An Overview and Synthesis on Timed Process Algebras*

Xavier Nicollin Joseph Sifakis

Laboratoire de Génie Informatique
IMAG-Campus B.P. 53X
38041 Grenoble Cedex - FRANCE

Abstract. We present an overview and synthesis of existing results about process algebras for the specification and analysis of timed systems. The motivation is double: present an overview of some relevant and representative approaches and suggest a unifying framework for them.

After presenting fundamental assumptions about timed systems and the nature of abstract time, we propose a general model for them: transition systems whose labels are either elements of a vocabulary of actions or elements of a *time domain.* Many properties of this model are studied concerning their impact on description capabilities and on realisability issues.

An overview of the language features of the process algebras considered is presented, by focusing on constructs used to express time constraints. The presentation is organised as an exercise of building a timed process algebra from a standard process algebra for untimed systems. The overview is completed by a discussion about description capabilities according to semantic and pragmatic criteria.

Keywords: real-time, specification of timed systems, process algebras

Contents

1 Introduction

The paper presents an overview and synthesis of existing results about process algebras for the specification and analysis of timed systems. It has been motivated both by the

*Work supported by the ESPRIT BRA SPEC

drastically increasing number of contributions in the area and by the authors' conviction that most of the existing work admits a unifying common framework. Thus, the motivation is double: first, the presentation of an overview of some relevant and representative approaches in the area and second, the proposal of a framework for these approaches. The paper presents the rather incomplete and eventually biased authors' point of view than a survey of existing work in the area.

Although emphasis is put on algebraic behavioural specification formalisms, we believe that most of the ideas presented here have a more general applicability scope, as they are independent of the features and the nature of the description formalism considered. For instance, general ideas about the nature of time and the underlying model of timed systems may be used when designing logical specification languages; the results on process algebras can be easily transposed on other behavioural specification formalisms like automata, timed graphs, timed transition systems, etc.

A timed system is usually considered to be a system with a global parameter (state variable) called time, used to constrain the occurrences of the actions. Introducing time requires consistent assumptions about its progress with respect to the evolution of the system: correspondence between instants (domain of definition of the time parameter) and action occurrences, duration of the actions.

Most of the existing description formalisms for timed systems adopt implicitly the following view concerning their functioning:

- A timed system is the composition of cooperating sequential components (processes). Each component has a state variable defined on an appropriate time domain D with a binary operation $+$ which has essentially the properties of addition on non negative numbers. A component may modify its state either by executing some (atomic) action or by increasing its time variable (letting time progress).
- System time progresses synchronously in all processes, i.e., from a given global state, time increases by a quantity d if all the components accept to do so.
- An execution sequence is a sequence of two-phase steps: In the first phase φ_1 of a step, components may execute, either independently or in cooperation, a finite though arbitrarily long sequence of actions. In the second phase φ_2, components coordinate to let time progress by some finite or infinite amount. A new step begins when the second phase terminates. Figure 1 illustrates this principle for two interacting processes.

The functioning described combines both synchronous and asynchronous cooperation in two alternating phases: one where all the components agree for the time to progress, and an eventually terminating asynchronous computation phase during which the progress of time is blocked.

Most modes of cooperation of concurrent systems can be obtained by simplifying this functioning scheme. In fact, in the so called asynchronous cooperation only the action execution phase exists. In synchronous languages like Lustre [CHPP87], Esterel [BC85]

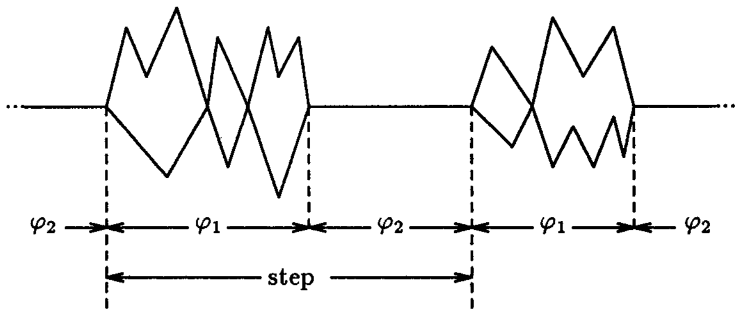

Figure 1: two-phase functioning schema

and the StateCharts [Har87], a step corresponds implicitly to one time unit and only the final state reached at the end of an asynchronous computation phase can be observed. This state is obtained by composing the effects of the actions (microsteps in the current terminology) and its computation raises some well-known causality problems. The so called synchronous cooperation, encountered in process algebras like SCCS [Mil83], CIRCAL [Mil91] and Meije [AB84], corresponds to a particular case of this functioning, where in addition, a process cannot perform more than one action in a step.

Such a mode of two-phase functioning is quite appropriate and natural for modelling reactive systems. For instance, the functioning of hardware and of systems for real-time control ideally follows this principle: a phase of asynchronous evolution is followed by a phase in which conceptually time progresses.

In a recent paper [NSY91], it is proposed a model for hybrid systems which adopts such a two-phase functioning principle. The phase where actions — "instantaneous" discrete changes of the state space — are executed is followed by a phase where state is transformed according to a law depending on time progress.

Considering such a mode of functioning allows to correlate the speeds of a system's components, as the flow of asynchronous computation can be cut by time progress phases in some appropriate manner. Furthermore, it introduces a concept of duration for an execution step and allows to assign durations to sequences of actions.

One might object that this two phase functioning assumption cannot faithfully model real systems where actions always take some non-zero time. In fact, direct consequences of this assumption are the following:

- Atomic actions take no time. This simplifies theoretical development and does not go against generality as non atomic actions can be modelled by sequences of atomic ones. It has been advantageously adopted by programming languages like Esterel [BC85].
- The time considered is *abstract* in the sense that it is used as a parameter to express constraints about instants of occurrences of actions. The implementability of such

constraints taking into account speeds and execution times of processors, is a separate though not independent issue. This distinction between abstract and concrete or physical time is an important one. It allows simplifications that are convenient at conceptual level as it leads to simpler and more tractable models. For instance, the assumption that an action may take zero time, though not realistic for physical time, is quite convenient for abstract time. Of course, such an abstraction should take into account realisability issues by integrating requirements for safe implementations. For instance, eventual termination of the asynchronous computation phase is such a requirement; for a correct implementation it should be possible to determine the clock period as the upper bound of step durations computed so as to take into account execution time of sequences of ideally zero time actions.

