Université de Liège

Faculté des Sciences Appliquées

Systèmes et
Automatique

Institut d'Electricité Montefiore, B28
Université de Liège au Sart Tilman
B-4000 Liège 1 (Belgique)

An Enhanced Version of Timed LOTOS and its Application to a Case Study

Luc Léonard
Guy Leduc

Research Assistant of the F.N.R.S.
(National Fund for Scientific Research, Belgium)
Research Associate of the F.N.R.S.

Published in:
Proceedings of FORTE'93, Boston, 23-29 october 1993
R. Tenney, P. Amer, M.U. Uyar (Ed.),
Elsevier Science Publishers BV (North Holland), 1994
pages 483-498

Date : October 1993
Status : Public
RS 94-01

An Enhanced Version of Timed LOTOS and its Application to a Case Study

Luc Léonard[a] and Guy Leduc[b]

We propose here ET-LOTOS, a timed extension of LOTOS. It is an enhancement of Timed LOTOS that we presented at FORTE 92. We show how some simple modifications allow us to improve the expressiveness of our former formalism. To assess ET-LOTOS, we apply it to the specification of a small case study. Finally, we show how the semantics of ET-LOTOS could be modified to easily define more powerful operators, if one ever had to.

Keyword Codes: F.4.3; D.3.1; D.3.3
Keywords: Mathematical Logic and Formal Languages, Formal Languages; Programming Languages, Formal Definitions and Theory; Language Constructs and Features

1. INTRODUCTION

The formalism presented here, ET-LOTOS (Enhanced Timed LOTOS), is the result of further thought about the introduction of time in the formal description technique LOTOS [BoB87], [ISO8807]. In FORTE'92 we presented Timed LOTOS [LeL93a]. Most of its basic design features are kept unchanged in ET-LOTOS but further study, and especially the application of Timed LOTOS to the specification of a case study, helped us to get a clearer idea of what our timed formalism had to look like, and of the improvements we should try to focus on, or the options that were less important. We have applied this knowledge to the design of ET-LOTOS.

This paper consists of three main parts. First, we present ET-LOTOS. In section 2, we explain the Timed LOTOS features we have tried to enhance, and why. In section 3, we show how ET-LOTOS responds to these goals and we define its semantics. In the second part, in section 4, we apply ET-LOTOS to the specification of a case study to illustrate and assess its capabilities. The third part is more theoretical. In section 5, we show how slight changes in the semantics allow an easy definition of more sophisticated "high level" operators. Finally, we conclude with a comparison between ET-LOTOS and some other timed extensions of LOTOS.

2. HOW TO IMPROVE TIMED LOTOS

2.1. A Brief Introduction to Timed LOTOS [LeL93a]

As an introduction to timed FDTs, an overview of the various works on this subject can be found in [NiS91], with a classification of them. A survey is also presented in our previous work, [LeL93a]. We will not repeat it again, but will just recall the main features of Timed LOTOS.

No action urgency. In timed process algebra, an opportunity must be given to express urgency in the model, by requiring that a given action occur immediately or within some precise time limits, or after a certain delay. Whereas enforcing a process to idle a certain amount of time is quite simple and intuitive, enforcing that an action occur before a certain time limit is more complex. The usual way to express that an event must occur at, or before, a given instant is by "blocking the passing of time". In other words, the possibility for the process to continue its progression beyond the given instant is conditioned by the occurrence of the event. This mechanism may create some counter-intuitive results when combined with the process algebraic paradigm, that states that the occurrence of an event (except for *i* the internal event) requires the simultaneous participation of the environment. If, at the limit instant, the environment is not willing to interact, this results in a temporal deadlock: no action can be performed and the process cannot idle beyond this instant. In such models the passing of time may thus be conditioned by the environment.

---

[a] Research Assistant of the F.N.R.S. (National Fund for Scientific Research, Belgium)

[b] Research Associate of the F.N.R.S.

In Timed LOTOS, we chose not to give the opportunity to enforce, by blocking the passing of time, the occurrence of an external event at a specific instant or before a certain time limit. Our experience in trying to describe various temporal mechanisms led us to the conclusion that such a capability is useless. Furthermore, such mechanisms based on a blocking of time are also less easy to use in practice. We discussed about these points in [LeL93a] and [LeL93b].

In Timed LOTOS, external actions were always persistent: prefixing a process by an external action means that this action will be proposed until it is accepted, without any restriction on the time it will require. The only way to stop this offer is by the occurrence of a conflicting event.

Maximal progress property. We have an opposite opinion as regards the urgency of the internal events[1], whose occurrences do not depend on the environment. An internal event must happen as soon as it becomes possible, or in other words: if an event is ready to occur, why should it wait? This rule, commonly designed as the maximal progress, might seem a little bit radical. But it turns out to be very powerful in practice, especially to ensure the synchronisation between concurrent processes.

No restriction on the time domain. Neither the syntax nor the semantics are restricted to a specific time domain, i.e. a dense time domain is supported as well as a discrete one.

A discrete time domain requires the careful definition of the smallest grain of time of the specified system, to be able to express any timed constraint w.r.t. this basic unit of reference. Even if, in a practical design, such a grain of time likely exists, this is not always the case in an abstract design. Consider for example a process that sends messages at an continuously increasing throughput. This will lead to continuously decreasing delays between successive transmissions. Moreover, expressing all time delays or time instants w.r.t. a reference time unit will likely turn out to be inconvenient. This phenomenon becomes really critical in a practical design where this grain of time is tightly bound to the abstraction level of the specification. Therefore, when a process is refined, this may require the selection of a smaller grain of time, and the whole specification needs to be rewritten for consistency. Such problems of grain of time can be avoided by using a dense time domain: in this case it is always possible to find a time value between any two other values.

Three new timed operators were introduced. $\Delta^{[d1,d2]}\ P$ allowed the introduction of a delay before the execution of process P. It was particular in that it brought a notion of non determinism: the delay had no fixed value, but instead a random value from the interval $[d_1, d_2]$. The other two operators, $\Delta^I_g(P)$ and $\lfloor P \rfloor^d_g(Q)$ were what we will call in the sequel "high level" operators. They tackled the problem of expressing constraints on the occurrence of interactions, what appeared to be much harder than for 'simple' actions introduced by prefixing. We will not explain more here about these two operators, as we will come back to this subject in chapter 5.

