time in order to force other actions to run. An absolute timer, which can be a lower-bound or a upper-bound, imposes timing bound on actions. A volatile $\delta$-timer counts how much time an action has been continuously enabled. A persistent $\delta$-timer is the same as a volatile $\delta$-timer without the continuous condition over the activation.

More generally the application of a formal-method over a timed system is a rich scientific domain. We can cite several review papers such as [21, 14]; we can also find

books on the subject such as [23]. Besides general study within this domain, many "untimed" formal methods have extensions to handle real-time. For the (classical) B method (which is the "parent" of the Event-B method used here) an extension using duration calculus is described in the thesis [11] and in the article [12]. Our work here is largely different because classical B principally uses operations, which take a certain amount of time to run, whereas an event is an instantaneous action. For Action Systems, the time aware system refinement in [22] and in this approach gives a theory for stepwise refinement with time, actions also take a certain time. It is also interesting to look at the work concerning CSP and time: a retrospective can be found in [18] and more complete material in the book [20].

In the same context as our work, i.e. explicit-time specification and model-checking, we can cite two articles.

In [16] Lamport advocates that it is easy to write and verify explicit-time specification (here with TLA [15]) with an ordinary model-checker. This work follows [1] for the modeling issue and shows a view symmetry over states, and results are compared with real-time model-checkers for two examples. The author introduces a symmetry under time-translation (two states are equivalent iff they are the same except for absolute time) and use this symmetry with the model-checker of TLA, namely TLC, with discrete time.

Similarly, Dutertre and Sorea in [13] use SAL to model and verify an explicit-time specification of a distributed algorithm of election. Here timing constraints are modeled as a timeout and calendar. The calendar gives the future times of execution of some events, and this notion is very close to our pattern for Event-B in [10]. But the authors do not use a non-deterministic "tick" event which makes the time progress. Instead the time goes directly from one event activation to the next event activation. This model prevent the use of clocks which vary continuously. Furthermore, this model allows the authors to use a continuous time with an ordinary model-checker because no values varies continuously in this model.

We are going to show here a different solution with a particular event for time-progression and which uses discrete time.

## 2 Overview of event-B development by stepwise refinement

### 2.1 Event-based modelling

Our event-driven approach [3] is based on the B notation. It extends the methodological scope of basic concepts in order to take into account the idea of *formal models*. Roughly speaking, a formal model is characterised by a (finite) list $x$ of *state variables* possibly modified by a (finite) list of *events*; an invariant $I(x)$ states properties that must always be satisfied by the variables $x$ and *maintained* by the activation of the events. In the following, we briefly recall definitions and principles of formal models and explain how they can be managed by tools [5].

Generalised substitutions are borrowed from the B notation. They provide a means for expressing changes to state variable values. In its simple form, $x := E(x)$, a generalised substitution looks like an assignment statement. In this construct, $x$ denotes a vector built on the set of state variables of the model, and $E(x)$ a vector of expressions. However, the interpretation we shall give here to this statement is not that of an assignment statement. We interpret it as a *logical simultaneous substitution* of each variable of the vector $x$ by the corresponding expression of the vector $E(x)$. There exists a more general normal form, denoted by the construct $x :| P(x, x')$. This should be read: "$x$ is modified in such a way that the predicate $P(x, x')$ holds", where $x'$ denotes the *new value* of the vector and $x$ denotes its *old value*. This is clearly non-deterministic in general.

An event has two main parts: a *guard*, which is a predicate built on the state variables, and an *action*, which is a generalised substitution. An event can take one of the three normal forms. The first form ($evnt \;\widehat{=}$ BEGIN $x :| P(x, x')$ END) shows an event that is not guarded: it is thus always enabled and is semantically defined by $P(x, x')$. The second ($evt \;\widehat{=}$ WHEN $G(x)$ THEN $x :| Q(x, x')$ END) and third ($evt \;\widehat{=}$ ANY $t$ WHERE $G(t, x)$ THEN $x :| R(x, x')$ END) forms are guarded by a guard which states the necessary conditions for these events to occur. Such a guard is represented by WHEN $G(x)$ in the second form, and by ANY $t$ WHERE $G(t, x)$ (for $\exists\, t \cdot G(t, x)$ ) in the third form. We note that the third form defines a possibly non-deterministic event where $t$ represents a vector of distinct local variables. The, so-called, before-after predicate $BA(x, x')$ associated with each of the three event types, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before ($x$) and just after ($x'$) the "execution" of event $evt$.