It has been often argued that models where any action takes some non zero time — its execution time — allow more faithful descriptions. In fact such an assumption destroys abstractness of time as specifications depend on specific implementation choices. It will be shown that the zero duration assumption for atomic actions is more general and leads to much simpler theories.

The overview is carried out by considering successively the choices for a designer of a timed process algebra, at model level and at language level.

The choice of the model determines the semantics and thus the intrinsic expressivity of a process algebra. As various types of semantics have been used for the algebras considered, we take operational semantics — strong bisimulation semantics — as a basis of the comparison. The reason is that most algebras have been given such semantics or some operational semantics can be deduced in most of cases. In our comparison we take into account features allowing abstraction (silent actions, hiding) only as long as they enhance expressivity.

The languages used for timed process algebras can be viewed as extensions of the languages used for untimed process algebras by adding some specific constructs or by assuming that in some cases prefixing by an action may delay. Some criteria for the comparison of the languages considered are the minimality of the set of the operators and their appropriateness for a natural and direct description.

The paper is organised as follows:

- Section 2 is devoted to the presentation of a general model for timed systems defined for an arbitrary time domain. The model is transition systems labelled with either action names from an arbitrary action alphabet or by elements of an appropriate time domain D. Some general properties of this model are discussed as well as their importance concerning the capability to characterise time constraints of various types.
- In section 3, we present a comparison of the expressive capabilities of the following timed process algebras (presented in alphabetic order):
  - ACP_ρ (Real Time ACP) of J.C.M. Baeten and J.A. Bergstra [BB90,Klu91].

- ATP (Algebra of Timed Processes) of the authors [NRSV90,NS90]. We sometimes make reference to a variant of ATP presented in [NSY91].
- TCSP (Timed CSP) of G.M. Reed and A.W. Roscoe [RR88,DS89,Sch91].
- TeCCS (Temporal CCS, or TCCS) of F. Moller and C. Tofts [MT90].
- TiCCS (Timed CCS, or TCCS) of Wang Yi [Wan90,Wan91].
- TPCCS (Timed Probabilistic CCS) of H. Hansson and B. Jonsson. We focus only on features relative to time.
- TPL (Temporal Process Language) of M. Hennessy and T. Regan [HR90, HR91].
- U-LOTOS (Urgent LOTOS) of T. Bolognesi and F. Lucidi [BL91].

We especially focus on constructs used to describe time constraints and their semantics. The presentation is organised as an exercise for building a timed process algebra from a standard process algebra for untimed systems.

2 The models

In this section, we present a general model for timed systems. We consider labelled transition systems whose states are process expressions, and whose labels are either elements a of a *vocabulary of actions* A or elements d of a *time domain* D. A may contain non-visible (internal) actions denoted by τ; visible actions are denoted by α.

$P \xrightarrow{a} Q$ means that the process P may perform the atomic and timeless action a and then it behaves as Q.

$P \xrightarrow{d} Q$ means that the process P may *idle* for d time units after which it behaves as Q.

Before discussing the properties of such models, we propose a general definition of time domains.

2.1 Time domain

Definition: A *time domain* is a commutative monoid $(D, +, 0)$ satisfying the following requirements.

- $d + d' = d \Leftrightarrow d' = 0$
- the preorder $\leq$ defined by $d \leq d' \Leftrightarrow \exists d'' : d + d'' = d'$ is a total order

The following properties can be easily proved.

- 0 is the least element of D

for any d, d', if $d \leq d'$, then the element d'' such that $d + d'' = d'$ is unique. It is denoted by $d' - d$

We denote $D - \{0\}$ by D_*. We also write $d < d'$ instead of $d \leq d' \ \wedge \ d \neq d'$.

D is called *dense* if $\forall d, d' \ : \ d < d' \Rightarrow \exists d'' \ : \ d < d'' < d'$

D is called *discrete* if $\forall d \, \exists d' \ : \ d < d' \ \wedge \ \forall d'' \ : \ d < d'' \Rightarrow d' \leq d''$. Since the order is total, d' is unique and is called the *successor* of d, denoted by $\mathtt{succ}(d)$. An important property of a discrete domain is that for any d, $\mathtt{succ}(d) = d + \mathtt{succ}(0)$. That is, any element of D can be obtained from 0 by adding as many $\mathtt{succ}(0)$ as necessary.

Examples of time domains are $\mathbb{N}$ (discrete), $\mathbb{Q}^+$ and $\mathbb{R}^+$ (dense), or even the singleton $\{0\}$.

In the transition relation defined in the beginning of this section, we do not allow 0 to be a label, that is, labels are elements of $A \cup D_*$.

2.2 The time domain in the algebras considered

TCSP and ACP$_\rho$ are explicitly defined over a dense time domain.

For TiCCS, TeCCS and U-LOTOS, the choice of a discrete or dense time domain is important neither for the syntax nor for the semantics. However, the axiomatisation strongly depends on this choice, especially for that of parallel composition. In TeCCS a complete set of axioms is provided in the discrete case. In [Wan91], Wang explains how an expansion theorem can be given in the dense case. This is possible only if we have a way of recording and use the instant when an action is performed.

TPCCS, TPL and ATP are defined over a discrete time domain. Extending them to a dense time domain requires some modification of the syntax. In [NSY91], a generalisation of ATP, parametrised by an arbitrary time domain, has been proposed.

2.3 Model properties

In this section we give an overview of the most important model properties and their importance for the characterisation of features of timed systems.

2.3.1 Time determinism

It is usually admitted that when a process P is idle (does not perform any action) for some duration d, then the resulting behaviour is completely determined from P and d. In other words, the progress of time should be deterministic. This property, satisfied by the models of all the algebras, we consider, can be expressed by

$$\forall P, P', P'', d \ : \ P \xrightarrow{d} P' \ \wedge \ P \xrightarrow{d} P'' \ \Rightarrow P' = P''$$

where $=$ is the syntactic equality.