In order to define these new operators, a special event was introduced, θ, that only appeared in the semantic model. It can be seen as an invisible event, like i, but whose occurrence does not solve a choice. θ helped giving Timed LOTOS the persistency and reverse persistency properties[2], which were useful to define $\Delta^I_g(P)$ and $\lfloor P \rfloor^d_g(Q)$.

2.2. Nothing is Perfect

Further study, and especially the application of Timed LOTOS to the specification of a case study, enlightened some limitations in using Timed LOTOS which thus required some improvements. We picked up:

- The difficulty to offer actions for only a finite period of time. This had to be done by way of a time-out like mechanism. For instance: $g;\ A\ []\ \Delta^{[5,5]}\ i;\ stop$ to describe an event g only offered until time 5. Such a construction is even heavier in case of a choice between several such actions with different times.

---

[1] An internal event, denoted i in LOTOS, can be introduced either directly by prefixing, or by way of the hiding operator, that "hides" an external event by turning it into i.

[2] The definitions of these properties are given in point 5

- A particular limit case of the above mentioned problem is the *punctual offer*, in which an action is only proposed at a given, punctual instant. Such offers do not correspond to a real situation, but are useful abstractions. It is not possible in Timed LOTOS to describe these actions in a satisfying way. For instance, in *g; A [] i; stop* , *i* is introduced to ensure that the offer of *g* will not last beyond the initial instant. But at time zero, *i* competes with *g* and could prevent an interaction with the environment, which is not the desired effect. To avoid this problem, one has to use an approximation: $g;\ A\ []\ \Delta^{[\varepsilon,\varepsilon]}\ i\ ;\ stop$, with ε taken very small.
- In Timed LOTOS, we did not provide explicitly a means to handle time values as normal data values.
- The measure of an elapsed time, for instance the delay between the occurrences of consecutive events, turned out to be very difficult in general. Timed LOTOS did not allow us to measure it in a satisfying way. Again, one had to use an approximation: time could not be measured precisely, but only with a potential error ε. And the trick to do it was quite complex [1].
- To express non deterministic delays and obtain the persistency and reverse persistency properties which were necessary for the definition of higher level operators, we introduced the special event θ. But this complicated the semantics, and we wanted to find a way to avoid it.

In the sequel we present ET-LOTOS, and explain how a few changes give an adequate response to the weaknesses evoked above.

3. DESCRIPTION OF ET-LOTOS

3.1. Time Domain

Like Timed LOTOS, ET-LOTOS is not restricted to a discrete time domain.

Another point, that we did not explicitly treat in Timed LOTOS, is the way time values should be defined. It is mandatory that time values can be treated as normal data values. Thus, they should be defined the same way, the time domain being the set (D) of values of a given data sort (time). This subject has been developed in [MFV92]. In this formalism, the specifier is allowed to design its time domain himself. But there is a minimal set of semantic elements, matching some properties, that (s)he must include. We retained this policy for ET-LOTOS.

In ET-LOTOS (on the basis of what is proposed in [MFV92]), the following elements must be defined over D:

- A total order relation, represented by “>“
- An element $0 \in D$ such that for each value $r \in D$ and $r \neq 0$: $r > 0$
- An element $\omega \in D$ such that for each value $r \in D$ and $r \neq \omega$: $\omega > r$
- A commutative and associative operation + : D, D --> D such that
  - For every $r, r_1 \in D$: $r_1 < r \Leftrightarrow \exists r' > 0$ such that $(r_1 + r') = r$
    This expresses that from a given instant, it is possible to reach any later instant.
    $r > 0$ and $r_1 \neq \omega \Rightarrow r + r_1 > r_1$
  - $0 \in D$ is an identity element of + such that for all value $r \in D$: $r + 0 = r$
  - $\omega \in D$ is an absorbent element of + such that for all value $r \in D$: $r + \omega = \omega$

In the sequel, D_0 will denote the set of all the values of sort time, except 0: $D_0 = D \setminus \{0\}$

3.2. Prefixing

The explanations given in this section sometimes make reference to the axioms and inference rules given in section 3.5.

Basically all the differences between ET-LOTOS and Timed LOTOS come from the new design of the action prefix. This operator is made more complex and complete in ET-LOTOS. The action prefix syntax is as follows: *g @t {d}; A*, where @t and *{d}* are optional.

First, let us present *{d}*, the “life reducer”. $d \in D$. It expresses that action *g* is offered for a *period* of time equal to d. In other words, the offer starts immediately but stops after a time d, if

---

[1] It would involve the use of the $\lfloor P \rfloor^{d}_{g}(Q)$ operator evoked above, and we will not reproduce it here

it has not been accepted yet[1]. Rules AP1 and AP2 show that *g{d};P* may accomplish *g* at any time before d.

What happens at time d depends on whether *g* is an external event or not. We remain true to our opinion that one should not try to enforce the occurrence of an external event by blocking the passing of time. So, process *g @t {d};A* may progress in time beyond time d. What happens then is that the g action is retracted and the process starts behaving like *stop* (rule AP3), which does not prevent the passing of time (rule S).

But *{d}* can also be applied to the event *i*. We still want to preserve the ability to enforce the occurrence of internal events. In Timed LOTOS the maximal progress rule is applied, that imposes an immediate occurrence to all internal events. *{d}* helps us to extend this rule: we define here that an internal event must occur within the delay expressed by the label. So, the passing of time after d is conditioned by the occurrence of *i* (or of a conflicting action if any). Rule AP3 only applies to the elements of L, i.e. not to *i*. Therefore, *i{0};P* cannot execute any timed transition. If d is given the value 0, we are back to the classical maximal progress. *{ω}* suppresses the obligation of occurrence.

The moment when *i* happens in *i{d};P* is before d but unpredictable. This is thus a way to introduce a non deterministic delay.