Proof obligations are produced from events in order to state that an invariant condition $I(x)$ is preserved. Their general form follows immediately from the definition of the before-after predicate, $BA(x, x')$, of each event:

$$\boxed{I(x) \wedge BA(x, x') \;\Rightarrow\; I(x')}$$

Note that it follows from the two guarded forms of the events that this obligation is trivially discharged when the guard of the event is false.

### 2.2 Model Refinement

The refinement of a formal model allows us to enrich a model in a *step-by-step* approach, and is the foundation of

our *correct-by-construction* approach. Refinement provides a way to strengthen invariants and to add details to a model. It is also used to transform an abstract model into a more concrete version by modifying the state description. This is done by extending the list of state variables, by refining each abstract event into a corresponding concrete version, and by adding new events. The abstract state variables, $x$, and the concrete ones, $y$, are linked together by means of a, so-called, *gluing invariant* $J(x, y)$. A number of proof obligations ensure that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines $skip$, (3) no new event takes control for ever, and (4) relative deadlock-freeness is preserved. Details of the formulation of these proofs follows.

We suppose that an abstract model $AM$ with variables $x$ and invariant $I(x)$ is refined by a concrete model $CM$ with variables $y$ and gluing invariant $J(x, y)$. If $BAA(x, x')$ and $BAC(y, y')$ are respectively the abstract and concrete before-after predicates of the same event, we have to prove the following statement, corresponding to proof obligation (1):

$$\boxed{\begin{array}{l} I(x) \wedge J(x,y) \wedge BAC(y,y') \\ \quad \Rightarrow \\ \exists x' \cdot (BAA(x,x') \wedge J(x',y')) \end{array}}$$

Now, proof obligation (2) states that $BA(y, y')$ must refine $skip$ ($x' = x$), generating the following simple statement to prove (2):

$$\boxed{I(x) \wedge J(x,y) \wedge BA(y,y') \;\Rightarrow\; J(x,y')}$$

For the third proof obligation, we formalise the notion of the system advancing in its execution; a standard technique is to introduce a variant $V(y)$ that is decreased by each new event (to guarantee that an abstract step may occur). This leads to the following statement to prove (3):

$$\boxed{I(x) \wedge J(x,y) \wedge BA(y,y') \;\Rightarrow\; V(y') < V(y)}$$

Finally, to prove that the concrete model does not introduce additional deadlocks, we give formalisms for reasoning about the event guards in the concrete and abstract models: $\mathsf{grds}(AM)$ represents the disjunction of the guards of the events of the abstract model, and $\mathsf{grds}(CM)$ represents the disjunction of the guards of the events of the concrete model. Relative deadlock freeness is now easily formalised as the following proof obligation (4):

$$\boxed{I(x) \wedge J(x,y) \wedge \mathsf{grds}(AM) \;\Rightarrow\; \mathsf{grds}(CM)}$$

To review, refinement guarantees that the set of traces of the refined model contains (modulo stuttering) the traces of the resulting model.

| | |
|---|---|
| $\mathbb{P}(E)$ | Power set of E |
| $E \setminus F$ | Set difference |
| $x \mapsto y$ | Pair $(x, y)$ |
| $E \rightarrow F$ | Set of total functions from E to F |
| $\mathrm{dom}(f)$ | Domain of f |
| $f\,;g$ | Forward composition of function |
| $f \lhd\!\!+ \{x \mapsto y\}$ | Overriding with $(x, y)$ over $f$ |
| $x := y$ | $x$ becomes equal to $y$ |
| $x :\in E$ | $x$ becomes element of $E$ |

**Figure 1. Event-B Notations used in this paper**

Fig. 1 shows notations, from the B-method, used in this paper. Most of them are classical. The $\lhd\!\!+$ operator can be defined by $r \lhd\!\!+ s \mathrel{\widehat{=}} \{x \mapsto y |$ if $x \in dom(s)$ then $x \mapsto y \in s$ else $x \mapsto y \in r\}$. This operator gives a convenient way to change only a subset of the mappings of a relation. To change only one mapping of a function we can use: $f(x) := E$ which is defined by $f := (f \setminus \{x \mapsto f(x)\}) \cup \{x \mapsto E\}$. The two last operators $:=$ and $:\in$ define a substitution and can be used only in the action part of an event.

