Electronic Notes in Theoretical Computer Science 65 No. 6 (2002)
URL: `http://www.elsevier.nl/locate/entcs/volume65.html` 18 pages

TPAP: an algebra of preemptive processes for verifying real-time systems with shared resources

Jérôme Ermont and Frédéric Boniol [1]

ONERA - CERT - DTIM
2 avenue Edouard Belin - BP4025
31055 Toulouse Cedex

Abstract

This paper describes a timed process algebra called TPAP. The aim of this algebra is to allow the modelisation of real time embedded processes sharing common resources, and which are sensitive to communication delays and scheduling strategies. Timed broadcasting and process preemption by interruption events are the two main fundamental notions of the algebra. They allow description of schedulers and asynchronous communication mediums, thus which can be taken into account when verifying the real time behaviour of the global system. We first present the process algebra and discuss its properties. A case study from the avionics area is then developed using TPAP, and formally verified by translation into the UPPAAL model checker.

1 Introduction

Design and mastery of embedded systems is a major challenge whose relevance has increased with the arrival of new generations airplane. Their complexity stems from their critical, real time and highly distributed and integrated nature. To overcome this complexity, aircraft manufacturers have chosen to build avionics systems on the federated architecture principle: avionics systems of modern aircrafts (such as Airbus A340/A330, Boeing B777, EuroFighter Aircraft, or Rafale...) are mainly composed of functions running in parallel, and communicating through multiplex data buses, and sharing at set of common computers. The functional architecture of such systems can be described by a general term like

$$S = (S_1 \| S_2 \cdots \| S_n)$$

[1] Email: `{ermont,boniol}@cert.fr`

In order to avoid single catastrophic failures, communications between avionics functions $S_1 \dots S_n$ are based on broadcasting (i.e., from one to n non blocking transmission) through shared asynchronous channels. Unfortunately, these channels can introduce non deterministic (but necessarily bounded) delays in the system. Furthermore, avionics functions may share common computers. Time on such computers is then divided into time slices controlled by one scheduler per computer. The underlying mechanism is preemption. According to the operating context and to the environment, each scheduler on each computer decides at any time which function (allocated on this computer) has to be executed and which function must be preempted. Preemption is then the main mechanism when dealing with real time embedded systems such as avionics systems. However, preemption actions can introduce non deterministic delays. Consequently to take into account the functional architecture of the system is necessary but not sufficient for analyzing the behavior of the global system. One has to consider also the scheduling strategy executed by each computer and the communication delays through the communication channels.

Several formalisms have been defined for verifying some properties on real-time systems. Timed automata [3] allow both to model communication and the behaviour of processes. Unfortunately, due to the richness of the automata expressivity, their behaviour is hard to control. Thus it is necessary to restrict this expressivity to the behaviours which are connected to the studied problem. So using a process algebra allows to describe more easily real-time systems we have to study and then to avoid timed automata problems.

Most of the real time process algebras [17,4,15,9,10] adequately capture delays due to process synchronization or action; however, they often abstract resource-specific details by assuming idealistic operating environment. The algebra proposed in this paper provides a formal framework that combines the areas of process algebra and real time scheduling, and thus can help to reason about systems that are sensitive to deadlines, process interaction, and resource availability.

To allow preemption mechanism, the process algebra ACSR [7] defined by P. Brémond-Grégoire and I. Lee uses a combination of priorities and timed actions which represent resource consuming. Thus if P and Q are two processes which use the resource r with respectively priority 2 and 3, then Q preempts P and is executed on resource r. Another process algebra defined by M. Buchholtz and al. [8] has been designed to model shared processors. This algebra is based on the fact that the time can evolve by two different ways: when using a processor or not. When a process is active (i.e. can be executed on a processor), the time passes through the processor on which the process is characterizing execution time. When a process is preempted, it lets the time to pass independently from the processor. Scheduling strategies are also implemented using priorities. The computation model of our algebra is based on the view that a real time embedded system consists of a set of communicating processes with the same priority, and which may be preempted when receiving a given message. The preemption mechanism thus formalized is very similar to the "abort" instruction of the reactive synchronous language ESTEREL [5]. The use of shared resources will be modeled by preemption of processes, and communication will be supported by non blocking broadcasting actions and timed delays. The execution of a process is then subject to interruptions sent by the scheduler controlling the resource it uses, i.e., is subject to the availability of the resource.

The rest of the paper is organized as follows. Section 2 describes the abstract syntax of the algebra. Some properties like deadlock freeness are discussed. A strong timed bisimulation and its equational laws are then proposed. Afterwards, section 3 presents an avionics case study formalized i, our algebra and verified by translation into the UPPAAL model checker [12,1].

2 Timed processes algebra with preemption (TPAP)

2.1 Syntax

Let $\mathcal{A}$ be a set of urgent actions representing event transmission. Elements of $\mathcal{A}$ are denoted by a, b, c... Let us consider $\overline{\mathcal{A}}$ as a set of co-names defined by $\overline{\mathcal{A}} = \{\overline{a} | a \in \mathcal{A}\}$. $\overline{a}$ in $\overline{\mathcal{A}}$ represents reception of event a. Let us consider also an invisible action ε such that $\varepsilon \notin \mathcal{A} \cup \overline{\mathcal{A}}$. ε will be used to denote an invisible action performed when breaking a delay. A set $\mathcal{V} = \{X, Y, \cdots\}$ of variables is used for recursive definitions. Finally the time-domain considered in our algebra is $\mathbb{R}_{\geq 0}$ and is denoted by $\mathcal{T}$.

Now we can define the expressions of the algebra $\mathcal{P}$, ranged over by P, Q..., by the following grammar:

$$P ::= \delta \mid X \mid aP \mid \overline{a}P \mid \mathcal{I}P \mid P + P \mid P \| P \mid recX.P \mid (P \uparrow \overline{a})P$$

where:

- The process δ can do nothing but only idling.
- Two types of actions are defined: actions $a \in \mathcal{A}$ representing event non blocking transmission, and actions $\overline{a} \in \overline{\mathcal{A}}$ representing event blocking reception. The process aP executes the urgent action a and behaves like P. The process $\overline{a}P$ will behave like P after receiving the event a. Else it idles.
- The process $P \| Q$ can perform actions of P and Q independently, or can synchronize on complementary actions (like a or $\overline{a}$), or can idle if P and Q are able to do so.
- The process $P + Q$ represents a choice between P and Q. A choice is not only made by actions but by time too. For instance, if time can progress in

P more than it can in Q then $P + Q$ may evolve like P.