Now, let us come to the attribute *@t*. The construction *g@t* has been proposed by Wang Yi in [Wan91]. It indicates that the (relative) time at which *g* will occur will be recorded in variable *t* (*t* can be replaced by any other variable name). This time is the relative time from the moment when *g* began to be offered. In other words *t* will memorise the duration *g* has remained offered before occurring. *@t* is optional, its use can be restricted to the cases where it is useful.

3.3. Delay

In Timed LOTOS we presented $\Delta^{[d1,d2]}$, a non-determined-delay operator. The definition of such an operator is not simple. In practice it requires a way to indicate the moment when the delay *actually* expires (we will not explain more here, details can be found in [LeL93a]). This is done in Timed LOTOS by the special event θ. θ presents an advantage on *i* : its occurrence does not resolve a choice. One can thus prefix a process by a delay more easily, as it will not interfere with the environment. It is also possible with $\Delta^{[d1,d2]}$ to express a “classical”, precise delay, by equating both d_1 and d_2 to the desired value.

In ET-LOTOS, we have seen above that a way to express a non determined delay already exists: *i{d}*, where of course, the expiration is indicated by *i*. However, *i{d}* is not as expressive as $\Delta^{[d1,d2]}$: it would be “similar” to $\Delta^{[0,d]}$. Clearly, *i{d}* alone is not sufficient to express delays, in particular precise delays. So, we couple it with another delay operator: Δ^{d}. Δ^{d} expresses a delay of (determined) value d. It can be considered as a short notation for $\Delta^{[d,d]}$. But the absence of non determinism allows us to get rid of θ (as can be seen in rules D1,2).

The pair *i{d}* + Δ^{d} is as expressive as $\Delta^{[d1,d2]}$, because $\Delta^{[d1,d2]}$ is equivalent to $\Delta^{d1}\ i\{d_2 - d_1\}$. The shortcoming of this option is also its advantage: the nondeterminism is introduced via *i* instead of θ. As written above, this implies that the expiration of a delay resolves a choice if any. In practice, the operator obtained is thus less easy to use. For example, try to express a choice between two processes, both of them prefixed by a nondeterministic delay in the interval [2,5] whose expiration should not resolve the choice. This is obvious with $\Delta^{[d1,d2]}$: $\Delta^{[2,5]}P\ []\ \Delta^{[2,5]}Q$, but cannot be realised simply with *i{d}* and Δ^{d}. However, we think that such a problem is really marginal. Being able to express a non determined delay is a useful abstraction, but probably not of intensive use, and *i{d}* combined with Δ^{d} is already an interesting solution. To our knowledge indeed, no other timed extension of LOTOS proposes something better. So we retained this option, that helps us to get rid of the use of the special action θ, which is of great benefit for the simplicity of the semantics.

---

[1] One must not confuse the meaning of this label with the one introduced in [MFV92], where *a [d]* means that *a* is *only* offered at the punctual instant d after the moment the control arrived at *a [d]* .

3.4. Usual LOTOS Operators

Among the other LOTOS operators, we also extended **exit** with an optional life reducer, so that the successful termination action δ is treated like any other observable action. The semantics of all the other basic operators of Timed LOTOS are unchanged. Their meaning is quite simple and easy to understand from the inference rules we give in the next section. The reader is referred to `[LeL93a]` for more detailed explanations. Let us just recall that the maximal progress for the internal events created by hiding is ensured in the definition of the hide operator itself. Rule H3 shows that time is blocked if an internal event is possible. Let us also remark that we keep the maximal progress rule for these internal events created by hiding, i.e. these internal events must occur as soon as possible. Extending to **hide** the non determinism permitted by the action-prefix would have been too difficult and of less practical interest.

3.5. The Semantics of ET-LOTOS

In this section the axioms and inference rules for the basic operators of ET-LOTOS are given, which define the semantics of ET-LOTOS. We adopt the presentation of Moller and Tofts [MoT90], in two columns. It allows a clear separation between the occurrence of events and the passing of time.

Notations

L is the alphabet of observable actions. *i* and δ are special actions, respectively the internal action and the termination action. $L^{\delta} = L \cup \{\delta\}$, $L^{i} = L \cup \{i\}$, $L^{i,\delta} = L \cup \{i,\delta\}$, …

$P \xrightarrow{a} P'$	where $a \in L^{i,\delta}$, means that process *P* may engage in action *a* and, after doing so, behave like process *P'*.
$P \xrightarrow{a}$	means ∃ P' such that $P \xrightarrow{a} P'$
$P \overset{a}{\nrightarrow}$	where $a \in L^{i,\delta}$, means that ∄ P', such that $P \xrightarrow{a} P'$, i.e. *P* cannot accept (or must refuse) the action *a*.
$P \xrightarrow{d} P'$	where $d \in D_0$, means that process *P* may idle (i.e. not execute any action in $L^{i,\delta}$) during a period of *d* units of time and, after doing so, behave like process *P'*.
$P \overset{d}{\nrightarrow}$	where $d \in D_0$, means that ∄ P', such that $P \xrightarrow{d} P'$.

In the inference rules below, $\mathbf{d \in D_0}$ **and** $\mathbf{d_1 \in D}$, unless otherwise stated.

	(S) stop $\xrightarrow{d}$ stop
(AP1) $a\{d\}; P \xrightarrow{a} P$ $(a \in L^i)$	(AP2) $a\{d_1+d\}; P \xrightarrow{d} a\{d_1\}; P$ $(a \in L^i)$ (AP3) $a\{d1\}; P \xrightarrow{d}$ stop $(a \in L, d > d1)$
(TM1) $a@t\{d\}; P \xrightarrow{a} P\,[0/t]$ $(a \in L^i)$	(TM2) $a@t\{d_1+d\}; P \xrightarrow{d} a@t\{d_1\}; P\,[t+d/t]$ $(a \in L^i)$ (TM3) $a@t\{d1\}; P \xrightarrow{d}$ stop $(a \in L, d > d1)$
(D1) $\dfrac{P \xrightarrow{a} P'}{\Delta^0 P \xrightarrow{a} P'}$ $(a \in L^{i,\delta})$	(D2) $\Delta^{d1+d} P \xrightarrow{d} \Delta^{d1} P$ (D3) $\dfrac{P \xrightarrow{d} P'}{\Delta^0 P \xrightarrow{d} P'}$
(Ex1) $exit\{d\} \xrightarrow{\delta}$ stop	(Ex2) $exit\{d_1+d\} \xrightarrow{d} exit\{d_1\}$ (Ex3) $exit\{d_1\} \xrightarrow{d}$ stop $(d > d_1)$
(Ch1) $\dfrac{P \xrightarrow{a} P'}{P\ [\,]\ Q \xrightarrow{a} P'}$ $(a \in L^{i,\delta})$ (+ Ch1')	(Ch2) $\dfrac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q'}{P\ [\,]\ Q \xrightarrow{d} P'\ [\,]\ Q'}$