For more details of the event-B method see [9, 7], and for the B notation see [2]. Tools can be found at the Event-B website[1].

## 3 Real-Time Pattern

Our explicit-time pattern [10] for real-time system is based on an event-calendar. Let $evts$ be the finite set of events for one model. And let the variables $now$ and $at$ (stands for Activation Times). The pattern variable $now$ represents the current real-time, here we have a discrete time: $now \in \mathbb{N}$. And $at$ is the event-calendar. In this pattern, an event-calendar is a function that gives for every element of $evts$, a set of activation times in the future: $at \in evts \rightarrow \mathbb{P}(\mathbb{N})$. Therefore, we have in invariant:

$$\forall e \cdot (e \in evts \wedge at(e) \neq \varnothing \Rightarrow now \leq min(at(e)))$$

Fig. 2 gives the pattern which shows how to write an event-B model of a real-time system. Each event of the model will refine one (or maybe several) event(s) of the pattern. In [10], which originally defines the pattern, a possible specific improvement is given: we can index different sets by a process or a name. As you can see, we decided here to systematically index the $at$ set by the corresponding event, which will use this $at$ set. We think this improvement should become standard as it provides a crucial information for verifying and validating a model.

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

**INITIALISATION** $\widehat{=}$
**BEGIN**
act1: $now := 0$
act2: $at :\in evts \rightarrow \mathbb{P}(\mathbb{N})$
**END**

**add** $\widehat{=}$
**ANY**
$e$
$ntime$
**WHERE**
grd1: $e \in dom(at)$
grd2: $now < ntime$
**THEN**
act1: $at(e) := at(e) \cup \{ntime\}$
**END**

**use** $\widehat{=}$
**ANY**
$e$
**WHERE**
grd1: $e \in dom(at)$
grd2: $now \in at(e)$
**THEN**
act1: $at(e) := at(e) \setminus \{now\}$
**END**

**tic** $\widehat{=}$
**ANY**
$n\_now$
**WHERE**
grd1: $now < n\_now$
grd2: $\forall e \cdot e \in dom(at) \wedge at(e) \neq \varnothing$
$\Rightarrow n\_now \leq min(at(e))$
**THEN**
act1: $now := n\_now$
**END**

**Figure 2. Real-time pattern for Event-B.**

The pattern has three aspects and of course one initialisation. The event **add** shows how to add a new future activation time $ntime$ in the calendar of event $e$ . The event **use** shows how to activate an event $e$, at the current time $now$, if $e$ has been scheduled to this current time ($now \in at(e)$); in this case we remove the current time from the calendar of $e$. Finally the event **tic** represents the time progression, we increase the current time at least to $now + 1$ and at most to the smallest time of the calendar (if any).

## 4 Real-Time Pattern with Relative Timing

Most, if not all, non trivial real-time systems are cyclic, or are composed by cyclic elements. Their behaviours do not depend on an absolute timing but only on relative delays between events. Therefore, for model-checking, it is crucial to exploit this property. With timed automata [8] (and so for many real-time model-checkers) the model relies on clocks which can be reset. Therefore we can easily specify a finite model. But in explicit-time models (see section 1.1) we use a time model more close to an absolute interpretation of time. This interpretation is natural for verification by theorem proving but inefficient for direct model-checking because many variables of the model increase indefinitely. The article [16] uses three kinds of timer variables: countdown timer, count-up timer, and expiration timer. The article [13] uses timeout and event-calendar. In our approach we use the event-calendar $at$ defined in the previous section. The expiration timer, timeout and event-calendar are quite similar, in the sense that they refere to a future absolute time. We consider here the (variable) function $at$, which has already been formally defined as an event-calendar.

We show here how to refine a model with absolute timing to a model with relative timing and we show the equivalence (in sense of bi-refinement) of the two models.

Firstly $now$ is rewritten to 0. Secondly $at$ is refined to $rat$ (represents the Relative Activation Time) with the invariant: $rat \in evts \rightarrow \mathbb{P}(\mathbb{N})$, and $dom(at) = dom(rat) = evts$, and:

$$\forall e \cdot \left( \begin{array}{l} e \in dom(at) \Rightarrow \\ \quad (\forall x \cdot x \in rat(e) \Leftrightarrow x + now \in at(e)) \end{array} \right)$$

As you can see the two variables $now$ and $at$ of the abstract model disappear and the new refined model only has $rat$ as variable. We can see in this invariant that the relation between abstract and concrete variables is stated, we call this a gluing invariant. Instead of letting the current time $now$ progress to a event activation time, we decrease all activation times to zero. As in the previous abstract pattern we can add new timeout in the future and use it when the time comes (it is when one timeout of $rat$ is equal to 0). In Fig. 3 we can see the formal pattern refined accordingly. All the proofs for the invariant and the refinement were done with the Rodin tool [4], we found the proof easy and only a few

interactive steps are needed. The refinement proof is done in two directions to show the equivalence.

