Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method

Joris Rehm*,***, Dominique Cansell**,*** [1]

*Université Henri Poincaré Nancy 1
joris.rehm@loria.fr
**Université de Metz
cansell@loria.fr
***LORIA - BP 239 - 54506 Vandœvre-lès-Nancy - France

Abstract. We present a model of the IEEE 1394 Root Contention Protocol with a proof of Safety. This model has real-time properties which are expressed in the language of the event B method: first-order classical logic and set theory. Verification is done by proof using the event B method and its prover, we also have a way to model-check models. Refinement is used to describe the studied system at different levels of abstraction: first without time to fix the scheduling of events abstracly, and then with more and more time constraints.

1 Introduction

In this paper, we present a model of the IEEE 1394 Root Contention Protocol with a proof of safety and of real-time properties. We already described the pattern of our model of time, applied in a simple case study, in Cansell et al. (2007) as a pattern of refinement for the event B method. We show here how this pattern works over a proven development of the IEEE case study. Many different models for real-time already exist. Our goal is to find a model of time adapted to make proof by invariant with refinement over systems of events. We also argue that is better to start a proven development by an abstract model without time and to use refinement to add real-time properties. Therefore our model of time must allow us to use refinement.

The IEEE 1394, also known as FireWire, is used to connect devices like external hard-disks or movie cameras. Devices are able to configure themselves by the IEEE 1394 leader election protocol. This protocol takes the network as an acyclic graph and orients edges to obtain a spanning tree rooted by a leader. This general case is already done with the event B method in Abrial et al. (2003). This work extends this result to the following case: at the end of the algorithm, or when only two devices are connected, the general algorithm can fail. In this case the signals can cross in the bi-directional channel between the two devices if they send signals at almost the same send time. Consequently the IEEE 1394 Root Contention Protocol takes place in order to choose a leader between the two devices. The algorithm is probabilistic and uses a random choice between a short and a long waiting time. This sleeping time and signal sending between devices leads to a (probable) election. We can see this illustrated in

[1] This work was supported by grant No. ANR-06-SETI-015-03 awarded by the Agence Nationale de la Recherche.

Fig. 3 on page 9. We do not take into account probabilistic properties (we replace them by non-determinism) nor the loss of signals. To model this system we need to quantify the two different sleeping times and the progression of signals over the channels. We want to prove safety properties on this algorithm and we want to be compatible with the existing B model (Abrial et al. (2003)) in the general case. Furthermore, we want to use the language and tools of B without modifications.

The language of the B method is based on the first order classical logic with set theory. This method can be used to specifying, designing and coding software systems. B models of system are accompanied by mathematical proofs. Proofs validate an invariant over the events of the system and validate the refinement relation between models. The goal of the refinement is to connect an abstract specification to a more concrete model. And step by step we can reach a precise model of the implementation. A description of the event B method can be found in Cansell and Méry (2003) or a shorter introduction is present in the section 2 of Abrial et al. (2003). Of course, in this paper, every formal descriptions are followed by textual explanations.

The language of the B method does not contain specific real-time or distributed features but we can model them. The idea is to guard events with a time constraint like a timeout or an alarm. We say that events are linked to an "activation time" (AT). We have several sets of AT, one for each constraint timed event which corresponds to encoding a multiset of ATs.

To represent the real-time progression we use a global clock represented by the variable $time$. Our time is discrete so $time \in \mathbb{N}$, but we can use unknown constants or logical expressions between different times. The time progression is expressed by an event called $tick\_tock$. No events except $tick\_tock$ make the time progress therefore several events can trigger in the same clock granule. This event nondeterministicly increases the variable $time$ between $time + 1$ and the first activation time (if any). So we have in invariant: $(at \neq \varnothing \Rightarrow time \leq \min(at))$ where the variable at is the union of all different AT sets. As $time$ is a natural number we are sure that the system will reach the next active time if $tick\_tock$ is activated often enough. Finally, when time reaches an AT value we have $time \in at$; in other words, $time = min(at)$. Therefore the event linked to this AT can trigger, do its work and remove the reached AT from its AT set. After this suppression, $time$ is free to reach the next AT, or simply increase if $at = \varnothing$.

This paper is organised as follows. In Section 2, 3, 4 and 5 we show the model of the case-study. We follow the methodology of refinement and we introduce respectively: the goal of the system, the details of the system without time, the real-time properties of the signals passing and the real-time properties of the sleeping times. In Section 6, we show the verification of the model. Finally in Section 7, we show related-work and we conclude.