(PC1) $\dfrac{P \xrightarrow{a} P'}{P \vert[\Gamma]\vert Q \xrightarrow{a} P' \vert[\Gamma]\vert Q}$ $(a \in L^{i} - \Gamma)$ (+ PC1') (PC2) $\dfrac{P \xrightarrow{a} P', Q \xrightarrow{a} Q'}{P \vert[\Gamma]\vert Q \xrightarrow{a} P' \vert[\Gamma]\vert Q'}$ $(a \in \Gamma \cup \{\delta\})$	(PC3) $\dfrac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{P \vert[\Gamma]\vert Q \xrightarrow{d} P' \vert[\Gamma]\vert Q'}$
(H1) $\dfrac{P \xrightarrow{a} P'}{\text{hide } \Gamma \text{ in } P \xrightarrow{a} \text{hide } \Gamma \text{ in } P'}$ $(a \in L^{i,\delta} - \Gamma)$ (H2) $\dfrac{P \xrightarrow{a} P'}{\text{hide } \Gamma \text{ in } P \xrightarrow{i} \text{hide } \Gamma \text{ in } P'}$ $(a \in \Gamma)$	(H3) $\dfrac{P \xrightarrow{d} P', P \not\xrightarrow{a} \forall a \in \Gamma}{\text{hide } \Gamma \text{ in } P \xrightarrow{d} \text{hide } \Gamma \text{ in } P'}$
(En1) $\dfrac{P \xrightarrow{a} P'}{P >> Q \xrightarrow{a} P' >> Q}$ $(a \in L^{i})$ (En2) $\dfrac{P \xrightarrow{\delta} P'}{P >> Q \xrightarrow{i} Q}$	(En3) $\dfrac{P \xrightarrow{d} P', P \not\xrightarrow{\delta}}{P >> Q \xrightarrow{d} P' >> Q}$
(Di1) $\dfrac{P \xrightarrow{a} P'}{P [> Q \xrightarrow{a} P' [> Q}$ $(a \in L^{i})$ (Di2) $\dfrac{Q \xrightarrow{a} Q'}{P [> Q \xrightarrow{a} Q'}$ $(a \in L^{i,\delta})$ (Di3) $\dfrac{P \xrightarrow{\delta} P'}{P [> Q \xrightarrow{\delta} P'}$	(Di4) $\dfrac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{P [> Q \xrightarrow{d} P' [> Q'}$
(In1) $\dfrac{P[g_1/h_1,\ldots g_n/h_n] \xrightarrow{a} P', Q[h_1,\ldots h_n]:=P}{Q\,[g_1, \ldots g_n] \xrightarrow{a} P'}$ $(g \in L^{i,\delta})$	(In2) $\dfrac{P[g_1/h_1,\ldots g_n/h_n] \xrightarrow{d} P', Q[h_1,\ldots h_n]:=P}{Q\,[g_1,\ldots g_n] \xrightarrow{d} P'}$

Short notations

The life reducer *{d}* and the attribute *@t* are optional. By default, if g ∈ L, *g;P* means *g{ω};P*. This is consistent with the LOTOS intuition in which all actions are persistent and no life reducer exists. Similarly *exit* means *exit {ω}*. On the other hand, by default, *i;P* means *i{0};P*. This allows us to preserve in ET-LOTOS the observation equivalence between the LOTOS processes *hide g in g;P* and *i; hide g in P*. We also introduce the notations $g\{d_1,d_2\};P$ to mean $\Delta^{d_1}g\{d_2-d_1\};P$ and $g@t\{d_1,d_2\};P$ for $\Delta^{d_1}g@t\{d_2-d_1\};P[t+d_1/t]$. The last definition means that *@t* starts to count when the control arrives at $g@t\{d_1,d_2\};P$, and not when g actually begins to be offered. We made this choice because we think it is more intuitive, and because it complements the basic construction $\Delta^{d_1}a@t\{d_2-d_1\};P$, where *t* does not include the delay d_1.

4. AN EXAMPLE OF THE USE OF ET-LOTOS

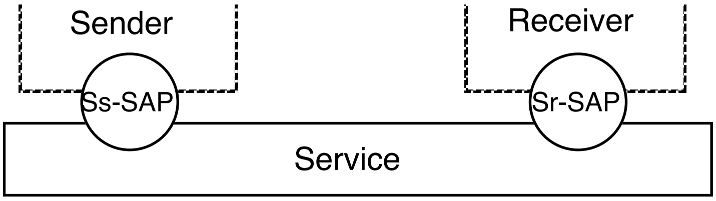
In order to justify our design choices and prove their advantages, we will illustrate here the expressiveness of ET-LOTOS by applying it to the specification of a small system, taken from the Tick-Tock case study [LLD93], which has been specially designed to assess timed FDTs. Some features have been added to the original version to present a more complete overview of the capabilities of ET-LOTOS. Other interesting examples are presented and specified in [LeL93b].

4.1. Description of the Case Study [1]

The case study consists of a service, named *service* in the sequel. To keep things simple, the *service* specification is restricted to its interactions with just two users, *sender* and *receiver*, through their respective S-SAPs. *Service* transmits data from *sender* to *receiver*. Let us recall that *service* is focused on the assessment of timed FDTs. It tries to propose a realistic environment, but it is not a real system and its definition has been (over)simplified of all the details that were not relevant to timing aspects.

One will use, in the sequel, a referential time unit, simply called “unit”. Note however that this “unit” is not an elementary grain of time: the times expressed in the following description could be fractions of a unit.