- $\mathcal{I}$ is a closed time interval which denotes a non-deterministic delay with limits in $\mathcal{T} \cup \{\infty\}$. If the upper bound of $\mathcal{I}$ is non null, $\mathcal{I}P$ may also idle t time units provided that t less (or equal) than this upper bound. Furthermore, if its lower limit is zero, the process $\mathcal{I}P$ can execute the invisible action ε and behave like P.
- The process $(P \uparrow \overline{a})Q$ is a preemption process which behaves like P until event a is received, and then abandons P and becomes Q. Such an operation will be useful for describing the behaviour of processes sharing a same resource (see example in section 3.1). Let $R = (P \uparrow \overline{a})Q$, P is called the preempted subterm of R.
- Finally, the process $recX.P$ is the classical recursive definition which allows specification of infinite behaviours.

Definition 2.1 (Free Variables)
Let $P \in \mathcal{P}$. The set of free variables of P is called $free(P)$ and is defined by structural induction on $\mathcal{P}$:

$$\begin{aligned}
free(\delta) &= \emptyset \\
free(aP) &= free(P) \\
free(\overline{a}P) &= free(P) \\
free(\mathcal{I}P) &= free(P) \\
free(P+Q) &= free(P) \cup free(Q) \\
free(P\|Q) &= free(P) \cup free(Q) \\
free((P \uparrow \overline{a})Q) &= free(P) \cup free(Q) \\
free(X) &= \{X\} \\
free(recX.P) &= free(P) \backslash \{X\}
\end{aligned}$$

A term $P \in \mathcal{P}$ is closed if and only if $free(P) = \emptyset$. A term $P \in \mathcal{P}$ is regular if and only if all its parallel and preempted subterms are closed. Let $\mathcal{P}^r$ be the set of regular processes. For instance, $P = recX.(aX\|b\delta)$ and $Q = recX.(aX \uparrow \overline{b})\delta$ are considered to be not regular processes.

Let us now introduce the temporal upper and lower limits of a process.

Definition 2.2 (Upper and lower limits)

Let $\mathbf{U} : \mathcal{P} \to \mathcal{T} \cup \{\infty\}$ be the upper limit function defined by induction:

$$\begin{array}{ll}
\mathbf{U}(\delta) = \infty & \mathbf{U}(\mathcal{I}P) = p \; if \; \mathcal{I} = [m,p] \\
\mathbf{U}(aP) = 0 & \mathbf{U}(\overline{a}P) = \infty \\
\mathbf{U}(X) = 0 & \mathbf{U}(recX.P) = \mathbf{U}(P) \\
\mathbf{U}(P+Q) = \max(\mathbf{U}(P), \mathbf{U}(Q)) & \mathbf{U}(P\|Q) = \min(\mathbf{U}(P), \mathbf{U}(Q)) \\
\mathbf{U}((P \uparrow \overline{a})Q) = \mathbf{U}(P) &
\end{array}$$

Let $\mathbf{L} : \mathcal{P} \to \mathcal{T} \cup \{\infty\}$ be the lower limit function defined by induction:

$$\begin{array}{ll}
\mathbf{L}(\delta) = \infty & \mathbf{L}(\mathcal{I}P) = m \; if \; \mathcal{I} = [m,p] \\
\mathbf{L}(aP) = 0 & \mathbf{L}(\overline{a}P) = 0 \\
\mathbf{L}(X) = 0 & \mathbf{L}(recX.P) = \mathbf{L}(P) \\
\mathbf{L}(P+Q) = \min(\mathbf{L}(P), \mathbf{L}(Q)) & \mathbf{L}(P\|Q) = \min(\mathbf{L}(P), \mathbf{L}(Q)) \\
\mathbf{L}((P \uparrow \overline{a})Q) = \mathbf{L}(P) &
\end{array}$$

For a given process P in $\mathcal{P}^r$, $\mathbf{U}(P) = N$ means that P can idle during a duration t if and only if $t \leq N$. Furthermore $\mathbf{L}(P) = n$ means that P must idle during a duration n, i.e. no action is possible before n time units.

These two functions being defined, we can now introduce the notion of strongly well-timed processes [17], i.e. processes which cannot prevent time to diverge. To do so all executions of such processes must perform a non null minimal delay, i.e. some interval $\mathcal{I}$ with strictly positive lower bound. To determine if a process is strongly well-timed, we define the following predicate.

Definition 2.3 Let $swt : \mathcal{P} \times 2^{\mathcal{V} \times \{\mathbf{tt},\mathbf{ff}\}}$ be a predicate defined by induction:

$$\begin{array}{ll}
swt(\delta, C_{\mathcal{V},B}) & \\
swt(aP, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B}) \\
swt(\overline{a}P, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B}) \\
swt(\mathcal{I}P, C_{\mathcal{V},B}) & \Leftrightarrow (\mathbf{L}(\mathcal{I}P) > 0 \wedge swt(P, C_{\mathcal{V},\mathbf{tt}})) \vee swt(P, C_{\mathcal{V},B}) \\
swt(P+Q, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B}) \wedge swt(Q, C_{\mathcal{V},B}) \\
swt(P\|Q, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B}) \wedge swt(Q, C_{\mathcal{V},B}) \\
swt((P \uparrow \overline{a})Q, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B}) \wedge swt(Q, C_{\mathcal{V},B}) \\
swt(recX.P, C_{\mathcal{V},B}) & \Leftrightarrow swt(P, C_{\mathcal{V},B} \cup \{X, \mathbf{ff}\}) \\
swt(X, C_{\mathcal{V},B} \cup \{X, B_X\}) & \Leftrightarrow (B_X = \mathbf{tt})
\end{array}$$

with $C_{\mathcal{V},B} = \{(X,B) | X \in \mathcal{V} \; and \; B \in \{\mathbf{tt},\mathbf{ff}\}\}$ and $C_{\mathcal{V},\mathbf{tt}} = \{(X,B) \in C_{\mathcal{V},B} | B = \mathbf{tt}\}$

Let $\mathcal{P}^{swt} = \{P \in \mathcal{P} | swt(P, \emptyset)\}$ be the set of strongly well-timed processes.