2 First Model

This first model is the most abstract specification of our system. The general behaviour is to choose (elect) one device in the set $N = \{a, b\}$. The only variable $leader$ is a subset of N and contains the chosen device (if not empty). Apart from this, the invariant states that the set $leader$ is $\{a\}$ or $\{b\}$ or $\varnothing$.

Finally the transitions of this abstract system are given by the event $accept$ in Fig. 1. This event occurs when the set $leader$ is empty and fills it with a device. All of the following models in the paper will refine this behaviour.

```
accept ≙
  ANY $x$ WHERE
    x ∈ N ∧
    $leader$ = ∅
  THEN
    leader := {x}
  END
```

FIG. 1 – *Accept event*

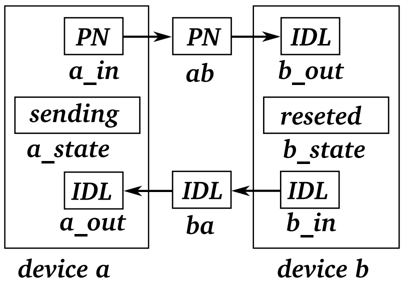

FIG. 2 – *Devices and channels*

3 First Refinement

We now introduce, through refinement, the local state of devices and two communication channels between device a and b. Almost all behaviour of the system can already be expressed abstractly at this level of abstraction. Communication will be asynchronous. In other words a signal from a to b can cross a signal from b to a. So the system can progress in two ways: if only one signal is sent, a leader will be elected; if two signals cross, a situation called "contention" will appear. In this situation the election is impossible, so the two devices will remove their signals. After a contention the devices wait for a random length of time before retrying the election process. We note that real-time properties will be added later and that model is a specification for the future real-time properties. For example, we will add a precise propagation time for the signal progress in a channel.

Devices communicate with two $SIGNALS$: IDL and PN. IDL is a blank signal. PN (for Parent Notify) means that the sender does not want to be the leader. We have four different $STATES$ for the two devices a and b, devices are in the state: $reset$ when they start; $sending$ when they are sending the signal PN; $sleeping$ when they wait after a contention; and $accepting$ when they accept to be the leader. In the refinement, the variable $leader$ disappears through data refinement and we add nine new variables: variables a_state and b_state for devices and three variables for each channels a_in, ab, b_out and b_in, ba, a_out. (The names of variables are chosen with the intuition that the signal go into the channel, not outside the devices). We need three variables for a channel because we want asynchronous communication and we can have a maximum of two changes of signals at the same time. The three variables of a channel act in a "first in first out" way and we have an event to make the values progress inside the channel from the input to the output. Finally, the variable $case$ does not take place in the behaviour of the system but is used to denote a special case in the invariant. We can see a graphical representation in fig. 2, which shows the very first sending of the signal PN.

3.1 Invariant

The first point to specify in an invariant is the type of the new variables: all variables of the channels are members of the set $SIGNALS$; Variables a_state and b_state are in $STATES$; and $case$ is a boolean.

In this refinement the variable $leader$ is not required anymore because we can deduce the leader of the election from the state of the devices. So we can replace the abstract variable $leader$ by the concrete variables a_state and b_state. We call this a "data refinement". For this we need a "gluing invariant" that relates the value of the abstract variable with the value of the concrete variables. Here we want $(leader = \{a\} \Leftrightarrow a\_state = accepting)$ and the same thing for b. We also know that if the PN signal is present in both channels then $leader = \varnothing$. Furthermore, if one device is accepting then the other is sending (the signal PN) and all signals have been received. (For device a: leader = {a} $\Rightarrow$ a_state = accepting $\wedge$ b_state = sending $\wedge$ ab = IDL $\wedge$ b_out = IDL $\wedge$ ba = PN $\wedge$ a_out = PN.)

This part of the invariant is the most important, but there are more things to express. First of all, in the initial state of the system devices are $reset$. With that condition all variables of the channel from this device are equal to IDL, for device a: a_state = reset $\Rightarrow$ (a_in = IDL $\wedge$ b_out = IDL $\wedge$ ab = IDL). When a device, for instance a, is sending then the beginning of the channel is set to PN, for instance $a\_in = PN$. We have the equivalence: $(a\_state = sending \Leftrightarrow a\_in = PN)$. We are also sure that if a device is reset and this device receives the PN signal then the other must be sending. In this case, the receiving device can safely accept to be the leader. Consider the case where the receiving device is also in the state $sending$: the election is now impossible and we are in a situation of "contention". The device discovers this situation and sets its state to $sleeping$. In this state we have for device a: $a\_in = IDL$ and $b\_state \in \{sending, sleeping\}$ (and the symmetric case for b: b_state = sleeping $\Rightarrow$ b_in = IDL $\wedge$ b_state $\in$ {sending, sleeping}). So the signal PN will be erased by IDL. After this, each device will go back to the $sending$ state and the previous part of the invariant is used to describe the state. The level of modelling is quite abstract and we will see in the final refinement many more (real-time) statements over states, especially in the situation of contention.