Under these conditions, *service* can be characterised as follows:



Sketch of the system

Service Primitives: They carry a data cell as parameter. Primitives are instantaneous and atomic events. Our system is so simplified (the exchanges are always done between the same two S-SAPs) that no other parameter needs to be specified. In the sequel, these exchanges between the service and its users will simply be referred to as cells instead of primitives.

Isochronism: The given service is isochronous: a cell from *sender* is only accepted at some precise, punctual instants, that follow one another regularly in time, with a given period. An opportunity of transmission can be neglected by *sender*. Just one cell can be exchanged at any instant.

Adaptation of the Access Period to Service: The period between two consecutive interaction offers made by *service* may vary in time. The aim is to adapt the access to *service* to the presumed needs of *sender*, estimated from the use *sender* makes of the actual capabilities it has at its disposal.

The mechanism proposed here segments the stream of proposals into consecutive and separated sequences of 10 proposals, at the end of which *service* is allowed to modify the period. Two main rules apply:

- at the end of a sequence of successive transmissions of 10 cells (all the offers having been accepted), the period is divided by two.
- at the end of 3 consecutive sequences, none of which being already taken into account by the previous or the present rule, the period is multiplied by a coefficient determined by the following table, according to the number of proposals effectively used among the 30 ones.

Number of proposals used	Coefficient
0 → 15	3/2
16 → 30	6/5

Bounds are however imposed on the possibilities of variation of the period, which must always remain between η and ψ units. Initially, the period is supposed to be equal to π units.

---

[1] This subsection is mainly made of excerpts from [LLD93], with slight changes

Transmission Delays: A cell is always proposed to receiver between τmin and τmax after its transmission.
Immediate Acceptation: A cell offered to *receiver* must be immediately accepted by *receiver*. If it is not the case, *service* loses the cell immediately.
Spacing Between the Deliveries: There is always a delay of at least α units between two successive offers of cells at Sr-SAP.
"Crash" of Service: At any instant, without any reason, *service* may "crash". All the cells in transit are then lost. *Service* only restarts if a delay of at least γ units has occurred since its previous (re)start. In this case, *service* needs an unpredictable delay in the interval $[\delta, \phi]$ before restarting. It restarts free of any cell and with a period of π. It does not restart and stops all activity if the delay is smaller than γ.
Loss Free Transmission: The previous two points describe the only way a cell received from *sender* can be lost: no cell is lost in transit through *service*.
FIFO-Ordering of Cells: The cells arrive in their transmission order.

The last two constraints - "Loss free transmission" and "FIFO-ordering of cells" - are not strictly necessary and might seem less realistic. However, they help to avoid unnecessary complications in the specification of *service*.

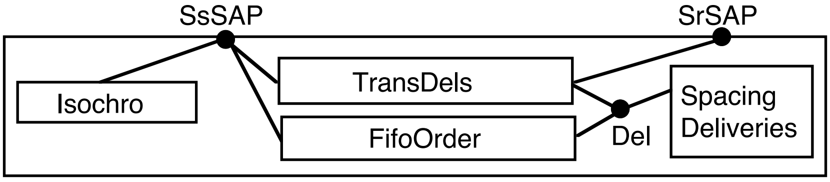
Let us notice that there is no incompatibility between the constraint "transmission delays" and the constraints "spacing between the deliveries" and "FIFO-ordering of cells", as long as the minimal period of admission η is greater or equal to the minimal delay between two deliveries α. We will suppose that this is the case in the sequel.

4.2. ET-LOTOS Specification of *Service*

We have tried to describe *service* in the constraint oriented style [Bri89], to get a structured specification. This requirement is often neglected but, from our experience, it turns out to be an additional difficulty with the existing timed formalisms. It imposes to describe, as far as possible, the various features of *service* by different processes, and it requires to avoid internal synchronisation, i.e. the use of the **hide** operator. But we think that a timed extension of LOTOS should preserve the ability to specify in a given style, and we wanted to test ET-LOTOS against this problem.

As we will see, we did not succeed in our attempt. We managed to separate the main service features into distinct constraints, but we had to introduce an internal synchronisation.

The picture below shows the general structure of the specification, represented by process `Service` :



```
process Service [SsSAP,SrSAP] : noexit :=
( Isochro [SsSAP] (π,0,0,0,0) |[SsSAP]|
   (hide Del in    (TransDels [SsSAP,SrSAP,Del] |[SsSAP,Del]|
                    FifoOrder [SsSAP,Del] (NoCell) |[Del]|
                    SpacingDeliveries [Del] )))

[> Crash [SsSAP,SrSAP]
endproc (* Service *)
```

It consists of five sub-processes:

- `Isochro` is local to the SsSAP. It expresses the isochronism of *service* at the SsSAP, and the way the period varies in time.
- `TransDels` describes the constraint on the transmission delay for each cell, i.e. a cell is always delivered between τmin and τmax after its transmission.
- `FifoOrder` expresses that the cells are delivered in their transmission order.

- `SpacingDeliveries` expresses the minimal delay α between successive deliveries at a same SsSAP.
- `Crash` describes the effects of a crash of the system.

As we can see, `TransDels`, `FifoOrder` and `SpacingDeliveries` synchronise on the internal gate `Del`. The reason for this will appear more clearly in the individual presentation of each process. This gate is used to express that each cell proposed at SrSAP must be accepted immediately or lost. We found no way to express this constraint by an independent process, or to integrate it in one of the three above mentioned ones. This would require, for the process in charge of this constraint, the ability to determine the moment when the cell is actually proposed, in order to abort the offer if it is not accepted immediately. The problem is that this moment depends on the conjunction of the effects of the three processes. None of them knows this moment by itself, and no other concurrent process could. The only way for a process to get information about the state of the others is by interacting with them. Interacting on the external event, i.e. the transmission of the cell, is of course of no use when the aim is to determine when this action should have occurred, but did not. We thus had to introduce an internal synchronisation on `Del`. When a process is ready to transmit a cell, it proposes `Del`. As `Del` is hidden, it is supposed to occur as soon as it becomes possible, i.e. when all three processes are ready for it. The occurrence of `Del` thus signals to the processes that the transmission of the cell is possible and thereby they know that the cell must be transmitted immediately or never.