In event **add** we can see in the witness that the absolute new timeout $ntime$ is equal to $ntimer + now$, with $ntimer$ the relative new timeout. In the witness one is allowed to use old variables of the abstract event in order to prove the refinement. Therefore we can see that the concrete version of the event **add** shows the same behaviour as the concrete version if we shift the value of the added timeout by $now$.

For the event **use** it is easy to see that $0 \in rat(e)$ is equivalent to $now \in at(e)$ as the values of $rat$ are the value of $at$ shifted by $now$, as stated in the gluing invariant.

Finally for the equivalence of the event **tic** we need to show that increasing the variable $now$ (not after the first value of $at$) is equivalent to decreasing all the values inside $rat$ (not below zero). In the abstract version, the local variable $n\_now$ is the new increased value of $now$; this local variable disappears and is replaced by $now + shift$. The variable $shift$ is a new local variable which is the relative value of time progression and this value cannot be more than the smallest value of $rat$. We also defined an auxiliary function as a local variable of this event: the function $m$ is used to decreased by $shift$ the values inside a set given by the parameter of $m$. In order to decrease all the timeouts of the relative event-calendar $rat$, we replace $rat$ by the forward composition of $rat$ and $m$. Intuitively the equivalence seems to be correct and to verify this with the proof assistant we just needed to instantiate correctly the gluing invariant and to apply the definition of $m$.

Of course everything works here because we use discret time. We must consider very carefully the choice between discrete or continuous time modelling: a discrete time adds a synchronisation between transitions. Consequently, some subtle error of interleaving may be not revealed by the model checking. If the model-checker requires to instantiate parameters, and if the values of those parameters control the possible interleaving between events, then one has to choose parameters carefully in order to not forbid important traces. Of course, if you use a parametric model-checker or do verification by theorem-proving, then parameters of the model can be left non instantiated (with maybe some abstract conditions between them). In this case, discrete time is not a big issue because the real-time timing value can be as high as we want. And if the timing betwen event activations is as long as we want, we can split the time as small as we want, but not indefinitely.

For continuous time, model-checking with a non-specialised model-checker becomes difficult. The continuity of the time domain leads to an infinite number of states, for example see how [13] deals with this problem employing a classical model-checker.

**INITIALISATION** $\widehat{=}$
**BEGIN**
act1: $rat :\in evts \to \mathbb{P}(\mathbb{N})$
**END**

**add** $\widehat{=}$
**ANY**
$e$
$ntimer$
**WHERE**
grd1: $e \in dom(rat)$
grd2: $0 < ntimer$
**WITH**
ntime: $ntime = now + ntimer$
**THEN**
act1: $rat(e) := rat(e) \cup \{ntimer\}$
**END**

**use** $\widehat{=}$
**ANY**
$e$
**WHERE**
grd1: $e \in dom(rat)$
grd2: $0 \in rat(e)$
**THEN**
act1: $rat(e) := rat(e) \setminus \{0\}$
**END**

**tic** $\widehat{=}$
**ANY**
$shift$
$m$
**WHERE**
grd1: $0 < shift$
grd2: $\forall e \cdot \left( \begin{array}{c} e \in dom(rat) \wedge rat(e) \neq \varnothing \\ \Rightarrow shift \leq min(rat(e)) \end{array} \right)$
grd3: $m \in ran(rat) \to \mathbb{P}(\mathbb{N})$
grd4: $\forall s \cdot \left( \begin{array}{c} s \in \mathbb{P}(\mathbb{N}) \\ \Rightarrow m(s) = \{x | x + shift \in s\} \end{array} \right)$
**WITH**
$n\_now : n\_now = now + shift$
**THEN**
$act1 : rat := rat; m$
**END**