Finally the $case$ boolean variable is true if and only if: one of devices is sending and the other is sleeping: case = TRUE $\Rightarrow$ (a_state = sending $\wedge$ b_state = sleeping) $\vee$ (b_state = sending $\wedge$ a_state = sleeping); and the device currently sending was previously in the state $sleeping$: property ensure by the events.

The variable $case$ expresses this with relations between values of $case$ and values of other variables. This is the key to the algorithm because when a device goes back to the state $sending$ after $sleeping$, it can elect a leader if the duration of this case is long enough. We will see the utility of $case$ in the last refinement. In other words, if the second device awakes a long time after the first: then the first device has enough time to send their signal. Otherwise, the contention reappears.

3.2 Events

Events give properties over the transitions of the system. Here we have four kinds of events: $send$, $pass$, $accept$ and $sleep$. From now on we will describe the model only from the point of view of the device a. As the system is totally symmetrical between devices a and b, the reader can easily fill in the blanks.

```
init $\widehat{=}$ BEGIN
    $a\_in, b\_in, a\_out, b\_out, ab, ba :=$
      $IDL, IDL, IDL, IDL, IDL, IDL ||$
    $a\_state, b\_state := reset, reset ||$
    $case :=$ FALSE
  END;
a_send $\widehat{=}$ WHEN
    $a\_state = reset \wedge$
    $a\_out = IDL$
  THEN
    $a\_state := sending ||$
    $a\_in := PN ||$
    $ab := PN$
  END;
b_send $\widehat{=} \ldots$
ab_pass_out $\widehat{=}$ WHEN
    $ab \neq b\_out \wedge$
    $(b\_state \neq sending \vee b\_out \neq PN)$
  THEN
    $b\_out := ab ||$
    $ab := a\_in$
  END;
ba_pass_out $\widehat{=} \ldots$
pass_out $\widehat{=}$ WHEN
    $ab \neq b\_out \wedge$
    $ba \neq a\_out$
  THEN
    $b\_out := ab ||$
    $ab := a\_in ||$
    $a\_out := ba ||$
    $ba := b\_in$
  END;
a_accept $\widehat{=}$ REFINES accept WHEN
    $a\_state = reset \wedge$
    $a\_out = PN$
  THEN
    $a\_state := accepting$
  END;
b_accept $\widehat{=} \ldots$
a_sleep $\widehat{=}$ ANY $new\_ab$ WHERE
    $a\_state = sending \wedge$
    $a\_out = PN \wedge$
    $new\_ab \in SIGNALS \wedge$
    $(ab = b\_out \Rightarrow new\_ab = IDL) \wedge$
    $(ab \neq b\_out \Rightarrow new\_ab = PN)$
  THEN
    $a\_state := sleeping ||$
    $a\_in := IDL ||$
    $ab := new\_ab$
  END;
b_sleep $\widehat{=} \ldots$
a_awake_send $\widehat{=}$ WHEN
    $a\_state = sleeping \wedge$
    $a\_out = IDL \wedge$
    $ab = IDL \wedge$
    $b\_out = IDL$
  THEN
    $a\_state := sending ||$
    $a\_in := PN ||$
    $ab := PN ||$
    $case := \neg case$
  END;
b_awake_send $\widehat{=} \ldots$
a_awake_accept $\widehat{=}$ REFINES accept WHEN
    $a\_state = sleeping \wedge$
    $a\_out = PN \wedge$
    $b\_state = sending \wedge$
    $ab = IDL \wedge$
    $b\_out = IDL$
  THEN
    $a\_state := accepting ||$
    $case := \neg case$
  END;
b_awake_accept $\widehat{=} \ldots$
```

As written with the keyword “REFINES”, four events refine the abstract event $accept$.

The guard of the event a_send express that a has never sent anything and is not receiving a signal. With this condition we send a signal to the device b. The substitution: $ab := PN$ comes with a simplification in the use of the channel.