For instance, let $P = recX.[0,2]aX$. By definition $swt(P, \emptyset) = \mathbf{ff}$, i.e. P is not strongly well-timed. It can perform an infinite number of action a within a finite duration. Conversely, process $Q = recX.[1,2]aX$ is strongly well-timed. It must idle at least one time unit between two consecutive actions.
Let $\mathcal{P}^{rswt}$ be the set of regular strongly well-timed processes, formally $\mathcal{P}^{rswt} = \mathcal{P}^r \cap \mathcal{P}^{swt}$. In the following, we only consider processes in $\mathcal{P}^{rswt}$ which will be renamed as TPAP.

2.2 Operational semantic

Elements of $\mathcal{T}$ are denoted by t. The operational semantic of TPAP is denoted by the labeled transition system $(\mathcal{P}^{rswt}, \rightarrow)$ where $\rightarrow \subseteq (\mathcal{P}^{rswt} \times \mathcal{A} \cup \mathcal{T} \cup \{\varepsilon\} \times \mathcal{P}^{rswt})$ defined in table 1. We write $P \xrightarrow{a} P'$ to mean that P may produce event a and in so doing become P'. In the same way, we write $P \xrightarrow{t} P'$ to mean that P may idle during t time units and in so doing become P'.
As explain in previous section, the process $\overline{a}P$ is waiting for an event a. Then it becomes P when receiving a from a parallel co-process. Else $\overline{a}P$ must idle. Consequently such a behaviour cannot be defined by a relation $\overline{a}P \xrightarrow{a} Q$. We have then to define an auxiliary relation $P \overset{a}{\rightsquigarrow} P'$ where $\rightsquigarrow \subseteq (\mathcal{P}^{rswt} \times \mathcal{A} \times \mathcal{P}^{rswt})$, in order to mean that P waits for event a and becomes P' when receiving a.

The main relation $\rightarrow$ and the auxiliary relation $\rightsquigarrow$ are completely defined by rules in table 1. However let us notice some semantical points:

- Rule 7, $\overline{a}P \overset{a}{\rightsquigarrow} P$, defines the semantics of the reception action: "the process $\overline{a}P$ waits for the event a and when receiving it behaves like P".
- The synchronization of two processes (rules 5c and 5d) is possible when one transmits an event (relation $\xrightarrow{a}$) and the other is waiting for this event (relation $\overset{a}{\rightsquigarrow}$).
- The rule 10c shows that communication is based on broadcasting: one transmission may be received by several processes at the same time.
- As said before, the choice may be made by time. Following rules 16b and 16c, if P may idle t time units and in so doing become P' and if Q cannot (because $t > \mathbf{U}(Q)$) then $P + Q$ may idle t time units and in so doing become P'.
- Invisible action ε is use to "cut" an interval. $\mathcal{I}P$ can perform ε at any time (ε is a persistent action) and in so doing become P if and only if the lower bound of $\mathcal{I}$ is 0 (rule 2).
- Finally, Rules 8 define the preemption mechanisms. Process $(P \uparrow \overline{a})Q$ can receive the event a and then behave like Q. However, if no preemption

$$(1)\ \frac{}{aP \xrightarrow{a} P}\ a \in \mathcal{A}$$

$$(2)\ \frac{}{[0,p]P \xrightarrow{\varepsilon} P} \qquad (3)\ \frac{P \xrightarrow{b} P'}{(P \uparrow \overline{a})Q \xrightarrow{b} (P' \uparrow \overline{a})Q}\ b \in \mathcal{A}$$

$$(4a)\ \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}\ a \in \mathcal{A} \cup \{\varepsilon\} \qquad (4b)\ \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$(5a)\ \frac{P \xrightarrow{a} P' \quad Q \overset{a}{\not\rightsquigarrow}}{P \| Q \xrightarrow{a} P' \| Q}\ a \in \mathcal{A} \cup \{\varepsilon\} \qquad (5b)\ \frac{P \overset{a}{\not\rightsquigarrow} \quad Q \xrightarrow{a} Q'}{P \| Q \xrightarrow{a} P \| Q'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$(5c)\ \frac{P \xrightarrow{a} P' \quad Q \overset{a}{\rightsquigarrow} Q'}{P \| Q \xrightarrow{a} P' \| Q'}\ a \in \mathcal{A} \qquad (5d)\ \frac{P \overset{a}{\rightsquigarrow} P' \quad Q \xrightarrow{a} Q'}{P \| Q \xrightarrow{a} P' \| Q'}\ a \in \mathcal{A}$$

$$(6)\ \frac{P[recX.P/X] \xrightarrow{a} P'}{recX.P \xrightarrow{a} P'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$(7)\ \frac{}{\overline{a}P \overset{a}{\rightsquigarrow} P}\ \overline{a} \in \overline{\mathcal{A}}$$

$$(8a)\ \frac{}{(P \uparrow \overline{a})Q \overset{a}{\rightsquigarrow} Q} \qquad (8b)\ \frac{P \overset{b}{\rightsquigarrow} P'}{(P \uparrow \overline{a})Q \overset{b}{\rightsquigarrow} (P' \uparrow \overline{a})Q}\ a \neq b$$

$$(9a)\ \frac{P \overset{a}{\rightsquigarrow} P'}{P + Q \overset{a}{\rightsquigarrow} P'} \qquad (9b)\ \frac{Q \overset{a}{\rightsquigarrow} Q'}{P + Q \overset{a}{\rightsquigarrow} Q'}$$