**Figure 3. Refined real-time pattern for Event-B with relative timing.**

**init** $\widehat{=}$
  **BEGIN**
    act2: rat := { off $\mapsto \varnothing$ }
    act3: light_on := FALSE
  **END**

**switch_on** $\widehat{=}$
  **ANY**
    d
  **WHERE**
    grd1: d$\in$ 9.. 11
    grd2: light_on=FALSE
  **WHERE**
    act1: light_on:= TRUE
    act2: $rat(off) := rat(off) \cup \{d\}$
  **END**

**switch_off** $\widehat{=}$
  **WHEN**
    grd1: 0 $\in$ rat(off)
  **THEN**
    act1: $rat(off) := rat(off) \setminus \{0\}$
    act2: light_on:= FALSE
  **END**

**tic** $\widehat{=}$
  **ANY**
    shift
  **WHERE**
    grd1: 0 $<$ shift
    grd2:$\forall$ e$\cdot$e$\in$dom(rat) $\wedge$ rat(e)$\neq \varnothing \Rightarrow$shift$\leq$min(rat(e))
  **THEN**
    act1: rat(off) := { x| x+shift$\in$ rat(off)}
  **END**

**Figure 4. Events of the example model**

### 4.1 Example: a timer for light switch

Before the case study, we can consider an example with a light and a timer which switch off this light. Here we have only one timed event: $evts = \{off\}$. An user can push the button of the light and then we observe the **switch_on** event. Some delay after the light automatically turn off with the event **switch_off**. Thoses events can be seen in Fig. 4. The variables are $rat$ from the pattern and $light\_on$ for the state of the light.

The event **switch_on** observes the change on the system when somebody push the button, is so the light turn on (see act1) and a timer is launched (see act2) to switch off the light in $10\pm1$ units of time. This event **switch_on** refines the event **add** of the pattern. The added value $d$ to the variable $rat$ is choosen in the interval $9..11$ to represent the possible inaccuracy of the timer. In fact, in this simple example, act2 can be replaced by

$$\text{act2: } rat(off) := \{d\}$$

Since we can deduce $rat(off) = \varnothing$ from the guard grd2 of this event and the invariant inv3.

The event **switch_off** is triggered 10 units (more or less 1) of time after the **switch_on** event. This event refines the event **use** of the pattern. In the same way as previously, the line act1 of the guard can be simplified by

$$\text{act1: } rat(off) := \varnothing$$

Finally the event **tic** is given in a less general form. While $evts$ is a finite set, this formalisation can always be used.

The invariants of this model are:

$$\text{inv1: } rat \in evts \rightarrow \mathbb{P}(\mathbb{N})$$

$$\text{inv2: } light\_on \in BOOL$$

$$\text{inv3: } light\_on = FALSE \Leftrightarrow rat(off) = \varnothing$$

$$\text{inv4: } card(rat(off)) \leq 1$$

The invariant $inv1$ comes from the pattern; $inv2$ gives the type for $light\_on$ the state of the light, $BOOL$ is the set of boolean; $inv3$ states that the calendar for the event $off$ is empty iff the light is off; and with $inv4$ we can only have one value in the calendar of $off$.

### 4.2 Model-checking

In order to model-check our model we have some final considerations. In this work we did experimentation with ProB [17].

In the section 4 we give a general form of the substitution in the event **tic**. ProB can evaluate this general form but slowly, therefore we recommend to use the form given in the previous section. This change preserves the equivalence between the two pattern while for the next considerations the pattern for model-checking is only a refinement of the original pattern.

If all values of the range of $rat$ (a set of subset of $\mathbb{N}$) are empty, the value of the variable $shift$ of the event **tic** is not limited and the model-checker may loop while finding all possible values of the variable $shift$. But, in this case, the substitution done by the **tic** event does not have an effect because the sets of the range of $rat$ are empty. In other word this activation of the event **tic** represent the progression of the time without effect on the studied system. In the original pattern this progression was represented by the unlimited incrementation of the variable $now$ of the model.

We have, so far, 4 solutions to this problem.

1. We can add the folowing guard to the event **tic**:

   grd5: $\exists x \cdot x \in ran(rat) \land x \neq \varnothing$

   This guard just block the event in the problematic case.

2. Another possible solution is to limite the size of the variable $shift$ by a constant $c$:

   grd5': $shift \leq c$

3. As a sub-case of solution 2 we can choose $c = 1$. This value limits the number of **tic** transitions studied by the model-checker.