The ab_pass event shows how a signal progresses in the channel from a to b. We know that $(a\_in \neq ab \Rightarrow ab \neq b\_out)$. Therefore when there is at least one change of signal in the channel we know that $ab \neq b\_out$. Then we advance values from a_in to ab and from ab to b_out. However, when there are changes of signals in both channels we can make the values pass in both channels at the same time. This is done by the event $pass\_out$. Without this event, in this abstract model, one channel can take priority over the other, for example with several activations of ab_pass_out without activations of ba_pass_out. This is not realistic behaviour, but in model with real-time properties this problem is solved. In the last line of the guard of ab_pass_out we can see an expression of the priority of the event b_sleep. To express priority, we take the guard (or a crucial part of the guard) of the event with the higher priority and put its negation in the guard of the event with the lower priority. A part of the guard is crucial if it negation is enought to prevent the execution of the event in any case. Hence both events can not trigger in the same state. Without this priority the system could execute the sequence: a_send, b_send, $pass\_out$, a_sleep, ab_pass, a_accept. However, such a sequence is not allowed by the standard as a device has to have discovered the contention situation. In this sequence the device b failed to discover it (i.e. to execute b_sleep). In this refinement the problem is solved abstractly and in later models we will use real-time constraints.

The event a_accept triggers when the situation of contention never occurred (impossible when a state is still $reset$) and a device receives a signal PN.

In contrast, a device can discover a situation of contention when it is sending and it has received a signal PN. In this case it goes to sleep and starts to remove its signal PN. This is the only state where we can have two changes of signal in the same channel. It's achieved when we already have $ab \neq b\_out$. Therefore the new state of the channel will be $(a\_in = IDL \wedge ab = PN \wedge b\_out = IDL)$.

Finally, there are two events left: a_awake_send and a_awake_accept. They contain the same behaviour as a_send and a_accept plus the management of the variable $case$ and some extra conditions.

In addition to the guard of a_send, the event a_awake_send must check that the previous signal PN of device a is erased from the channel from a to b. This can be seen in the last two lines of the guard of a_awake_send.

In addition to the guard of a_accept, the event a_awake_accept must check that the signal is erased in the same way. It has also to check if the device b is actually in the $sending$ state. Otherwise one device would accept to be the leader with the other device in state $sleeping$ because $a\_out = PN$ does not imply that $b\_state = sending$; and it would lead to an incorrect election situation.

In conclusion, this model is enough to express all behaviours abstractly and now we can introduce the real-time.

4 Second Refinement

In this second refinement, we add a precise propagation time for the propagation of a signal inside a channel. This constant, called $prop$, is in $\mathbb{N}$ and is not equal to zero. This model contains three new variables: $time$, at_a_pass and at_b_pass. The variable $time$ is a global clock, the value of $time$ represents "now" the current time. The algorithm itself does not require a global clock or a synchronisation between devices. But in the B method we need closed systems, i.e. we need to model the environment. The two other variables are two sets of "Activation Time" (AT) i.e. a set of timeouts or alarms. Each AT set is linked to some particular event. Here the set at_a_pass (respectively at_b_pass) is linked to ab_pass_out (ba_pass_out). Therefore, they are both linked to $pass\_out$. The meaning is: the AT set contains the time in the future when the linked event will be triggered. Of course, other events can fill the AT set of another event. In a very natural way, we let the time progress in a non-deterministic way betwen $time + 1$ and the first AT. If there is no AT then time progression is not limited. When time reaches an activation time, the linked event can be triggered and we remove the AT from the set in the related event. With this model, we are sure that time is progressing, and every time constraint events linked to a AT set will also be triggered at the right moment. This model of time has already been described in our article Cansell et al. (2007).

4.1 Invariant

The typing of the new variables are ($time \in \mathbb{N}$) and ($at\_a\_pass \subseteq \mathbb{N}$) and ($at\_b\_pass \subseteq \mathbb{N}$). As the system is symmetrical we only show invariants concerning the device a. The time can not go beyond an activation time, as we don't want to miss the timeout of an event:

$at\_a\_pass \cup at\_b\_pass \neq \varnothing \Rightarrow time \leq \min(at\_a\_pass \cup at\_b\_pass)$

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 AT set is bound by $time + prop$:

$\forall x \cdot (x \in at\_a\_pass \Rightarrow x \leq time + prop)$

The set at_a_pass is finite and its cardinality reflects the number of signal changes travelling in the channel. In the computer model we use a formula with quantification instead of a cardinality because it is more convenient for the interactive proof.