$$(10a)\ \frac{P \overset{a}{\rightsquigarrow} P' \quad Q \overset{a}{\not\rightsquigarrow} \quad Q \not\xrightarrow{a}}{P \| Q \overset{a}{\rightsquigarrow} P' \| Q} \qquad (10b)\ \frac{P \overset{a}{\not\rightsquigarrow} \quad P \not\xrightarrow{a} \quad Q \overset{a}{\rightsquigarrow} Q'}{P \| Q \overset{a}{\rightsquigarrow} P' \| Q}$$

$$(10c)\ \frac{P \overset{a}{\rightsquigarrow} P' \quad Q \overset{a}{\rightsquigarrow} Q'}{P \| Q \overset{a}{\rightsquigarrow} P' \| Q'} \qquad (11)\ \frac{P[recX.P/X] \overset{a}{\rightsquigarrow} P'}{recX.P \overset{a}{\rightsquigarrow} P'}$$

$$(12)\ \frac{}{\delta \xrightarrow{t} \delta}$$

$$(13)\ \frac{}{\overline{a}P \xrightarrow{t} \overline{a}P}$$

$$(14)\ \frac{}{[m,p]P \xrightarrow{t} [\max(0, m-t), p-t]P}\ t \leq p \qquad (15)\ \frac{P \xrightarrow{t} P'}{(P \uparrow \overline{a})Q \xrightarrow{t} (P' \uparrow \overline{a})Q}$$

$$(16a)\ \frac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \qquad (17)\ \frac{P \xrightarrow{t} P' \quad Q \xrightarrow{t} Q'}{P \| Q \xrightarrow{t} P' \| Q'}$$

$$(16b)\ \frac{P \xrightarrow{t} P'}{P + Q \xrightarrow{t} P'}\ t > \mathbf{U}(Q) \qquad (16c)\ \frac{Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} Q'}\ t > \mathbf{U}(P)$$

$$(18)\ \frac{P[recX.P/X] \xrightarrow{t} P'}{recX.P \xrightarrow{t} P'}$$

Table 1
Operational semantic rules

occurs, this process executes actions of P (rule 3) or lets time to progress if P is able to do so. This expression is similar to “abort p when a do q” of the untimed reactive language ESTEREL [5].

2.3 Properties

This section discusses some of the model properties defined in [16].

Time Determinism. The time determinism is satisfied if and only if the resulting process by time passing is uniquely determined.

Proposition 2.4 *TPAP is time deterministic, i.e.*

$$\forall P, P', P'', t : (P \xrightarrow{t} P' \land P \xrightarrow{t} P'') \Rightarrow P' = P''$$

Time additivity. The time additivity is satisfied if and only if: a process P can pass $t + t'$ time units to the process P' if and only if P can pass t time units to a process Q can pass t' time units to the process P'.

Proposition 2.5 *TPAP satisfies time additivity, i.e.*

$$\forall P, P', t, t' : (\exists P'' : P \xrightarrow{t} P'' \land P'' \xrightarrow{t'} P') \Leftrightarrow P \xrightarrow{t+t'} P'$$

Deadlock-freeness. Deadlock-freeness is satisfied if and only if any process can execute an action or pass time.

Proposition 2.6 *TPAP satisfies deadlock-freeness, i.e.*

$$\forall P \; \exists l \in \mathcal{A} \cup \{\varepsilon\} \cup \mathcal{T} \; \exists P' : P \xrightarrow{l} P'$$

Finite variability. Finite variability is satisfied if and only if any process can perform only finitely many actions in a finite time interval.

Proposition 2.7 *TPAP satisfies finite variability.*

Action urgency. Action urgency is satisfied if and only if there is at least a process which must execute an action $a \in \mathcal{A}$ without letting time to pass.

Proposition 2.8 *TPAP satisfies action urgency, i.e.*

$$\exists P, P' \in \mathcal{P}^{rswt}, a \in \mathcal{A}, \forall t \in \mathcal{T} : P \xrightarrow{a} P' \Rightarrow P \overset{t}{\not\rightarrow}$$

Persistency. Persistency is satisfied if and only if time progress cannot suppress the ability to perform an action, i.e.

$$\forall P, Q, P', t, a : P \xrightarrow{a} P' \land P \xrightarrow{t} Q \Rightarrow \exists P'' : Q \xrightarrow{a} P''$$

TPAP does not satisfy persistency. For instance, process $aP + [2,3]Q$ can execute a immediately. But it can also let 2 time units to pass and in so doing become $[0,1]Q$. Then it is not able to perform a.

Meanwhile, invisible action ε is used to "cut" time interval. An delayed process $[0,p]P$ can perform action ε or idle t time units ($t \leq p$) and in so doing become $[0, p-t]P$ which can still perform ε. We talk about ε-persistency, i.e.

$$\forall P, Q, P' : P \xrightarrow{\varepsilon} P' \land P \xrightarrow{t} Q \Rightarrow \exists P'' : Q \xrightarrow{\varepsilon} P''$$

2.4 Timed equivalences, congruence and equational laws

This section now introduces the notion of timed equivalence between processes. Definitions used here are related to works of Milner [14] and Larsen and Wang [13].

Definition 2.9 (Strong timed equivalence)
A binary relation $\mathcal{R}$ is a strong timed simulation if $\forall (P, Q) \in \mathcal{R} \Rightarrow \forall a \in \mathcal{A}, \forall t \in \mathcal{T}$:

(i) whenever $P \xrightarrow{a} P', \exists Q : Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{R}$
(ii) whenever $P \xrightarrow{\varepsilon} P', \exists Q : Q \xrightarrow{\varepsilon} Q'$ and $(P', Q') \in \mathcal{R}$
(iii) whenever $P \xrightarrow{t} P', \exists Q : Q \xrightarrow{t} Q'$ and $(P', Q') \in \mathcal{R}$
(iv) whenever $P \overset{a}{\rightsquigarrow} P', \exists Q : Q \overset{a}{\rightsquigarrow} Q'$ and $(P', Q') \in \mathcal{R}$

Symmetrically, the relation $\mathcal{R}$ is a strong timed bisimulation. Let $\sim$ be the largest strong timed bisimulation. $\sim$ is called strong timed equivalence.