2.3.2 Time additivity

In order to ensure the soundness of the notion of time, it is usually required that

- a process which can idle for $d + d'$ time units, can idle for d and then for d' time units, and vice-versa
- in both cases, the resulting behaviour is the same

We call this property *time additivity* (*time continuity* in [Wan90]). It is present in all the algebras and it is formally defined by

$$\forall P, P', d, d' \; : \; (\exists P'' \; : \; P \xrightarrow{d} P'' \wedge P'' \xrightarrow{d'} P') \; \Leftrightarrow \; P \xrightarrow{d + d'} P'$$

2.3.3 Deadlock-freeness

In untimed systems, a blocked or terminated process is represented by a deadlock in the model, since it cannot perform any action. For timed systems, it is natural to demand that, a terminated process does not block time, because of the strong synchrony hypothesis concerning time progress. If no distinction is made between termination and deadlock, this implies that there is no sink state in the model, which can be written as

$$\forall P \exists l \in A \cup D_* \exists P' \; : \; P \xrightarrow{l} P'$$

In algebras like TeCCS, U-LOTOS and ACP$_\rho$, there exist processes whose models do not satisfy this property, and thus they can block the progress of time Such *time-locks* may be used to detect some timing inconsistencies in specifications.

2.3.4 Action urgency

In all the considered algebras, there are processes which *must* perform an action without letting time pass, that is,

$$\exists P, a, P' \; : \; P \xrightarrow{a} P' \wedge \forall d \; : \; P \overset{d}{\not\rightarrow}$$

This defines a notion of urgency for actions, as a process may block the progress of time and enforce the execution of an action before some delay.

However, in TPCCS, TPL, TCSP and TiCCS, urgency is possible — and is enforced — for invisible actions only; this can be expressed by

$$\forall P, P', d, Q \,:\, P \xrightarrow{\tau} P' \;\Rightarrow\; P \overset{d}{\not\longrightarrow} Q$$

This property is called *minimal delay*, *maximal progress* or *tau-urgency*. In CCS-based algebras, it is strongly related to the communication mechanism. Indeed, a communication in CCS yields a tau action; thus, this property allows to ensure that two processes communicate as soon as they are ready to do so.

In models without the general action urgency, it is not possible, for instance, to characterise the situation where a process sends a message at most 3 time units after it has been requested to do so.

2.3.5 Persistency

In some algebras (TiCCS, U-LOTOS and TCSP), the progress of time cannot suppress the ability to perform an action. This property, called persistency, is expressed by

$$\forall P, Q, P', d, a \,:\, P \xrightarrow{a} P' \wedge P \xrightarrow{d} Q \;\Rightarrow\; \exists P'' \,:\, Q \xrightarrow{a} P''$$

This property is not satisfied by ATP, TPL, TPCCS, TeCCS and ACP$_\rho$. In the latter two, it is even possible, for instance, to specify a process which may perform an action a at time $\frac{1}{2}$ or an action b at time $\frac{5}{7}$. In TPCCS, TPL and ATP, such a behaviour does not exist. In the generic version of ATP presented in [NSY91], where the time domain may be dense, the models satisfy a weaker requirement than persistency, which we call *interval persistency*. This property asserts that if a process may let time progress, then any action it can perform remains possible during some time interval. This is expressed by

$$\forall P \,\exists d > 0 \,\forall d' \in]0, d[, \forall Q, P', a \,:\, P \xrightarrow{d'} Q \wedge P \xrightarrow{a} P' \;\Rightarrow\; \exists P'' \,:\, Q \xrightarrow{a} P''$$

Notice that this property is always true for a discrete time domain. Like ATP, TPL and TPCCS could be easily adapted to a dense domain, in which case their models would also have the interval persistency property.

2.3.6 Finite variability and bounded variability

A process has the *finite variability* (*non-Zenoness*, *well-timedness*) property if it can perform only finitely many actions in a finite time interval. The only algebra for which every process satisfy this requirement is TCSP. This is achieved by enforcing a *system-delay* between two actions of a sequential process. This assumption seems in fact to be the only solution to ensure finite variability, but it yields a complicated theory, and destroys abstractness of time.

To define formally this property, consider the family of relations $\overset{(a,d)}{\Longrightarrow}$ for $(a, d) \in A \times D_*$ on processes, defined by

$$P \overset{(a,d)}{\Longrightarrow} R \Leftrightarrow P \xrightarrow{d} Q \wedge Q \xrightarrow{a} R$$

A *time trace* of a process P is a maximal sequence $(a_0, d_0)\,(a_1, d_1) \ldots (a_i, d_i) \ldots$ such that

$$\exists P_1, P_2, \ldots, P_i, \ldots \;:\; P \overset{(a_0,d_0)}{\Longrightarrow} P_1 \overset{(a_1,d_1)}{\Longrightarrow} P_2 \ldots \overset{(a_i,d_i)}{\Longrightarrow} P_i \ldots$$

We represent by $T(P)$ the set of traces of P.

P satisfies the finite variability property if and only if

$$\forall d\, \forall \sigma = (a_0, d_0)\,(a_1, d_1) \ldots \in T(P) \;:$$
$$\left(i < j \leq \text{length}(\sigma) \;\wedge\; \sum_{k=i+1}^{j} d_k \leq d \right) \Rightarrow j - i < \infty$$

A stronger requirement should be satisfied by models in order that they represent implementable behaviours — we consider a behaviour to be *implementable* if it can be executed on a processor where the measure of time is provided by a discrete clock. We call this requirement *bounded variability*; it demands that for any duration d, there is an upper bound n of the number of actions performed within any time interval of length d.