$b\_in = ba \wedge ba = a\_out \Leftrightarrow at\_a\_pass = \varnothing$
$a\_in = ab \wedge ab \neq b\_out \Leftrightarrow card(at\_a\_pass) = 1$
$b\_in \neq ba \wedge ba \neq a\_out \Leftrightarrow card(at\_a\_pass) = 2$

A device can not start to send after the reception of a signal PN so we have:

$\forall (x, y) \cdot (x \in at\_a\_pass \wedge y \in at\_b\_pass \Rightarrow |x - y| < prop)$

If cardinality of at_b_pass is two, then the difference of members are strictly under $prop$ because they are both bound by $time + prop$ and because the $pass$ events have higher priority than $sleep$ events.

$\forall (x, y) \cdot (x \in at\_b\_pass \wedge y \in at\_b\_pass \Rightarrow |x - y| < prop)$

The time continues to progress after a sending and if contention is reached we have:

$b\_in = PN \wedge b\_out = PN \Rightarrow time + prop \notin at\_a\_pass$

The events $pass$ have higher priority than the events $send$:

$time \in at\_a\_pass \cup at\_b\_pass \Rightarrow time + prop \notin at\_a\_pass \cup at\_b\_pass$

This part allows us to prove the refinement of the event ab_pass_out:

$ab \neq b\_out \wedge time \in at\_b\_pass - at\_a\_pass \Rightarrow b\_state \neq sending \vee b\_out = IDL$

4.2 Events

In this refinement we have a new event $tick\_tock$. This event makes the time progress. In almost all guards, we add an extra clause to model the priority between events. The main reason for the use of priorities is the fact that the environment must act before the devices react. Here, the environment is the three $pass$ events, so we add $time \notin (at\_a\_pass \cup at\_b\_pass)$ in guards to let $pass$ events trigger before the other. Of course, between the three $pass$ events, the simultaneous passing event $pass\_out$ has higher priority. Finally, the real-time constraints model the propagation time of signals. In the next description of events, we show only the differences between events of the previous refinement. For that we mark new lines with a $\oplus$ and removed lines with a $\ominus$. All new lines of guards are connected with "$\wedge$" and lines of substitution with "$\|$".

```
init ≙ BEGIN
    $\oplus time := 0$
    $\oplus at\_a\_pass, at\_b\_pass := \varnothing, \varnothing$ END;
a_send ≙ WHEN
    $\oplus time \notin (at\_a\_pass \cup at\_b\_pass)$
  THEN
    $\oplus at\_b\_pass := at\_b\_pass \cup \{time+prop\}$
  END;
b_send ≙ ...
ab_pass_out ≙ WHEN
    $\ominus(b\_state \neq sending \vee b\_out \neq PN)$
    $\oplus time \in at\_b\_pass - at\_a\_pass$
  THEN
    $\oplus at\_b\_pass := at\_b\_pass - \{time\}$ END;
ba_pass_out ≙ ...
pass_out ≙ WHEN
    $\oplus time \in at\_a\_pass \cap at\_b\_pass$
  THEN
    $\oplus at\_a\_pass := at\_a\_pass - \{time\}$
    $\oplus at\_b\_pass := at\_b\_pass - \{time\}$ END;
a_accept ≙ WHEN
    $\oplus time \notin (at\_a\_pass \cup at\_b\_pass)$
  THEN ... END;
b_accept ≙ ...
a_sleep ≙ WHERE
    $\oplus time \notin (at\_a\_pass \cup at\_b\_pass)$
  THEN
    $\oplus at\_b\_pass := at\_b\_pass \cup \{time+prop\}$
  END;
b_sleep ≙ ...
a_awake_send ≙ WHEN
    $\oplus time \notin (at\_a\_pass \cup at\_b\_pass)$
  THEN
    $\oplus at\_b\_pass := at\_b\_pass \cup \{time+prop\}$
  END;
b_awake_send ≙ ...
a_awake_accept ≙ WHEN
    $\oplus time \notin (at\_a\_pass \cup at\_b\_pass) \wedge$
  THEN ... END;
b_awake_accept ≙ ...
tick_tock ≙
  ANY $tm$ WHERE
    $tm \in \mathbb{N} \wedge tm > time \wedge$
    $((at\_a\_pass \cup at\_b\_pass) \neq \varnothing \Rightarrow$
    $tm \leq \min(at\_a\_pass \cup at\_b\_pass)) \wedge$
    $(a\_state \neq sending \vee a\_out \neq PN) \wedge$
    $(b\_state \neq sending \vee b\_out \neq PN)$
  THEN $time := tm$ END;
```