$\sim$ is a congruence and can be defined by a complete and sound axiomatic given by table 2 (proofs of congruence, completeness and soundness are omitted in this paper).

Remark 1: as explained in [11] for a more simple real-time calculus no expansion theorem exists for TPAP, i.e. parallel composition can not be removed in general. This point may be explained by the use of time interval for modeling non-deterministic delay without explicit clock variables. An immediate consequence is that TPAP processes cannot be compiled into sequential terms without using global and explicit clocks.

Remark 2: in order to abstract the invisible action ε, a weak timed equivalence may be defined.

Definition 2.10 (Weak timed equivalence)
Let $\Rightarrow$ and $\Rrightarrow$ defined by:

- $P \overset{a}{\Rightarrow} P'$ if $P(\xrightarrow{\varepsilon})^* \xrightarrow{a} (\xrightarrow{\varepsilon})^* P'$
- $P \overset{t}{\Rightarrow} P'$ if $P(\xrightarrow{\varepsilon})^* \xrightarrow{t_1} (\xrightarrow{\varepsilon})^* \cdots (\xrightarrow{\varepsilon})^* \xrightarrow{t_n} (\xrightarrow{\varepsilon})^* P'$ with $t = \sum_{i \leq n} t_i$
- $P \overset{a}{\Rrightarrow} P'$ if $P(\xrightarrow{\varepsilon})^* \overset{a}{\rightsquigarrow} (\xrightarrow{\varepsilon})^* P'$

A binary relation $\mathcal{R}$ is a weak timed simulation if $\forall (P, Q) \in \mathcal{R} \Rightarrow \forall a \in \mathcal{A}, \forall t \in \mathcal{T}$:

(i) whenever $P \xrightarrow{a} P', \exists Q : Q \overset{a}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$
(ii) whenever $P \xrightarrow{t} P', \exists Q : Q \overset{t}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$
(iii) whenever $P \overset{a}{\rightsquigarrow} P', \exists Q : Q \overset{a}{\Rrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$

Symmetrically, $\mathcal{R}$ is a weak timed bisimulation. Let $\approx$ be the largest weak

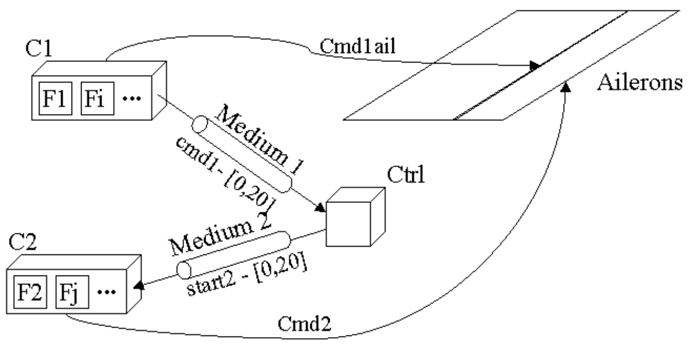

Fig. 1. A control surface system

timed bisimulation.

Unfortunately, $\approx$ is not a congruence for TPAP since $[0,0][0,2]Q \approx [0,2]Q$ but $[0,1]P + [0,0][0,2]Q \not\approx [0,1]P + [0,2]Q$.

3 Example and verification using UPPAAL

The aim of this real-time calculus with preemption introduced in the previous sections is to allow modelisation and verification of embedded systems composed of shared resources, schedulers, communication channels, . . . This section proposes a case study of such a system formally described in TPAP and verified by using the UPPAAL model checker. This case study was first explained (in French) in [6].

3.1 An avionics case study

Let us consider a subsystem, part of a whole avionics system, which controls the ailerons of a plane. This subsystem is pictured in figure 1. It is composed of:

- two shared computers C1 and C2, which executes all the on-board functions (F1, Fi, . . . , F2, Fj, . . .)
- two functions F1 and F2 (executed respectively by C1 and C2) which control the ailerons. F2 is the recovery function of F1 (i.e. which replaces F1 when F1 fails).
- a controller Ctrl, deciding at any time if F1 fails or not.
- two communication channels, Medium1 and Medium2 which transmit cmd1 and start2 signals from C1 to Ctrl and from Ctrl to C2. Transmission of each signal needs between 0 and 20 time units through each channel.

In normal mode, F1 is active and F2 is passive (is not executed). F1 sends periodically a command “cmd1ail” to the ailerons and a message “cmd1” to

$(+1)$	$P + Q \equiv Q + P$	
$(+2)$	$P + (Q + R) \equiv (P + Q) + R$	
$(+3)$	$P + P \equiv P$	
$(\uparrow_1)$	$(\delta \uparrow \overline{a})P \equiv \overline{a}P$	
$(\uparrow_2)$	$(\overline{a}P \uparrow \overline{a})Q \equiv \overline{a}Q$	
$(\uparrow_3)$	$((P + Q) \uparrow \overline{a})R \equiv (P \uparrow \overline{a})R + (Q \uparrow \overline{a})R$	
$(\|_1)$	$P\|\delta \equiv P$	
$(\|_2)$	$P\|Q \equiv Q\|P$	
$(\|_3)$	$(P\|Q)\|R \equiv P\|(Q\|R)$	
$(\|_4)$	$(P + Q)\|R \equiv P\|R + Q\|R$	
$(\|_5)$	$aP\|bQ \equiv a(P\|bQ) + b(aP\|Q)$	
$(\|_6)$	$aP\|\overline{a}Q \equiv a(P\|Q)$	
$(\|_7)$	$\overline{a}P\|\overline{a}Q \equiv \overline{a}(P\|Q)$	
$(\|_8)$	$[0,m]P\|aQ \equiv a([0,m]P\|Q) + [0,0](P\|aQ)$	
$(\|_9)$	$[n,n]P\|[n,m]Q \equiv [n,n]([0,0]P\|Q) + [n,n](P\|[0,m-n]Q)$	
$(\|_{10})$	$[n,n]P\|[n+m,n+p]Q \equiv [n,n](P\|[m,p]Q)$	$m > 0$
$(\|_{11})$	$\mathcal{I}P\|aQ \equiv a(\mathcal{I}P\|Q)$	if $\mathbf{L}(\mathcal{I}P) > 0$
$(\|_{12})$	$aP\|(bQ \uparrow \overline{a})R \equiv b(aP\|(Q \uparrow \overline{a})R) + (a(P\|R)$	
$(\|_{13})$	$aP\|([0,m]Q \uparrow \overline{a})R \equiv [0,0](aP\|(Q \uparrow \overline{a})R) + a(P\|R)$	
$(\|_{14})$	$aP\|(\mathcal{I}Q \uparrow \overline{a})R \equiv (a(P\|R)$	if $\mathbf{L}(\mathcal{I}P) > 0$
$(recX_1)$	$recX.P \equiv P[recX.P/X]$	
$(recX_2)$	if $P[Q/X] \equiv Q$ then $Q \equiv recX.P$	