4. As in [13] we can also take $shift = min(rat(e))$ where $e \in dom(rat)$ and $rat(e) \neq \varnothing$. In this way the time jumps directly to the next scheduled event activation and of course the event **tic** is blocked when $rat(e) = \varnothing$ for all $e$. This solution works if all events are scheduled in the calendar, but if we still have event not constraints by real-time they are not taked into acount by the progession of the time.

Now if we want to verify, by model-checking, a model written with this pattern then the values inside the co-domain of $rat$ need to be finite. A good way to respect this is to bound every value added to $rat$. Therefore, we need the constraint:

grd3: $ntimer \leq m$

where $m \in \mathbb{N}$, which must hold in the guard of the event **add**. Then we have:

$$\forall e \cdot e \in dom(rat) \Rightarrow (\forall x \cdot x \in rat(e) \Rightarrow x \leq m)$$

and by the gluing invariant between $rat$ and $at$:

$$\forall e \cdot e \in dom(at) \Rightarrow (\forall x \cdot x \in at(e) \Rightarrow x - now \leq m)$$

Finally all the values of the co-domain of $rat$ are bounded naturals and while $evts$ is a finite set (see Section 2) then $rat$ has a finite number of values and a finite number of transitions can be obtained with the pattern event **use**. We also learn that, from the perspective of the first pattern, the system has a finite number of states if it uses only the timing values (of $at$) inside a "time window" from $now$ to $now + m$. In other words if the system is relative to the current time, and does not refer to an absolute time.

We see also that if nothing reacts to the time in the system then we do not care about the time progression. This real-time pattern respects this because the **tic** event has an effect on the variables only if $\exists x \cdot x \in ran(rat) \land x \neq \varnothing$. The previous version of **tic** does not respect this because it always increases the value of $now$.

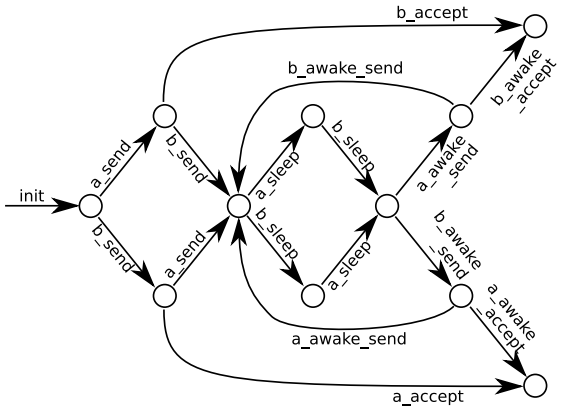


**Figure 5. Available events for the devices in the RCP case study**

## 5 Case study: IEEE 1394 Root Contention Protocol

We have applied our method, with success, on the IEEE 1394 Root Contention Protocol (RCP). In a previous paper [19] we have created a model of the RCP and verified it by theorem proving. We describe here the model-checking of this case-study with the method of this paper. The RCP is an election procedure between two devices with asynchronous communication. In order to elect a device, signals are sent on a bidirectional channel. Of course with asynchronous communication, signals can be sent and received in almost the same time. If a device receives a signal and has not yet sent a signal then it is elected. In the case where the signals from the two devices cross each other then it is not possible to elect a device, we call this situation the "contention". To resolve the contention, device choses randomly a delay between a short ($st$) and a long ($lt$), and re-send a signal after that delay. The development of RCP consists of 4 models link by 3 refinements in this order: m0, m1, m2,and m3. Each model of the development introduces new details: m0 specifies we want an election between two devices; m1 introduces device-states and communication-channels (without timing); m2 adds the propagation time ($prop$) of signals over channels; m3 adds a randomly chosen waiting delay.

In Fig. 5 we can see possible event activation of the model m1 labelled with the name of the corresponding events. In this figure only the events relevent for the two devices appear, in particular we also have in the model events which make signal progressing on the wires between devices. The device can **send** a signal and **accept** it. If the contention appears they start by **sleep** and wake up after the delay with either **awake_send** or **awake_accept**. Event-

names start with **a** or **b** to express which device observes the event. The possible traces present a cycle with two **sleep** events the **sleep** event for the device $a$) followed by two **awake_send** event the **awake_send** event for the device $a$). If we use the real-time pattern (section 2) this cycle is unfolded by the absolute time. If we use the real-time pattern with relative timing (section 3) this cycle is still present in the states reached by the model-checker.