This can be stated formally, for a given process P, by

$$\forall d\, \exists n\, \forall \sigma = (a_0, d_0)\,(a_1, d_1) \ldots \forall i, j \;:$$
$$\left(i < j \leq \text{length}(\sigma) \;\wedge\; \sum_{k=i+1}^{j} d_k \leq d \right) \Rightarrow j - i \leq n$$

This property guarantees implementability in the sense that one can establish a correspondence between model time d and a clock period for safe implementations. From this definition, for model time d one can take a clock period greater than or equal to $n.d_0$ where d_0 is an upper bound of atomic action durations. Bounded variability is satisfied by none of the considered algebras.

2.3.7 Bounded control

If we consider again realisability issues, for the same reasons as above, the set of initial actions of a process should not change too fast in a given time interval.

Given a process P, represent by *init*(P) the set of the actions it can perform, i.e.,

$$init(P) = \{a \mid \exists P' \;:\; P \xrightarrow{a} P'\}$$

A model has the bounded control property if there exists d such that for all states P and P' in the model, if $P \xrightarrow{d_1} P'$ and $init(P) \neq init(P')$, then $d_1 \geq d$. In fact, the modification of the initial actions corresponds to a change of the "control state" of the process. Bounded control expresses the fact that for any d, there is a bounded number of such changes in any time interval of duration d, and it means that it is possible to find a clock period which allows to handle these changes.

The bounded control property is satisfied for models defined over a discrete time domain, and for models of TCSP.

2.4 Discussion

An important question is which properties are essential and how their presence or absence influences description capabilities.

Most of the existing work adopt time determinism and additivity. Deadlock-freeness is not in our opinion an essential model property. Time-lock is of course an abnormal situation but in some theories it can correspond to non-realisable specifications.

A property like persistency seems to be a strong requirement which is very often adopted without justification. Saying that time progress does not change system's capabilities to perform actions seems to be counterintuitive as time is often used precisely as a parameter to control action executability (as in timeouts). This property, often combined with urgency with respect to τ only, allows to express the fact that some action *may* be executed during some time interval, but cannot guarantee *obligation* of execution. This corresponds, we believe, to a major distinction concerning description capabilities of formalisms.

Implementability properties prevent from having an unbounded number of state changes within finite time, which is an essential requirement for discrete machines. Both bounded variability and bounded control properties allow to establish a relationship connecting abstract (model) time with a processor's discrete clock period.

In the sequel, we consider models that are time deterministic and additive. As D is usually infinite, the models are generally infinitely branching transition systems. Figure 2 represents the model of the process "P timeout(3) Q" which behaves as P before time 3 or as Q at time 3, for $D = \mathbb{N}$, $D = \{0, 0.5, 1, 1.5, ...\}$ and D dense.

3 The Languages — How to Cook your own Timed Process Algebra

In this section we present an overview of the language features of the process algebras considered. We especially focus on constructs used to describe time constraints, their semantics, and the extension of the semantics of standard operators for timed transitions.

The presentation is organised as an exercise of building a timed process algebra TPA from a standard process algebra for untimed systems. Given such an untimed process algebra UPA we review the different ways of extending it encountered in the literature.

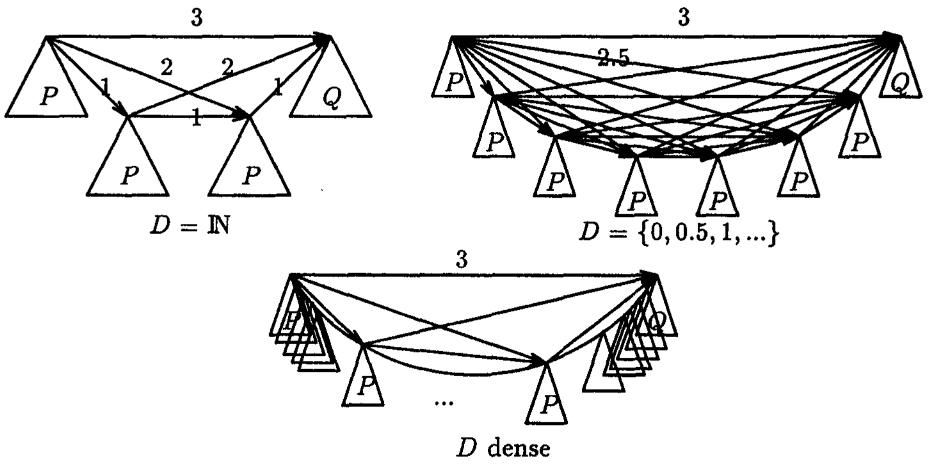

Figure 2

To make the comparison as concrete as possible, we define the meaning of constructs in terms of the common reference model presented in the previous section. The interpretation of the original semantics into this common framework required some simplifications that, we hope, do not bias the comparisons.

3.1 General principles

3.1.1 Process algebra

A process algebra PA is defined as a quadruple $(OP, L, R_L^{OP}, \sim)$ where

- OP is a set of operators defining the language of PA
- L is a set of transition labels
- R_L^{OP} is a set of structural operational semantics rules à la Plotkin [Plo81] associating with a term of PA a transition system labelled on L (a model)
- $\sim \subseteq$ PA $\times$ PA is a behavioural equivalence defined over the models. It it usually required that this equivalence be a congruence, since a compositional semantics is an crucial issue for a language

3.2 Timed process algebra

We consider here an *untimed process algebra* UPA $= (OP, A, R_A^{OP}, \sim)$, where A is a vocabulary of actions and $\sim$ is the strong equivalence relation [Mil80,Mil83].

The *timed process algebra* TPA $= (OP \cup OP', L, R_L^{OP \cup OP'}, \overset{T}{\sim})$ is defined from UPA by adding a set OP' of *time-constraining* operators.

Time-constraining operators are operators which can transform untimed processes into *time-constrained* ones. Examples of such operators are: time-lock constructs, operators which delay the execution of a process, operators which impose some urgency on the execution of a process, timeout operators, watchdog operators.

Processes of TPA are timed systems. Following the ideas in previous section, we decide to represent their models by transition systems labelled on $L = A \cup D_*$. Moreover, we require that the models of processes of TPA be time-deterministic and time-additive.

We choose for the equivalence relation $\overset{T}{\sim}$ the strong equivalence with respect to L-transitions.