Table 2
TPAP Axiomatic

Ctrl via Medium1. If F1 fails, no more occurrence of "cmd1" is sent to Ctrl. After waiting 70 time units, Ctrl sends a "start2" signal to C2. Then F2 becomes active and sends periodically "cmd2" to the ailerons. The system is then in failure mode. To simplify, we consider that only C1 can fail (Medium1, Medium2 and Ctrl are supposed to be reliable).

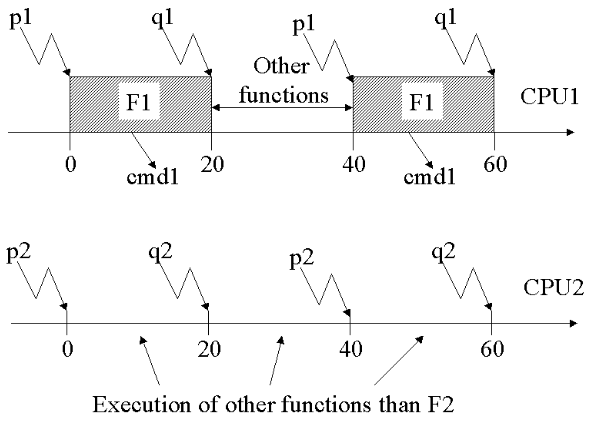

Fig. 2. Time behaviours of CPU1 and CPU2 in normal mode

Executions on each computer are controlled by a scheduler. These schedulers send periodically (every 20ms) signals to CPUs (p1 and q1 to CPU1) indicating which function has to be executed. We assume that there are two time slices on each CPU. F1 (in normal mode) and F2 (in failure mode) are executed in the first time slice of each CPU. Other functions (not involved in the ailerons control and then not considered in the modelisation) are executed in the second time slices. Figure 2 shows the behaviours of CPU1 and CPU2 in normal mode.

3.2 Modelisation with TPAP

According to the previous description the system under consideration is composed of seven processes which can be formally defined in TPAP.

F1 and F2 F1 sends "cmd1" and "cmd1ail" and idles. F2 sends "cmd2" and idles. These behaviours may be defined by the two following expressions:

$$F1 = cmd1\ cmd1ail\ \delta$$

$$F2 = cmd2\ \delta$$

Ctrl As said, Ctrl waits for the signal "cmd1ctrl" from Medium1. However if it does not receive any occurrence of "cmd1" for 70 time units, then it sends "start2":

$$Ctrl = recX.(\overline{cmd1ctrl}X + \overline{timeout}\ start2\ \delta)\|$$
$$recX.(([70,70]timeout\ \delta) \uparrow \overline{cmd1ctrl})X$$

Schedulers Before defining the two CPUs, we have to formalize the behaviours of the two schedulers. Each scheduler sends interruptions to CPUs every 20 time units: “qi” (respectively “pi”) to interrupt Fi (respectively the other functions) on Ci.

$$Sch1 = recX.p1[20,20]q1[20,20]X$$

$$Sch2 = recX.p2[20,20]q2[20,20]X$$

CPU1 In normal mode, CPU1 executes a sequence T1 which alternates F1 and other functions when receiving signals “p1” and “q1”. As we are not interested in the behaviour of the other functions, we abstract them by the idle process. In the failure mode, CPU1 does nothing and is abstracted by the idle process too. In order to simplify the modelisation, the transition from normal mode to failure mode is denoted by a single signal “fail” which can be sent at any time by an external failure process:

$$Failure = [0,\infty)fail\ \delta$$

$$T1 = \overline{p1}\ recX.(F1 \uparrow \overline{q1})(\delta \uparrow \overline{p1})X$$

$$CPU1 = (T1 \uparrow \overline{fail})\delta$$

CPU2 In normal mode, CPU2 executes a sequence T2 which alternates functions not involved in the ailerons control system when receiving signals “p2” and “q2”. These functions are abstracted by the idle process. In the failure mode, i.e. when receiving event “start2”, CPU2 executes a new sequence T2’ which alternates F2 and other functions:

$$T2 = recX.(\delta \uparrow \overline{q2})(\delta \uparrow \overline{p2})X$$

$$T2' = recX.(F2 \uparrow \overline{q2})(\delta \uparrow \overline{p2})X$$

$$CPU2 = (T2 \uparrow \overline{start2})T2'$$

Mediums To model Medium1 and Medium2, we use parallel composition of several simple channels as shown by Milner [14]. Whenever simple channel receives a signal, it transmits it after waiting a non-deterministic time $t \in [0,20]$. Since F1 sends signals every 20 time units and since Ctrl sends signals at least every 50 time units then only two simple channels are necessary per medium. We can then define formally the two mediums by:

$$M1 = recX.\overline{cmd1}\ [0,20]\ cmd1ctrl\ X$$

$$M2 = recX.\overline{start2ctrl}\ [0,20]\ start2\ X$$

$$Medium1 = M1 \| M1$$

$$Medium2 = M2 \| M2$$

Finally, the whole system is defined by the composition of the previous processes.

$$System = (CPU1\|CPU2\|Sch1\|Sch2\|Ctrl\|Medium1\|Medium2)$$

The problem is then to verify two safety properties (see next section) on this expression when assuming only F1 can fail. The algebraic expression under consideration is then $System' = (System \| Failure)$.