This problem enlightens a weakness of ET-LOTOS, but to our knowledge, no other timed extension of LOTOS could do better. The solution could lie in the definition of “higher” level operators, we will come back to this point in the next chapter. This example also illustrates the utility of the urgency on hidden events. If `Del` had been free to occur anytime, one would have lost the information about the moment when it became possible, and the specification of the constraint would have been impossible (or so complicated we do not even want to think about).

Let us now examine the sub-processes one by one to see how (well) ET-LOTOS copes with them. In the sequel we use data types, in particular the sort time. Their definition in Act One is quite classical and does not present special difficulties, so that we will not give it here because we lack of space.

```
process Isochro [SsSAP]: noexit:=
SsSAP{0} ?c:cell;  Isochro2 [SsSAP] (π, 1,1,1,1)
[] Isochro2 [SsSAP] (π, 1,0,1,0)
endproc (* Isochro *)

process Isochro2 [SsSAP] (per:time, slot,transm,slot1,transm1:Nat) : noexit:=
Δper  (SsSAP{0} ?c:cell;
        ([not(slot eq 9)] ->                Isochro2 [SsSAP](per,succ(slot),succ(transm),succ(slot1),succ(transm1))
         [] [(slot eq 9) and (transm eq 9)] -> Isochro2 [SsSAP](min(per/2,η),0,0,0,0)
         [] [(slot eq 9) and not(transm eq 9) and not(slot1 eq 29)] ->
                                               Isochro2 [SsSAP](per,0,0,succ(slot1),succ(transm1))
         [] [(slot1 eq 29) and not(transm eq 9)] ->
                    ([transm1 le 15] ->        Isochro2 [SsSAP](max(per*3/2,ψ),0,0,0,0)
                     []
                     [transm1 ge 16] ->        Isochro2 [SsSAP](max(per*6/5,ψ),0,0,0,0) )
      []
        ([not(slot eq 9)] ->                   Isochro2 [SsSAP](per,succ(slot),transm,succ(slot1),transm1)
         [] [(slot eq 9) and not(slot1 eq 29)] -> Isochro2 [SsSAP](per,0,0,succ(slot1),transm1)
         [] [slot1 eq 29] ->
                    ([transm1 le 15] ->        Isochro2 [SsSAP](max(per*3/2,ψ),0,0,0,0)
                     []
                     [transm1 ge 16] ->        Isochro2 [SsSAP](max(per*6/5,ψ),0,0,0,0) ) ))
endproc (* Isochro2 *)
```

`Isochro2` begins with a delay of value `per`. Then, like in `isochro`, it is a choice between two possible behaviours, and this choice is resolved immediately. SsSAP{0} expresses the offer of service to *user*. The label {0} ensures that this offer is punctual. So, either the offer is accepted immediately, or it behaves like `stop`. In the first case, a new occurrence of `Isochro2` is called, with the parameters changed to take account of the receipt of a new cell. In the second case, a new occurrence of `Isochro2` is called, with the parameters changed to take account of the rejection of the offer. In particular, in both cases the new period (`per`) is calculated and it is according to this new value that the next occurrence of `Isochro2` is delayed. `slot` and `slot1` respectively count the number of slots (or available offers) on successive sequences of ten and thirty, whereas `transm` and `transm1` count the number of cells actually transmitted during these sequences. Finally `per` is the period.

`Isochro` has been divided into two processes to avoid a delay at the beginning.

This example shows well the use and the expressiveness of the 'life reducer'.

```
process TransDels [SsSAP,SrSAP,Del] : noexit :=
SsSAP?c:Cell ; (TransDel [SrSAP,Del] (c) ||| TransDels [SsSAP,SrSAP,Del] )
endproc (* TransDels *)

process TransDel [SrSAP,Del] (c:Cell) : noexit :=
i{τmin,τmax} ; Del!c ; SrSAP{0}!c ; stop
endproc (* TransDel *)
```

For each cell, `TransDel` expresses the nondeterminism in the transmission delay, that can be chosen anywhere between τmin and τmax. The occurrence of `i` activates `Del`, but it is not sure that `Del` may happen immediately. It could be delayed because of the constraint expressed by `SpacingDeliveries`. However, as the minimal period of admission η is supposed to be greater or equal to the minimal delay between two deliveries α, this additional delay will never cause `Del` to occur after τmax.

```
process SpacingDeliveries [Del] : noexit :=   Del?c:Cell ; Δ^α SpacingDeliveries [Del]
endproc (* SpacingDeliveries *)
```

This process is very simple. It just takes care that two successive occurrences of `Del` be spaced out by at least α time units.

```
process FifoOrder [SsSAP,Del] (fifo:FifoQueue) : noexit :=
SsSAP?c:Cell ; FifoOrder [SsSAP,Del] (Append(c,fifo))
[] [not(IsEmpty(fifo))] -> Del!TopOf(fifo) ; FifoOrder [SsSAP,Del] (Cut(fifo))
endproc (* FifoOrder *)
```

`FifoOrder` is very simple too. It just uses a FIFO queue to ensure that all the cells be delivered in their transmission order.

```
process Crash [SsSAP,SrSAP] : noexit :=    i@ft{ω}; [ft gt γ] -> i{δ,ϕ}; Service [SsSAP,SrSAP]
endproc (* Crash *)
```

`Crash` illustrates the use of the time measurement mechanism. The first `i` is free to occur at any time, but the time at which it occurs is stored in the variable `ft`. According to this value, `Crash` decides if it restarts `Service` or not. If it does, the restart time is nondeterministically chosen in the interval $[\delta,\phi]$, which is expressed with the short notation `i{δ,ϕ}`.

4.3 Conclusion

We can see that ET-LOTOS succeeded in describing all the proposed mechanisms, but not in a fully constraint oriented style (but to our knowledge no other timed extension of LOTOS could do better). It did it in a light and clear way, with quite simple operators. In particular, the 'life reducer' appears to be really powerful.