In UPA, time progress does not have to be represented in the models, since by definition, it has no influence on the behaviour of an untimed process. In order to embed UPA in TPA, we have to choose the timed model corresponding to an untimed process (its time-equivalent). This must take into account the following requirements.

Semantics conservation: the untimed process and its time-equivalent should have the same behaviour as long as we observe execution of actions only. This imposes that the rules R_A^{OP} of UPA remain valid in TPA, *as far as they are applied on terms of UPA*.

Isomorphism: we also require that for any terms P, Q of UPA $P \sim Q$ if and only if $P \overset{T}{\sim} Q$. This requirement guarantees that the theory of processes of UPA is isomorphic to that of the restriction of TPA to operators of UPA. Thus, any theoretical development in UPA about an untimed process remains valid in TPA, and conversely.

In the sequel we consider a standard process algebra UPA and apply these principles.

3.3 Syntax and semantics of UPA

The language of terms of UPA is described by the following syntax, where Nil is is a constant, a and α are elements of A, $\alpha \neq \tau$, and X is an element of a set of process variables $\mathcal{X}$.

$$P ::= Nil \ \mid \ X \ \mid \ aP \ \mid \ P + Q \ \mid \ P \,\|\, Q \ \mid \ P \backslash \alpha \ \mid \ \mathbf{rec} X \cdot P$$

All the operators except $\|$ are taken from CCS [Mil80]. We choose prefixing instead of sequential composition to simplify the presentation, since it usually yields a simpler theory. The restriction operator $P \backslash \alpha$ prevents P from performing the visible action α.

We do not impose any choice of parallel composition operator, which can be CCS-like, ACP-like (in which case we must define a communication function for actions), or CSP or LOTOS-like (it should then be parametrised by sets of actions).

For such a language, we suppose that a standard operational semantics is defined, in terms of transition systems labelled by element of A (actions), by the following axiom and rules

$$aP \xrightarrow{a} P \qquad \frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P+Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{b} P',\ b \neq \alpha}{P\backslash\alpha \xrightarrow{b} P'\backslash\alpha} \qquad \frac{P[(\mathtt{rec}X \cdot P)/X] \xrightarrow{a} P'}{\mathtt{rec}X \cdot P \xrightarrow{a} P'}$$

We do not provide the semantic rules for the parallel composition operator; we only demand that it has an interleaving semantics, as we take the models to be transition systems labelled on A.

3.4 Embedding UPA in a TPA

We first have to define how to add time to an untimed process, taking into account the requirements presented in 3.2.

Isomorphism. A simple answer to the isomorphism requirement is obtained by considering that time progress is possible without effect at any point of the execution of an untimed process. We obtain then the timed model of a term P of UPA by adding loops of the form $P \xrightarrow{d} P$ for any d of D_*. Another possibility is to allow P to idle if and only if it cannot perform any action u of some subset U of *urgent actions* of A. For instance, U may be the subset of internal actions of A. This induces urgency in the models for states with a u among their initial actions. We can express this solution by the rule

$$\frac{\forall u \in U,\ P \overset{u}{\not\longrightarrow}}{P \xrightarrow{d} P}$$

The latter principle is implicitly adopted in TiCCS, TPL and TPCCS with $U = \{\tau\}$, whereas the former one is adopted in U-LOTOS, ACP$_\rho$, TeCCS and ATP. We can consider the first solution as a particular case of the second one, with $U = \emptyset$.

In TCSP, the solution adopted (adding a system delay after execution of an action in a sequential process and before each recursive call) does not satisfy the requirement. For instance, the process $\mathtt{rec}X \cdot aX$ is not equivalent to the process $a\,\mathtt{rec}X \cdot aX$ in TCSP while they are equivalent in untimed CSP. Hence, the laws of CSP are no longer valid in TCSP.

Notice that the correspondence between operators of UPA and their equivalent in TPA is not immediate in ACP$_\rho$, TeCCS and ATP.

- In basic ACP$_\rho$, some of the operators of ACP are not present, but may be derived (for instance the atomic action constants).
- In TeCCS and ATP, the timed process $a\,P$ does not correspond to the untimed process $a\,P$. In fact, the latter corresponds, in TeCCS to $\delta\, a\, P$, and in ATP to $\lfloor a\,P \rfloor^\omega$. Similarly, in TeCCS, Nil does not correspond to 0, but to $\delta 0$. The confusion for prefixing may be removed if we denote it differently in the timed algebra.

In the sequel, we consider the general solution to the isomorphism requirement, with a set of urgent actions U.

Semantics conservation. The other requirement demands that the operational semantics rules for time-equivalent of untimed processes of UPA remain valid in TPA. For instance, in TPA, there should be a rule

$$\frac{P, P', Q \in \text{ UPA }, P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$$

Since it is syntactically possible to determine whether a term of TPA is built using operators of UPA only, we can define a predicate InUPA(P), whose value is true if P is in UPA. The rule may then be rewritten in

$$\frac{\text{InUPA}(P), \text{InUPA}(P'), \text{InUPA}(Q),\ P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$$

However, if we want to obtain a compositional semantics of TPA, the premise $P, P', Q \in$ UPA is not enough; it should be replaced by

$$\exists P_1, P_1', Q_1' \ :\ \text{InUPA}(P_1),\ \text{InUPA}(P_1'),\ \text{InUPA}(Q_1),\ P \overset{T}{\sim} P_1,\ P' \overset{T}{\sim} P_1',\ Q \overset{T}{\sim} Q_1$$

This kind of premise is clearly not acceptable in a structural operational semantics rule, since it is not based on a syntactic predicate, but on an semantic one: indeed, it means that we have to decide operationally whether some process of TPA has a model equivalent to a process of UPA.

We have thus to admit that this rule, and all the other rules of R_A^{OP} in UPA, are also valid in TPA.

3.4.1 Timed transitions of UPA operators

Concerning the definition of timed transitions, the semantics of OP in TPA must be defined so that

$$\frac{\forall u \in U,\ P \stackrel{u}{\not\longrightarrow}}{P \xrightarrow{d} P}$$

for any P where only elements of OP occur.