3.3 Verification using UPPAAL

To make verification using UPPAAL, each term of TPAP has to be translated into timed automata ([3]) used by UPPAAL.

Definition 3.1 (Timed automaton)
A timed automaton is a tuple $A =< \mathcal{N}, n_0, \mathcal{E}, \mathcal{C}, I >$ where

- $\mathcal{N}$ is a set of nodes
- n_0 is the initial node (denoted graphically by double circles)
- $\mathcal{C}$ is the set of clocks
- I is the invariant function which associates an invariant at each node
- $\mathcal{E}$ is a set of edges, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N} \times Act \times 2^{\mathcal{C}} \times \mathcal{G}$.
  Let $e =< s, t, a, r_t, g_t > \in \mathcal{E}$.
  · s, t represent the source and target of the edge e.
  · a is the action executed by e (a! stands for emission of a and a? stands for reception of a).
  · r_t is the set of clock resets.
  · g_t is the set of guards.

The general translation rules from TPAP to UPPAAL timed automata are not given here (but will detailed in the long version of this paper). The timed automata corresponding to the TPAP processes defined in the previous subsection are pictured in figures 3, 4 and 5.

The ailerons control system is assumed to satisfy two safety properties:

- the aileron should receive a command before 120 time units;
- F2 should work only if F1 does not.

To express these properties, we use the TCTL temporal logic [2]:

$$\varphi_1 = \forall\Box(\forall\Diamond_{\leq 120}(cmd1 \vee cmd2))$$

$$\varphi_2 = \forall\Box(cmd2 \rightarrow \forall\Box\neg cmd1)$$

In order to verify these properties φ_1 and φ_2 are translated into timed automata pictured in figure 6. The final verification consists then in proving that the “unhappy” states (see figure 6) are unreachable. This verification, realized with UPPAAL v3.2.1 on a Sun Ultrasparc 10 with 256 Mo memory needs less than one second for φ_1 and φ_2.

4 Conclusion

We have described a time process algebra called TPAP that supports broadcasting and preemption in a dense time context. The aim of this formal framework is to allow the modelisation and afterward the verification of real time

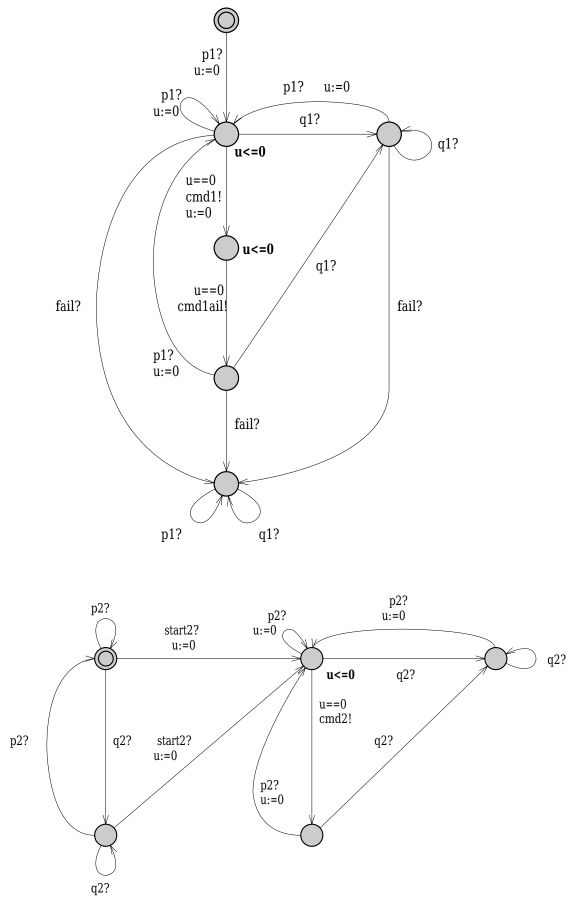

Fig. 3. Automata of CPU1 (up) and CPU2 (down)

embedded systems composed of loosely coupled communicating processes sensitive to scheduling strategies on shared computing resources. Such scheduling strategies may be modeled by preemption operations. We have then developed a small case study translated into UPPAAL timed automata. Verification of reachability properties by model checking is then available.

However, one of the main lack of our approach is the impossibility to model temporary preemption, i.e. suspension. The preemption mechanism $(P \uparrow a)Q$ leads to abort P when receiving event a. In that sense, the semantics of such expression is similar to the "control C" Unix command. However, real time operating systems often offer suspension mechanism (similar to the "control Z" Unix command), i.e. suspension of activity without losing the current state of the process. Abortion and suspension are the two fundamental kinds of preemption necessary to model concrete real time systems. Consequently, the main perspective of this work is to extend TPAP with a suspension operator, and to translate it into timed automata.