5 Third Refinement

This final refinement removes all abstract conditions in the guards and adds the two different sleep times. As we have already explained devices try to go out of the situation of contention by waiting a random time between a short and a long time. So we have two new constants st (short time) and lt (long time) both in $\mathbb{N}$ and non-zero. Their values have the two following properties: $st \geq prop \times 2$ and $lt \geq prop \times 2 + st - 1$. Properties are chosen in order to leave enough time for the devices to react. The whole invariant of this paper is a proof of that. In this refinement we have four new variables: at_a_awake, at_b_awake, a_sleept, b_sleept. Variables a_sleept and b_sleept contain the chosen sleep time. We can see in the Fig. 3 a timeline showing a typical situation of contention with values $prop = 3$, $st = 6$ and $lt = 11$. The election will succeed if devices chose two different delays between st and lt. In that case, we can see in the Fig. 3 that the difference between the two awake times of devices will be enough to transmit a signal. The invariant discussed in the following section will express this formally.

5.1 Invariant

Both new AT sets at_a_awake and at_b_awake are a subset of $\mathbb{N}$. Values of variables a_sleept and b_sleept are in $\{st, lt\}$. We have the same property as in the previous invariant about $time$ and values of AT set: $time$ can not go after the first timeout. The new AT sets contain zero or one value:

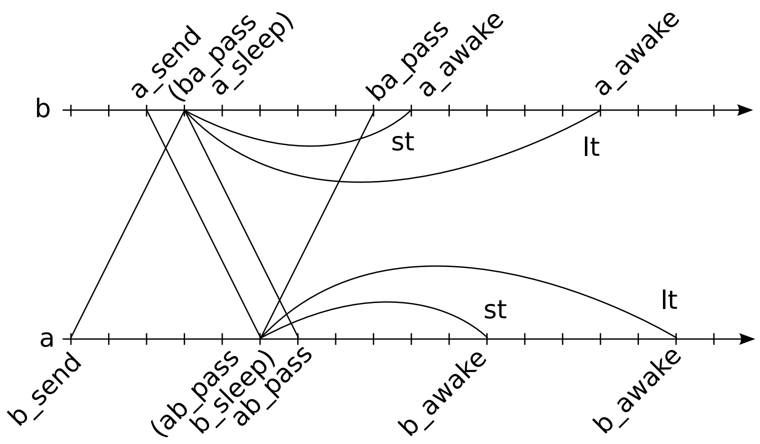

FIG. 3 – *Timeline*

$a\_state \neq sleeping \Leftrightarrow at\_a\_awake = \varnothing$
$a\_state = sleeping \Leftrightarrow card(at\_a\_awake) = 1$
We have a general upper bound for the new AT set and a special case:
$\forall x \cdot (x \in at\_a\_awake \Rightarrow x \leq time + a\_sleept)$
$(time \in at\_a\_awake \wedge a\_sleept = b\_sleept \Rightarrow$
$\forall x \cdot (x \in at\_b\_awake \Rightarrow x < time + prop))$
And we have several kinds of lower bound with conditions:
$(a\_in = PN \wedge (a\_out = PN \vee (a\_out = IDL \wedge ba = PN)) \Rightarrow$
$\forall x \cdot (x \in at\_b\_awake \Rightarrow time + b\_sleept - prop < x))$
$a\_in = PN \wedge a\_out = PN \Rightarrow \forall x \cdot (x \in at\_b\_awake \Rightarrow time + prop < x)$
$case = \text{FALSE} \wedge b\_state = sending \Rightarrow$
$\forall x \cdot (x \in at\_a\_awake \Rightarrow time + prop < x)$
$(time \in at\_a\_awake \wedge a\_sleept \neq b\_sleept \Rightarrow$
$\forall x \cdot (x \in at\_b\_awake \Rightarrow time + prop \leq x))$
This invariant restricts the possible values of at_a_awake:
$ab = PN \wedge b\_out = IDL \Rightarrow time + prop \notin at\_a\_awake$
Here we can see that, in two different cases, the signal is received before awake time:
$case = \text{FALSE} \Rightarrow \forall (x, y) \cdot (x \in at\_a\_pass \wedge y \in at\_a\_awake \Rightarrow x < y)$
$a\_sleept \neq b\_sleept \Rightarrow \forall (x, y) \cdot (x \in at\_a\_pass \wedge y \in at\_a\_awake \Rightarrow x \leq y)$
In the following conditions the awake time is before signal reception:
$(case = \text{TRUE} \wedge a\_sleept = b\_sleept \Rightarrow$
$\forall (x, y) \cdot (x \in at\_a\_awake \wedge y \in at\_a\_pass \Rightarrow x < y))$