Of course, such a rule should not be applicable to any TPA process. It can be checked that the semantic rules for TPA below satisfy the following properties.

- Their restriction to time-equivalents of UPA gives the same derivations as the above rule,
- They preserve time determinism and additivity,

Moreover, they are perfectly acceptable for TPA processes.

$$Nil \xrightarrow{d} Nil \qquad \frac{a \notin U}{aP \xrightarrow{d} aP} \qquad \frac{P \xrightarrow{d} P'\ ,\ Q \xrightarrow{d} Q'}{P + Q \xrightarrow{d} P' + Q'}$$

$$\frac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q',\ \forall u \in U : \left(P \| Q \stackrel{u}{\not\longrightarrow}\ \wedge\ \forall d' < d \forall P' : P \| Q \xrightarrow{d'} P' \Rightarrow P' \stackrel{u}{\not\longrightarrow}\right)}{P \parallel Q \xrightarrow{d} P' \parallel Q'}$$

$$\frac{P \xrightarrow{d} P'}{P \backslash \alpha \xrightarrow{d} P' \backslash \alpha} \qquad \frac{P[(\mathtt{rec}X \cdot P)/X] \xrightarrow{d} P'}{\mathtt{rec}X \cdot P \xrightarrow{d} P'}$$

The rule for parallel composition means that the composition of P and Q can idle for d if both can do so, and moreover, if the composition cannot perform an urgent action before d.

The rules for $+$, $\|$ and restriction apply when the operands of the operators have d-transitions. However, in TPS, there may be processes without such transitions. We have to decide what is the effect of these operators in this case.

- For parallel composition, it is not possible to add a rule without violating the strong synchrony hypothesis.
- In $P\backslash\alpha$, if α is in U and is the only urgent action that P may perform, then the rules for restriction yield a state which is, either a sink state, or a state where non-urgent actions are made urgent. If we do not want such behaviours, we have to add the following rule.

$$\frac{\forall u \in U : P \xrightarrow{u} \;\Rightarrow\; u = \alpha}{P\backslash\alpha \xrightarrow{d} P\backslash\alpha}$$

- For the alternative choice, we may decide that if P may idle, but not Q, then $P+Q$ may idle, by adding the rule

$$\frac{P \xrightarrow{d} P' \,,\, Q \overset{d}{\not\rightarrow}}{P+Q \xrightarrow{d} P'}$$

and its symmetric.

This semantics for $+$ is adopted in ACP$_\rho$. In TeCCS, two alternative choice operators are defined: the *weak choice* operator, for which the latter rules are added, and the *strong choice* one, where they are not. The other algebras propose the strong choice operator (we only consider the external choice of TCSP, which is strong).

3.5 Time-constraining operators

We present time-constraining operators used in the process algebras considered. The aim of the presentation is a classification and comparison without caring about minimality. We also emphasise the effect of the operators on the properties of the models.

3.5.1 Time-lock

It is a constant 0 representing the process performing no transition. It is a basic operator of TeCCS, and it can be derived in U-LOTOS. In ACP$_\rho$, the process $\delta(d)$ is a time-lock at time d.

Naturally, a time-lock process may be present only if the deadlock-freeness property is not required.

3.5.2 Delay operators

Operators for delaying processes are the most common ones in process algebras. Their effect consists in postponing the execution of a process by a parameter d. We can classify them into three groups.

Finite idling is defined with various syntaxes as a basic operator in TiCCS, TeCCS, TPL, U-LOTOS and TCSP. We denote it by a prefixing operator (d). where $d > 0$. $(d)P$ behaves as P after exactly d time units. Its semantics is given by the following rules.

$$\frac{d' < d}{(d)P \xrightarrow{d'} (d-d')P} \qquad (d)P \xrightarrow{d} P$$

Finite idling may be modelised in ATP, TPCCS and ACP$_\rho$, though its definition is not trivial in the latter, especially in the absolute time version.

Time-stamped actions. In ACP$_\rho$, the basic construct of the algebra is obtained by imposing a *time-stamp* to actions. Two versions are presented, with absolute or relative time-stamps. In the absolute case $a(d)$ performs a at time d since the beginning of the process, whereas in the relative case, $a[d]$ performs a d time units after the previous action has been performed. In the latter case, and if we consider it as a prefixing operator, its semantics may be described by

$$\frac{d' \leq d}{a[d] \xrightarrow{d'} a[d-d']P} \qquad a[0]P \xrightarrow{a} P$$

Time-stamped actions induce the general urgency property in the models. It can be modelled only in algebras where this property is allowed. In ACP$_\rho$, due to the choice for the semantics of alternative choice, it suppresses also any kind of persistency in the transition systems.

Unbounded idling is defined in algebras where there may be urgent actions, like TeCCS and ATP. Its purpose is to suppress this urgency. However, it has a different semantics in these two algebras:

- in TeCCS, the unary operator δ has the following semantics:

$$\frac{P \xrightarrow{a} P'}{\delta\, P \xrightarrow{a} P'} \qquad \delta\, P \xrightarrow{d} \delta\, P$$

- in ATP, the operator $\lfloor\,\rfloor^\omega$ has the following semantics:

$$\frac{P \xrightarrow{a} P'}{\lfloor P \rfloor^\omega \xrightarrow{a} \lfloor P' \rfloor^\omega} \qquad \frac{P \xrightarrow{d} P'}{\lfloor P \rfloor^\omega \xrightarrow{d} \lfloor P' \rfloor^\omega} \qquad \frac{\forall d',\, P \not\xrightarrow{d'}}{\lfloor P \rfloor^\omega \xrightarrow{d} \lfloor P \rfloor^\omega}$$

The difference is that in TeCCS, if P cannot execute immediately an action, then the process $\delta\, P$ may only idle forever, whereas in ATP, the process P is allowed to let time progress.

Such operators are trivially definable in the other algebras, with the restriction that in those where τ is urgent, it is not possible to delay it.