We have just described here a slightly modified excerpt of the Tick-Tock case study, as an illustration of the use of ET-LOTOS. We are not the first ones to assess our formalism with the Tick-Tock case study. Miguel, Fernandez and Vidaller have already applied their T-LOTOS to the specification of the whole system [MFV93]. In their T-LOTOS, the action prefix has a time stamp *[d]*. *g[d]* means that action *g* may *only* occur *at* the relative time d. No special delay operator exists as *[d]* already expresses one. With this simple extension, together with the maximal progress rule and no urgency on external events, T-LOTOS can give a quite good specification of the Tick-Tock case study. But if we compare the common parts of our respective works, the specification obtained with ET-LOTOS is lighter. The main weakness of T-LOTOS is that the label can only express punctual offers, and not intervals. Therefore, they often rely on the `choice` operator to simulate a continuous offer. But a problem arises when they try to express non deterministic delays, like what we do with *i{d}*. A choice between several *i[d]*, with different values of d, is not possible because the maximum progress rule is such that the first possible `i` will always be executed. So a more complicated and less readable mechanism has been used to circumvent the problem. The lack of a specific delay operator also leads to some difficult situations, especially in the description of the adaptive isochronous period, when one wants to delay the occurrence of a whole process. Finally, T-LOTOS has no special construct like @t to measure the passing of time. Such a measurement can however be done, but again with a mechanism that requires the use of the `choice` operator. In conclusion, in many situations, ET-LOTOS appears to be more flexible and expressive.

5 HOW CAN 'HIGH LEVEL' OPERATORS BE DEFINED IN ET-LOTOS?

An important part of the paper [LeL93a] about Timed LOTOS was concerned with the definition of 'high level' operators. 'High level' qualifies operators able to express constraints on the occurrence of interactions. For instance **hide**, as it is defined in ET-LOTOS, may be considered as being 'high level'. In the example of chapter 4, **hide** allowed us to ensure that the synchronisation on `Del` between `SpacingDeliveries`, `TransDels` and `FifoOrder` would occur as soon as possible. It would not have been possible to express such a constraint only with the other operators. But **hide** always turns the events to which it applies into internal events, and it can only impose an immediate occurrence. One could have more various needs. For instance, in [BoL92], a timer operator is proposed that allows both the introduction of a delay and a reduction of the offer of any external event, including interactions.

In [LeL93a] we introduced $\Delta_g^I(P)$ and $\lfloor P\rfloor_g^d(Q)$. $\Delta_g^I(P)$ delays by a time chosen in a nondeterministic way into the interval I, the moment when external action g is proposed to the environment from the moment when it was actually proposed by process P. $\lfloor P\rfloor_g^d(Q)$ is a time-out: if P has been proposing g for a period d, with no success, P is interrupted and Q is started.
We also explained how the persistency and reverse persistency properties[1], verified by the semantics, made the definition of the two operators easier, and how these properties were obtained by the introduction of the special event θ in the semantics.

However, in this section, we will present a better solution to this problem, that avoids the use of θ and requires weaker properties from the semantics than the persistency and reverse persistency. This solution is based on a semantics slightly different from the one we already presented in section 3.5. The reason why we did not immediately present this last semantics lies in the nature of the problem it tackles. In fact, we still miss of practical experience to be sure that other 'high level' operators than **hide** are really necessary. So our aim is not to present this new, and a little bit more complex, semantics as the standard semantics for ET-LOTOS, but to show how the simple version could be slightly changed to ease the definition of 'high level' operators if one ever had to.

5.1 A semantics that allows easy definitions of 'high level' operators

The first change from the rules given in section 3.5 is that, in this second semantics, the time transition $\xrightarrow{0}$ is possible, what was not the case before. In other words, one should consider, when reading the rules that d_1 **and d** $\in$ D. Otherwise, rules S, Ch, PC, H, En, Di and In remain unchanged and will not be repeated here. The new version of the other rules is given in

---

[1] The persistency property expresses that: $\forall$ P, P' $\forall d \in D$, $\forall g \in L^{i,\delta}$, $(P \xrightarrow{d} P' \wedge P \xrightarrow{g} \Rightarrow P' \xrightarrow{g})$
and the reverse persistency, that: $\forall$ P, P' $\forall d \in D$, $\forall g \in L^{i,\delta}$, $(P \xrightarrow{d} P' \wedge P \overset{g}{\not\rightarrow} \Rightarrow P' \overset{g}{\not\rightarrow})$

the next table. These changes will be justified in point 5.2. The main novelty lies in the definition of the life reducer. Rule AP3' shows that *g{0};P* will only progress in time after having been turned into *stop*, by a $\xrightarrow{0}$ transition. In other words, every expiration of a life reducer is now signalled by a $\xrightarrow{0}$ transition. But basically the meaning of the operator is unchanged.

In rules AP2' (and TM2', D2', Ex2'), we impose that $d_1+d \in D_0$ (i.e. $d_1+d \neq 0$), to keep the time determinacy property. We want to avoid that from *g{0}; P*, the transition $\xrightarrow{0}$ could lead to two different states: either *g{0}; P* again (what we do not want), or *stop*.

(AP1') $a\{d\}; P \xrightarrow{a} P$ $(a \in L^i)$	(AP2') $a\{d_1+d\}; P \xrightarrow{d} a\{d_1\}; P$ $(a \in L^i, d_1+d \in D_0)$ (AP3') $a\{0\}; P \xrightarrow{0} stop$ $(a \in L)$
(TM1') $a@t\{d\}; P \xrightarrow{a} P\,[0/t]$ $(a \in L^i)$	(TM2') $a@t\{d_1+d\}; P \xrightarrow{d} a@t\{d_1\}; P\,[t+d/t]$ $(d_1+d \in D_0)$ (TM3') $a@t\{0\}; P \xrightarrow{0} stop$ $(a \in L)$
(D1') $\dfrac{P \xrightarrow{a} P'}{\Delta^0 P \xrightarrow{a} P'}$ $(a \in L^{i,\delta})$	(D2') $\Delta^{d1+d} P \xrightarrow{d} \Delta^{d1} P$ $(d_1+d \in D_0)$ (D3') $\dfrac{P \xrightarrow{d} P'}{\Delta^0 P \xrightarrow{d} P'}$
(Ex1') $exit\{d\} \xrightarrow{\delta} stop$	(Ex2') $exit\{d_1+d\} \xrightarrow{d} exit\{d_1\}$ $(d_1+d \in D_0)$ (Ex3') $exit\{0\} \xrightarrow{0} stop$