After the discovering of the contention, device a erases its signal at a propagation time before awake time:
$\forall (x, y) \cdot (x \in at\_a\_pass \wedge y \in at\_b\_awake \Rightarrow x + prop \leq y)$
If chosen delays are equal, then devices do not have the time to transmit a signal:
$(a\_sleept = b\_sleept \Rightarrow$

$\forall(x,y)\cdot(x \in at\_a\_awake \wedge y \in at\_b\_awake \Rightarrow |x-y| < prop))$

If chosen delays are different, then devices have the time to transmit a signal. This formula shows why this algorithm works when chosen delays are different.

$(a\_sleept \neq b\_sleept \Rightarrow$
$\forall(x,y)\cdot(x \in at\_a\_awake \wedge y \in at\_b\_awake \Rightarrow prop \leq |x-y|))$

If the cardinality of at_a_pass is two then we have $b\_sleept - prop$ between awake time and the reception time of the first signal.

$(b\_in \neq ba \wedge ba \neq a\_out \Rightarrow$
$\forall x\cdot(x \in at\_b\_awake \Rightarrow \min(at\_a\_pass) + b\_sleept - prop < x))$

Finally, these formulae ensure the refinement of events a_awake_send and a_awake_accept:

$time \in at\_a\_awake \Rightarrow ab = IDL \wedge b\_out = IDL$
$time \in at\_a\_awake \wedge (a\_out = PN \vee ba = PN) \Rightarrow b\_state = sending$
$case = FALSE \wedge time \in at\_a\_awake \Rightarrow b\_state = sleeping$

5.2 Events

Again we show only the difference, we mark new lines with a $\oplus$ and removed lines with a $\ominus$. If an event is not present then it does not have any differences or it is symmetrical.

With the real-time properties of awake events we can remove all abstract conditions in guard of these events. The properties, expressed with the new AT sets, ensure this requirement as we can see in the last part of the invariant.

```
init ≙ BEGIN
    $\oplus at\_a\_awake, at\_b\_awake := \varnothing, \varnothing$
    $\oplus a\_sleept, b\_sleept := st, st$ END;
a_sleep ≙
  ANY $\oplus sleep$
  WHERE $\oplus sleep \in \{st, lt\}$
  THEN $\oplus at\_a\_awake :=$
        $at\_a\_awake \cup \{time + sleep\}$
    $\oplus a\_sleept := sleep$ END;
a_awake_send ≙ WHEN
    $\oplus time \in at\_a\_awake$
    $\ominus a\_state = sleeping$
    $\ominus ab = IDL$
    $\ominus b\_outs = IDL$
  THEN
    $\oplus at\_a\_awake := at\_a\_awake - \{time\}$
END;
a_awake_accept ≙ WHEN
    $\oplus time \in at\_a\_awake$
    $\ominus a\_state = sleeping$
    $\ominus b\_state = sending$
    $\ominus ab = IDL$
    $\ominus b\_out = IDL$
  THEN
    $\oplus at\_a\_awake := at\_a\_awake - \{time\}$
END;
tick_tock ≙ . . .
  “same pattern plus at_a_awake and at_b_awake”
```

6 Verification by Proof and Model-checking

For us, the primary way of verification is done by mechanical proof. But we also have used model-checking with ProB (Leuschel and Butler (2003)) in order to make partial verifications and to find counter-examples. In order to have a finite number of transitions with our models, we can not let the variable $time$ increase indefinitely. Therefore, we define another version for the event $tick\_tock$ for model-checking.

This version of the event $tick\_tock$ lets the variable $time$ always be zero but decreases the values inside the AT sets. With this version, the number of transitions of the system is finite, with a fixed value for $prop$. But as $time$ always equals zero and constants are valued, an invariant, determined to be correct with the model-checker, will not always be correct for the proof. In any case, model-checking provides a convenient way to discover invariants and to test them before the proof.

With the B prover the proof is cut into small "Proof Obligations" (PO). Some of those PO are automatically discharged and some need user interactions. All proofs of the first model are done automatically; for the first refinement 59 PO are interactive; for the second refinement 124; and for the last refinement 222. Of course all the proofs are done with the tool of the B method. As the model is symmetrical, the proof is also symmetrical, therefore, PO comes with a version for the device a and another similar version for the device b. Both versions are taken into consideration inside the number of PO. The first refinement is very easy to prove, the proofs of second and third refinement are short but numerous.