Integral. In ACP$_\rho$, the integral operator $\int_{v \in V} P(v)$ for a process P parametrised by a time variable v behaves as P where v may be replaced by any value of the subset V of D. A simple use of this operator is $\int_{v \in I} a[v]$, where I is an interval. This process may perform an a at any time in I. Thus, it delays the execution of a by some value v in I. It can be described in TeCCS and ATP, provided the interval is right-closed.

Notice that in TiCCS, prefixing $a@v\, P$ is equivalent to an integral, where the interval I is the whole domain. The value v may be used in delay values in P. Such a construct is useful to provide an expansion theorem for parallel composition when the time domain is dense.

3.5.3 Urgency operators

Immediate actions. In TeCCS and ATP, prefixing by an action (aP) impose that this action be performed immediately. To avoid confusion with the prefixing operator of UPA, we denote this urgent prefixing by $\dot{a}P$. Its semantics is given by the unique axiom $\dot{a}P \xrightarrow{a} P$. It is also expressible in the algebras where urgency of actions are allowed, that is in U-LOTOS and ACP$_\rho$, if in the latter we allow the time-stamp 0. Conversely, it has no equivalent in the other algebras considered.

Time-stamped actions (ACP$_\rho$). They have been presented above. They have the double effect of delaying and imposing urgency once the delay has expired. We could call them a *punctuality* feature.

As soon as possible. In U-LOTOS, the primitive operator **asap** enforces the urgency of a set of actions in the whole execution of a process. For sake of simplicity, we only present its semantics in the case where this set of actions is reduced to a singleton.

$$\frac{P \xrightarrow{a'} P'}{\mathbf{asap}_a \,\mathrm{in}\, P \xrightarrow{a'} \mathbf{asap}_a \,\mathrm{in}\, P'} \qquad \frac{P \overset{a}{\not\rightarrow},\ P \xrightarrow{d} P',\ \forall d' < d \forall Q,\ P \xrightarrow{d'} Q \Rightarrow Q \overset{a}{\not\rightarrow}}{\mathbf{asap}_a \,\mathrm{in}\, P \xrightarrow{d} \mathbf{asap}_a \,\mathrm{in}\, P'}$$

The second rule means that the $\mathbf{asap}_a \,\mathrm{in}\, P$ can idle for d if P can do so, and cannot perform an a before d.

This very powerful operator is expressible in ACP$_\rho$, ATP and TeCCS, and clearly not in the other algebras.

3.5.4 Timeout operators

A timeout is an operator with two arguments P and Q and a parameter $d \in D_*$. We call P the body and Q the exception of the timeout.

A timeout for P, Q and d behaves as P if an initial action of P is performed within time d, otherwise it behaves as Q, after time d.

Depending on the interpretation of "initial action" and "within time d", several variants of timeout operators have been proposed.

a) $P \overset{d}{\rhd} Q$ in ATP$_D$ ([NSY91]) with the following semantics.

$$\frac{P \xrightarrow{a} P'}{P \overset{d}{\rhd} Q \xrightarrow{a} P'} \qquad \frac{P \xrightarrow{d'} P' \,,\, d' < d}{P \overset{d}{\rhd} Q \xrightarrow{d'} P' \overset{d-d'}{\rhd} Q}$$

$$\frac{P \xrightarrow{d} P'}{P \overset{d}{\rhd} Q \xrightarrow{d} Q} \qquad \frac{P \xrightarrow{d} P' \,,\, Q \xrightarrow{d'} Q'}{P \overset{d}{\rhd} Q \xrightarrow{d+d'} Q'}$$

The last rule is necessary to preserve time additivity.

With this operator an action that P may perform after some time is also interpreted as an "initial action" of P. "Within time d" is interpreted as "before time d"; we call timeouts with such a interpretation *strong timeouts.*

b) $\lfloor P \rfloor^d(Q)$ in ATP (start-delay operator) with the following semantics.

$$\frac{P \xrightarrow{a} P'}{\lfloor P \rfloor^d(Q) \xrightarrow{a} P'} \qquad \frac{P \xrightarrow{d'} P' \,,\, d' < d}{\lfloor P \rfloor^d(Q) \xrightarrow{d'} \lfloor P' \rfloor^{d-d'}(Q)} \qquad \frac{\forall d'\, P \overset{d'}{\not\longrightarrow} \,,\, d'' < d}{\lfloor P \rfloor^d(Q) \xrightarrow{d''} \lfloor P \rfloor^{d-d''}(Q)}$$

$$\lfloor P \rfloor^d(Q) \xrightarrow{d} Q \qquad \frac{Q \xrightarrow{d'} Q'}{\lfloor P \rfloor^d(Q) \xrightarrow{d+d'} Q'}$$

This operator differs from the previous one in that it also allows to postpone the urgent actions P may perform. It is a strong timeout too.

c) $P \rhd_d Q$ in TPCCS is a strong timeout with a strict interpretation of initial actions:

$$\frac{P \xrightarrow{a} P'}{P \rhd_d Q \xrightarrow{a} P'} \qquad \frac{d' < d}{P \rhd_d Q \xrightarrow{d'} P \rhd_{d-d'} Q}$$

$$P \rhd_d Q \xrightarrow{d} Q \qquad \frac{Q \xrightarrow{d'} Q'}{P \rhd_d Q \xrightarrow{d+d'} Q'}$$

d) The timeout of TCSP is a weak one, in the sense that at time d both the body and the exception can be executed; that is, P may start in the interval [0,d], and Q may be chosen at d. The weak timeout preserves persistency in the models. The interpretation of initial actions is the same as in case a. In the semantics, an urgent internal action τ is used to enforce a choice between P and Q at time d. The weak timeout can be expressed in terms of the strong one, but the converse is not true.

3.5.5 Watchdog operators

A watchdog is an operator with two arguments P (body) and Q (exception) and a parameter d in D_*.

It behaves as P until time d. At time d, P is "aborted" and Q is started.

Such operators are proposed in ATP (execution delay) and TCSP (time interrupt). As for the timeouts, the watchdog is strong in ATP, and weak in TCSP (in the latter, P may still perform some action at time d, which is not the case in ATP).