In [19] we show, by theorem proving, safety properties of this model. The parameters of the model (called constants in Event-B) are $prop$, $st$, and $lt$ and verify the following axioms:

$$st \geq prop \times 2$$

and

$$lt \geq prop \times 2 + st - 1$$

We showed in particular that:

$$\begin{array}{ll} \forall x, y \cdot & x \in at(a\_awake) \wedge y \in at(b\_awake) \Rightarrow \\ & \textbf{if } a\_sleept \neq b\_sleept \quad \textbf{then } prop \leq |x - y| \\ & \qquad\qquad\qquad\quad \textbf{else } prop > |x - y| \end{array}$$

With $at(a\_awake)$ (respectively $at(b\_awake)$) the calendar of the events **a_awake_accept** and **a_awake_send** (**b_awake_accept** and **b_awake_send**), which expresses when the device **a** or **b** will wake up after the waiting time and re-try to send a signal or accept a signal; and with $a\_sleep$ (respectively $b\_sleep$) the chosen delay for **a** (**b**). This part of the invariant shows that if the two devices choose a different delay to re-send a signal then the propagation time of signals is smaller than the delay between the re-activation of the devices. Therefore the first device to wake up will have enough time to transmit its signal to the other device, and the protocol will work. Furthermore notice that this formula does not change if we would use $rat$ instead of $at$ because we take the difference between two values of $at(a\_awake)$ or $at(b\_awake)$.

In [19], we use an another form for the calendar $at$ or $rat$: instead we use several sub-sets of $\mathbb{N}$ specialised to our needs instead of the general function $at$. We call these sub-sets "at sets". For example, we use here $at(a\_awake)$ but in [19] we use $at\_a\_awake$. It is possible because with the B method we can employ data refinement to use a new version of a variable. Of course the equivalence between the abstract and concrete variable must be proven. Notice that several Event-B events can use (refine the event **use** of the pattern) one same event $e$ in the sens of a value inside an at set ($e \in dom(at)$ if we follow stricly the pattern). For example: the at set $at\_a\_awake$ can be used by the events **a_awake_send** or **a_awake_accept**.

Verification with the B Method is traditionally done by theorem proving, but recently a model-checker called ProB [17] became available. Therefore it is possible to check this case-study with our method. But with ProB it is not possible to do parametric model-checking, therefore we need to give a value (which verifies all hypotheses) to $prop$, $st$, and $lt$. We rewrote the model from the first pattern with absolute timing to the second pattern with relative timing. Finally we were able to check all models (invariant included) with ProB, Fig 6 and 7 give the reachable number of states for the model m2 and m3, and the used valuation of constants.

| $prop$ | reachable states |
|---|---|
| 1 | 25 |
| 2 | 51 |
| 3 | 81 |
| 4 | 117 |
| 5 | 159 |
| 6 | 207 |

**Figure 6. Number of reachable states: m2**

| $prop$ | $st$ | $lt$ | reachable states |
|---|---|---|---|
| 1 | 2 | 3 | 54 |
| 2 | 4 | 7 | 186 |
| 3 | 6 | 11 | 376 |
| 4 | 8 | 15 | 624 |
| 5 | 10 | 19 | 930 |
| 6 | 12 | 23 | 1294 |

**Figure 7. Number of reachable states: m3**

From [19] we can extract parts of the invariant of the models which shows that added values to the calendar have a bounded difference with the current time. As the system is symmetrical we show only invariants concerning the device $a$. As $at(a\_pass)$ represents the time of the reception of a signal, and signals take the propagation time $prop$ to progress in the channel, then this set is bound by $time + prop$:

$$\forall x \cdot (x \in at(a\_pass) \Rightarrow x \leq now + prop)$$

And we have a upper bound for the set $at(a\_awake)$:

$$\forall x \cdot (x \in at(a\_awake) \Rightarrow x \leq now + a\_sleept)$$

We can easly rewrite those formula with $at$ to a formula with $rat$, we just need to remove $now$.

Therefore, we could verify the invariant of the four models (m0 and m1 are trivially checked) over reachable states. It is also possible to check the properties of the constants $st$ and $lt$, if the properties are not well chosen then we can find counter-examples with the help of the invariant. In this way the model-checker is very useful to find errors before the proof process (while the proof verifies the models for any parameter values).

## 6 Conclusion

We have presented an approach to model and check real-time systems. Our approach can be used to check explicit

timed specifications with a generic model-checker.