For the model-checking the first model and the second model are finite with 4 states and 24 states respectively. The next two models have only been validated with given valuations. For the second model, with $prop \in \{1, 2, ..6\}$ the number of states are 25, 51, 81, 117, 159 and 207. For the last model, with $(prop, st, lt) \in \{(1, 2, 3), (2, 4, 7), (3, 6, 11), (4, 8, 15), (5, 10, 19), (6, 12, 23)\}$ the number of states are 54, 186, 376, 624, 930 and 1294.

The procedure of proof, with an invariant, leads to the discovery of an invariant strong enough to be inductive. We can always start with a small invariant containing types of variables and some requirements for the refinements of data or events. If we start to do the proof with an invariant that is too weak then the proof will fail with an impossible interactive proof. With this failure we see the missing piece of information about the system state: we add it to the invariant and retry to prove.

In the case of real-time systems, we can split PO in two parts: PO coming from the event $tick\_tock$; and PO coming from all other events. The first kind of PO requires the invariant to be inductive with the progression of the time. The second requires the invariant to be inductive with the transitions of devices states. This leads to different kinds of proof and different constraints over the invariant but the two invariants are dependant.

7 Related-works and Conclusion

Many of other works about IEEE Root Contention Protocol (RCP) use model-checking over timed automata (Alur and Dill (1994)). We can find a comparative study of works about RCP in Stoelinga (2002), this work extends thoses results by another approach. Our approach of verification is primary focused on proof by invariant. It's clear that an interactive proof takes more time than verification by model-checker or proof with decision procedures. This is the price to pay for an expressive general language based on set theory. But the tool cuts the verification proof in small and quite easy parts. And we plan to work on the rules of the tool in order to reduce the number of interactive steps, as the proofs show a repetitive scheme a lot of improvements can be done. The idea of using a variable to model the time is shown in Abadi and Lamport (1994). Here the model of time (AT sets) is different.

With our method, one can model a real-time system within the B method. The refinement relation between models can be used in order to introduce time in a model and can be used again to add real-time properties step by step. If a concrete model with time refines an abstract model we can prove the timing validity of the concrete model. At every step of refinement, we can verify the model and its invariants by model-checking it first and then by a computer-assisted proof. This paper shows how our pattern of refinement of real-time constraints works on the IEEE 1394 RCP. The safety proof of RCP is done and we can see from the last invariant how the election works when chosen delays are different:

If chosen delays are different ($a\_sleept \neq b\_sleept$), then devices have the time to transmit a signal:

$\forall(x, y) \cdot (x \in at\_a\_awake \wedge y \in at\_b\_awake \Rightarrow prop \leq |x - y|))$

But if chosen delays are equal ($a\_sleept = b\_sleept$), then devices do not have enough time:

$\forall(x, y) \cdot (x \in at\_a\_awake \wedge y \in at\_b\_awake \Rightarrow |x - y| < prop))$

Where at_a_awake and at_b_awake are sets of natural and represent the time when devices will stop to wait. And $prop$ is the propagation time needed by the signal to pass from one device to another. Notice when devices are not in the state $sleeping$ and do not plan to awake then the corresponding set is empty.

Our method can be used without changes to the language of B and therefore we can extend existing results and developments in B with real-time. As the proofs about passing of time are particular, we could consider a way of handling this specificity. Our time is discrete but in our proof the most important property used is the order over natural numbers. The time model used involves a global time which interacts with a number of time of activation stored in several sets as in a global multi-set. As the studied algorithm does not require synchronisation, the global time is not a problem but we can think about localising this into several distributed clocks for other case-studies.

References

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

Abrial, J.-R., D. Cansell, and D. Méry (2003). A mechanically proved and incremental development of IEEE 1394 tree identify protocol. *Formal Asp. Comput. 14*(3), 215–227.

Alur, R. and D. L. Dill (1994). A theory of timed automata. *Theoretical Computer Science 126*(2), 183–235.

Cansell, D. and D. Méry (2003). Foundations of the B method. *Computers and Artificial Intelligence 22*(3).

Cansell, D., D. Méry, and J. Rehm (2007). Time constraint patterns for event b development. In *B 2007: Formal Specification and Development in B*, Volume 4355/2006, pp. 140–154. Springer.

Leuschel, M. and M. Butler (2003). ProB: A Model Checker for B. pp. 855–. Springer-Verlag, LNCS.

Stoelinga, M. (2002). Fun with FireWire: Experiments with verifying the ieee1394 root contention protocol. In J. R. S. Maharaj, C. Shankland (Ed.), *Formal Aspects of Computing*.