In TCSP, if P terminates successfully, the watchdog is cancelled. In ATP, there is no operational notion of termination. However, the watchdog may be cancelled if P performs a special action ξ, called *cancel*. We present hereafter the semantics of the watchdog of ATP.

$$\frac{P \xrightarrow{a} P' \,,\, a \neq \xi}{\lceil P \rceil^d(Q) \xrightarrow{a} \lceil P' \rceil^d(Q)} \qquad \frac{P \xrightarrow{\xi} P'}{\lceil P \rceil^d(Q) \xrightarrow{\tau} P'}$$

$$\frac{P \xrightarrow{d'} P' \,,\, d' < d}{\lceil P \rceil^d(Q) \xrightarrow{d'} \lceil P' \rceil^{d-d'}(Q)} \qquad \frac{P \xrightarrow{d} P'}{\lceil P \rceil^d(Q) \xrightarrow{d} Q} \qquad \frac{P \xrightarrow{d} P' \,,\, Q \xrightarrow{d'} Q'}{\lceil P \rceil^d(Q) \xrightarrow{d+d'} Q'}$$

4 Discussion

The paper is an overview and synthesis of existing results about timed process algebras. It hopefully contributes to the clarification of the following three different problems, designers of timed specification languages should in principle address.

1. What are the underlying principles of functioning of timed systems? In the introduction, we formulate some assumptions about the two-phase mode of functioning and provide pragmatic justifications. This functioning corresponds to some abstraction of the reality which has the advantage of clearly separating the actions from the time progress issue. It is argued that adopting such an "orthogonality" principle between actions and timed transitions is more paying than other approaches

imposing some non-zero durations to actions. In the latter, time is not abstract, i.e., independent of implementation choices.

2. What is a general model for timed systems, and what are its most relevant properties? Following assumptions about the functioning of timed systems, we take as models transition systems whose labels are either elements of an action vocabulary or elements of an appropriately chosen time domain.

   Concerning the properties studied, they can be classified as follows.

   - time determinism and additivity characterise fundamental properties of time.
   - properties characterising the expressivity of the model, like presence of time-locks and the different types of persistency or urgency.
   - realisability properties.

   The choice of a particular class of models should be determined for a given time domain as a compromise between realisability and expressivity.

3. How an untimed specification language can be consistently extended so as to obtain a timed specification language? We suggest a principle which has been more or less followed in several cases of consistent extensions (except for TCSP). Concerning the description capabilities of the language, it is difficult to make a precise comparison due to the differences of the semantic framework adopted. However, an important distinction appears concerning the expression of urgency.

This is a first partial synthesis of results in the area, which hopefully contributes to structuring them and suggests an approach for tackling the problem of introducing time in process algebras.

References

[AB84] D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Science*, 30, 1984.

[BB90] J.C.M. Baeten and J.A. Bergstra. *Real Time Process Algebra.* Technical Report CS-R9053, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1990.

[BC85] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In *LNCS 197: Proceedings CMU Seminar on Concurrency*, Springer-Verlag, 1985.

[BL91] T. Bolognesi and F. Lucidi. LOTOS-like process algebra with urgent or timed interactions. In *Proceedings of REX Workshop "Real-Time: Theory in Practice".* Mook, the Netherlands, June 1991.

[CHPP87] P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. LUSTRE: a declarative language for programming synchronous systems. In *14th Symposium on Principles of Programming Languages*, Munich, January 1987.

[DS89] J. Davies and S. Schneider. *An Introduction to Timed CSP.* Technical Report PRG-75, Oxford University Computing Laboratory, UK, August 1989.

[Har87] D. Harel. StateCharts : a visual approach to complex systems. *Science of Computer Programming*, 8–3:231–275, 1987.

[HR90] M. Hennessy and T. Regan. *A Temporal Process Algebra.* Technical Report 2/90, University of Sussex, UK, April 1990.

[HR91] M. Hennessy and T. Regan. *A Process Algebra for Timed Systems.* Technical Report 5/91, University of Sussex, UK, April 1991.

[Klu91] A.S. Klusener. *Completeness in Real Time Process Algebra.* Technical Report CS-R9106, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, January 1991.

[Mil80] R. Milner. A Calculus of Communicating Systems. In *LNCS 92*, Springer Verlag, 1980.

[Mil83] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25, 1983.

[Mil91] G. J. Milne. The Formal Description and Verification of Hardware Timing. *IEEE Transactions on Computers*, 40 (7), July 1991.

[MT90] F. Moller and C. Tofts. A Temporal Calculus of Communicating Processes. In J.C.M. Baeten and J.W. Klop, editors, *LNCS 458. Proceedings of CONCUR '90 (Theories of Concurrency: Unification and Extension), Amsterdam, the Netherlands*, pages 401–415, Springer-Verlag, August 1990.

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an Algebra for Timed Processes. In *Proceedings of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, Sea of Gallilee, Israel, April 1990.

[NS90] X. Nicollin and J. Sifakis. *The algebra of timed processes ATP: theory and application.* Technical Report RT-C26, LGI-IMAG, Grenoble, France, December 1990.

[NSY91] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to Timed Graphs and Hybrid Systems. In *Proceedings of REX Workshop "Real-Time: Theory in Practice".* Mook, the Netherlands, June 1991.

[Plo81] G.D. Plotkin. *A Structural Approach to Operational Semantics.* Technical Report DAIMI FN-19, Århus University. Computer Science Department, Århus, Denmark, 1981.

[RR88] G.M. Reed and A.W. Roscoe. A timed model for Communicating Sequential Processes. *Theoretical Computer Science*, 58 (pp 249–261), 1988.

[Sch91] S. Schneider. *An Operational Semantics for Timed CSP.* Programming Research Group, Oxford University, UK, February 1991.

[Wan90] Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *LNCS 458. Proceedings of CONCUR '90 (Theories of Concurrency: Unification and Extension), Amsterdam, the Netherlands*, pages 502–520, Springer-Verlag, August 1990.

[Wan91] Wang Yi. CCS + Time = an Interleaving Model for Real Time Systems. In *Proceedings of ICALP '91, Madrid, Spain*, July 1991.