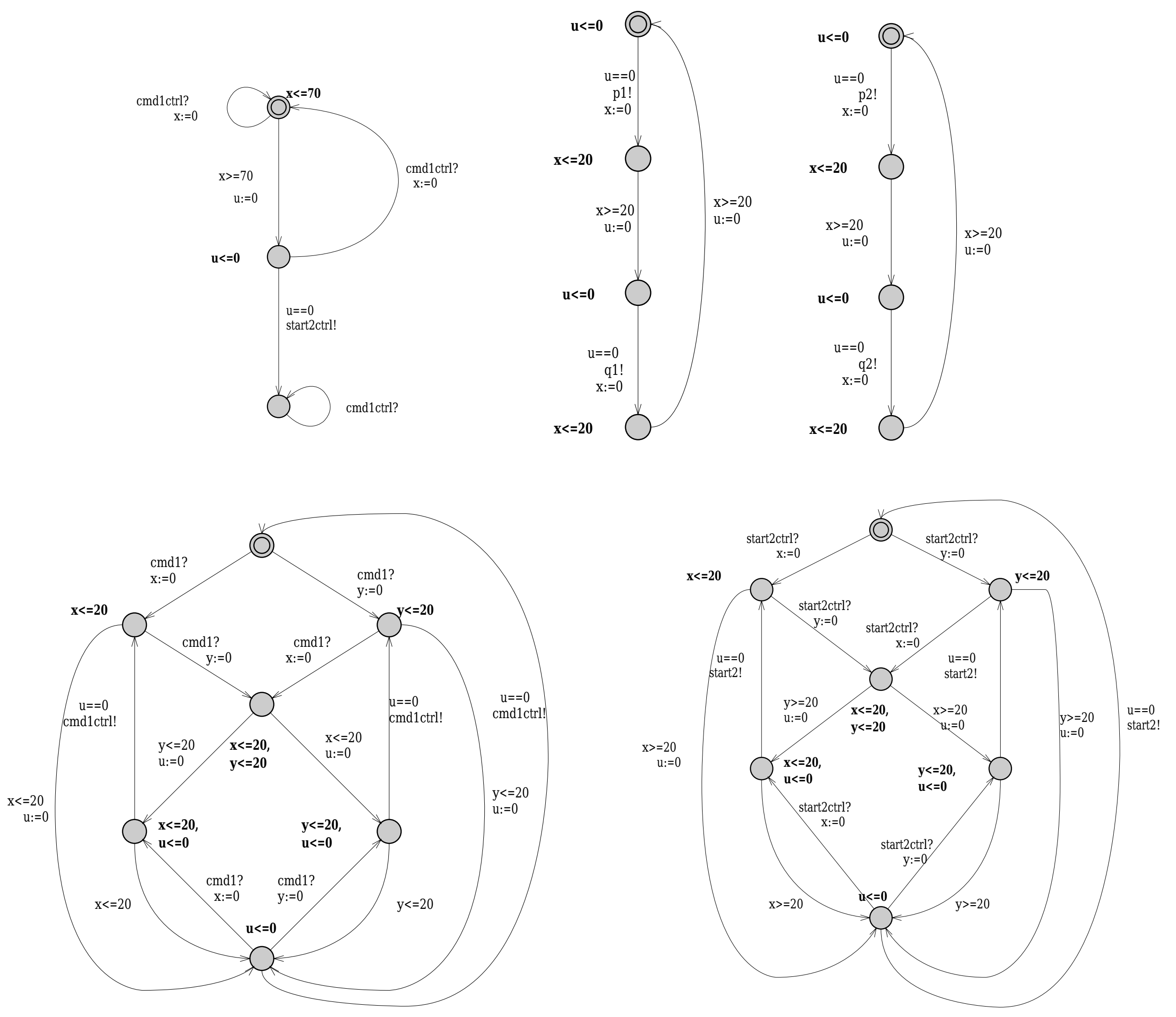

Fig. 4. Ctrl automaton (up - left), Sch1 and Sch2 automata (up - center and right) and mediums automata (down)

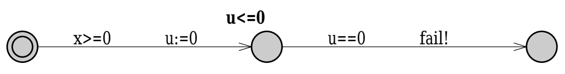

Fig. 5. The failure automata

References

[1] http://www.uppaal.com.

[2] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *Proceedings of the 5th annual IEEE Symposium on Logic in*

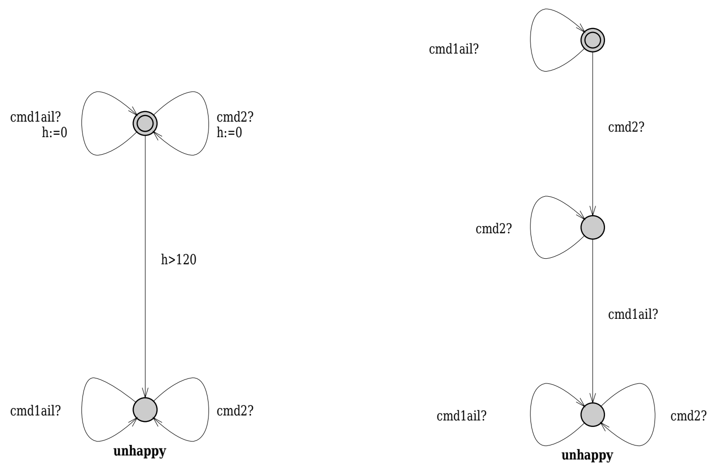

Fig. 6. Properties automata

Computer Science, Philadelphia, Pensylvania, 1990. IEEE Computer Society Press.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] J.C.M. Baeten and J.A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[5] Gerard Berry. Preemption in concurrent systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 72–93, 1993.

[6] F. Boniol, G. Bel, and J. Ermont. Modélisation et vérification de propriétés temps réel dans une architecture informatique distribuée asynchrone : étude de cas et approche comparative. In *9ème conférence internationale sur les systèmes temps réels RTS2001*, mars 2001.

[7] P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189, 1997.

[8] M. Buchholtz, J. Andersen, and H.H. Lovengreen. Towards a process algebra for shared resources. In *2nd International Workshop on Models for Time-Critical Systems*, Aalborg, Danemark, August 2001.

[9] P. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, Department of Computer Science, University of Twente, 1999.

[10] Harald Fecher. A real-time process algebra with open intervals and maximal progress. *Nordic Journal of Computing*, 8(3):346–365, 2001.

[11] J. Godskesen and K. Larsen. Real-time calculi and expansion theorems. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science*, volume 652, pages 302–315. Springer Verlag, 1992.

[12] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[13] Kim Guldstrand Larsen and Yi Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1997.

[14] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[15] C. Tofts F. Moller. A temporal calculus of communicating systems. In *Proceedings of Concur'90, Lecture Notes in Computer Science*, volume 458, pages 401–415. Springer-Verlag, 1990.

[16] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *3rd. Computer-Aided Verification*, pages 376–398. LNCS 575, Springer-Verlag, July 1991.

[17] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.