An explicit timed specification used a non specialised formal method with ordinary variables to represent the time. The papers [16, 13] show how to model-check this kind of system with a variable call $now$ (or $time$) which model the current time. We have proposed a similar method in [10] but for theorem proving, and here we propose a new version of this pattern. The two methods and our later pattern differ on the formal language used, the discrete or continuous time, and with the means of expressing time-constraints (timer, timeout or event calendar). Here we use the event-B formal method, a discrete time, event-calendar and the ProB model-checker [17]. The two methods [16, 13] use a variable ($now$ or $time$) for the current time, and we show that if we remove this variable (and modify all time-constraints of the studied model to keep the behaviour) then the model-checking is easier. This kind of model-checking works nicely if one uses discrete time as show in [16] (where the symmetry under time translation simplifies the checking by adding a relation of equivalence over states) or with a continuous time but without continuous dynamics as in [13].

As most real-time systems do not depend on absolute timing, it is essential to use this property. We proposed an equivalent version of our pattern [10], this version allows one to model-check systems which depend only on timings relative to the current time. The proof of equivalence is shown and we report the use of the Rodin software [4] as a proof assistant in order to formally verify this proof. We explain what properties over the models written with our pattern are required to be able to use a model-checker over those models. With a discrete time those properties directly lead to a finite number of states (of course, if other untimed elements of the model are also model-checkable).

Our approach is suitable for model-checking models with our pattern of time and has been successfully applied to the case study describes in [19]. We will continue to explore how this idea can enhance verification of models, as in [16, 13]. For the use with the Event-B method, we propose to add a plug-in to the software Rodin [4] in order to automatically translate the models written with our formal pattern (which is suited to theorem proving).

## References


[1] Martín Abadi and Leslie Lamport. An old-fashined recipe for real-time. *ACM Trans. Program. Lang. Syst.*, 16(5):1543–1571, 1994.

[2] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, August 1996.

[3] Jean-Raymond Abrial. B$^{\#}$: Toward a synthesis between Z and B. In Didier Bert, Jonathan P. Bowen, Steve King, and Marina A. Waldén, editors, *ZB*, volume 2651 of *Lecture Notes in Computer Science*, pages 168–177. Springer, 2003.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An open extensible tool environment for event-b. In Zhiming Liu and Jifeng He, editors, *ICFEM*, volume 4260 of *Lecture Notes in Computer Science*, pages 588–605. Springer, 2006.

[5] Jean-Raymond Abrial and Dominique Cansell. Click'n prove: Interactive proofs within set theory. In David A. Basin and Burkhart Wolff, editors, *TPHOLs*, volume 2758 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2003.

[6] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. A mechanically proved and incremental development of ieee 1394 tree identify protocol. *Formal Asp. Comput.*, 14(3):215–227, 2003.

[7] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inform.*, 77(1-2):1–28, 2007.

[8] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

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

[10] Dominique Cansell, Dominique Méry, and Joris Rehm. Time constraint patterns for event B development. In Jacques Julliand and Olga Kouchnarenko, editors, *B*, volume 4355 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2007.

[11] Samuel Colin. *Contribution à l'intégration de temporalité au formalisme B : Utilisation du calcul des durées en tant que sémantique temporelle pour B*. PhD thesis, Université de Valenciennes et du Hainaut-Cambrésis, 2006.

[12] Samuel Colin, Georges Mariano, and Vincent Poirriez. Duration calculus: A real-time semantic for b. In Zhiming Liu and Keijiro Araki, editors, *ICTAC*, volume 3407 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 2004.

[13] Bruno Dutertre and Maria Sorea. Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In Yassine Lakhnech and Sergio Yovine, editors, *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2004.

[14] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998.

[15] Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[16] Leslie Lamport. Real-time model checking is really simple. In *Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2005.

[17] Michael Leuschel and Michael J. Butler. Prob: A model checker for b. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. Springer, 2003.

[18] Joël Ouaknine and Steve Schneider. Timed csp: A retrospective. *Electr. Notes Theor. Comput. Sci.*, 162:273–276, 2006.

[19] Joris Rehm and Dominique Cansell. Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In Frederic Boniol 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.

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

[21] Farn Wang. Formal verification of timed systems: a survey and perspective. *Proceedings of the IEEE*, 92(8):1283–1305, 2004.

[22] Tomi Westerlund and Juha Plosila. Time aware system refinement. *Electr. Notes Theor. Comput. Sci.*, 187:91–106, 2007.

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