5.2 Some properties of the semantics

First ET-LOTOS has the time determinacy property in both semantics:

$\forall\ P, P', P'', \forall d \in D, (P \xrightarrow{d} P' \wedge P \xrightarrow{d} P'' \Rightarrow P' \equiv P'')$

It also has the **predecessor closedness** property on timed transitions:

$\forall\ P, P'', \forall d_1 \in D, d_2 \in D_0, (P \xrightarrow{d_1+d_2} P'' \Rightarrow \exists P', P \xrightarrow{d_1} P' \wedge P' \xrightarrow{d_2} P'')$

But, and again in both semantics, we can notice that the passing of time is not additive. This property of additivity is defined as follows:

$\forall\ P, P'', \forall\ d_1, d_2 \in D, (\exists\ P', P \xrightarrow{d_1} P' \wedge P' \xrightarrow{d_2} P'' \Leftrightarrow P \xrightarrow{d_1+d_2} P'')$

But getting the additivity is precisely what we do not want. What we intend to do in this section is to show how and why breaking the time additivity can be useful.

In the basic semantics, the time additivity is solely broken by the delay operator: $\Delta^d P \xrightarrow{d} \Delta^0 P$ and $\Delta^0 P \xrightarrow{d_1} P'$ does not imply $\Delta^d P \xrightarrow{d+d_1} P'$. In fact, the moment when a process possibly gains the ability to perform an action, by the expiration of a delay, is a break in the additivity of time. A process may idle until such a moment, and then from it, but cannot pass beyond it without “stopping”. This property can be expressed mathematically:

Property 1: $\forall\ P,P', \forall d \in D, \forall g \in L^{i,\delta}, (P \not\xrightarrow{g} \wedge P \xrightarrow{d} P' \wedge P' \xrightarrow{g} \Rightarrow \forall d_1 \in D, d_1 > d, P \not\xrightarrow{d_1})$

It is already useful to define the **hide** operator. **Hide** enforces the immediate occurrence of the events it hides. This implies that it must not idle beyond the moment when such an event becomes possible. Property 1 ensures that it will not .

In the new semantics, time additivity is also broken by the life reducer: *g{0+d};P* $\xrightarrow{d}$ *g{0};P* and *g{0};P* $\xrightarrow{0}$ *stop* do not imply *g{d};P* $\xrightarrow{d}$ *stop*. In fact, this operator does more than just breaking time additivity: it also enforces the occurrence of a zero-time transition. In other words, with this semantics, a process may not idle beyond the moment when one of its actions stops being possible because of the expiration of a life reducer: it must stop **and** accomplish a zero-time transition, before continuing to idle. This mechanism allows us to get the next property:

Property 2: $\forall$ P, P' $\forall d \in D$, $\forall g \in L^{i,\delta}$, $(P \xrightarrow{d} P' \wedge P \xrightarrow{g} \wedge P' \overset{g}{\not\rightarrow} \Rightarrow d = 0)$
which expresses that a process that idles never loses the ability to perform an action, except if it idles for a 'zero duration period'.

From properties 1 and 2, other properties can be deduced.

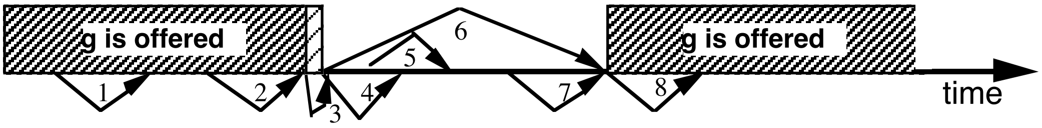
Property 3: $\forall$ P,P", $\forall d_1 \in D$, $\forall g \in L^{i,\delta}$, $(P \overset{g}{\not\rightarrow} \wedge P \xrightarrow{d_1} P'' \wedge P'' \xrightarrow{g} \Rightarrow \forall P', \forall d_2 \in D, (d_2 < d_1 \wedge P \xrightarrow{d_2} P' \Rightarrow P' \overset{g}{\not\rightarrow}))$

Property 4: $\forall$ P,P", $\forall d_1 \in D$, $\forall g \in L^{i,\delta}$, $(P \overset{g}{\not\rightarrow} \wedge P \xrightarrow{d_1} P'' \wedge P'' \overset{g}{\not\rightarrow} \Rightarrow \forall P', \forall d_2 \in D, (d_2 \leq d_1 \wedge P \xrightarrow{d_2} P' \Rightarrow P' \overset{g}{\not\rightarrow}))$

Property 5: $\forall$ P,P", $\forall d_1 \in D$, $\forall g \in L^{i,\delta}$, $(P \xrightarrow{g} \wedge P \xrightarrow{d_1} P'' \wedge P'' \xrightarrow{g} \Rightarrow \forall P', \forall d_2 \in D, (d_2 \leq d_1 \wedge P \xrightarrow{d_2} P' \Rightarrow P' \xrightarrow{g}))$

Property 3 says that if a process P is unable to accomplish an action g, but can idle to a state P" where it is able to do g, then all the states through which it passed before P" when idling were also unable to do g. In other words, P" is the first state able to accomplish g. Property 4 says that if a process may idle from a state where it was unable to accomplish an action g to a state where it is still unable to do so, than all the states in the meantime were unable too. Property 5 says that if a process may idle from a state where it was able to accomplish an action g to a state where it is still able to do so, then all the states in the meantime were able to do so.

Properties 2, 3, 4, 5 cover all the four possibilities a process P can face with a given action when it idles: either it loses the capability to perform it, or it gains it, or it remains unable, or it remains able. These four properties 2 to 5 are collectively referred to as the **atomicity properties** because they ensure together that action offers can only appear or disappear at the end of a timed transition, and NOT "during" a timed transition. We try to give an image of what these properties mean in the next picture:



The moments below the striped surfaces are the ones when a process may accomplish a given action, and the moments on the bold line are when it cannot. The arrows represent the possibilities to idle. The surface with lighter stripes should be very small, or should not even exist, because it represents the period of no duration that happens at the end of the offer of g. At the beginning of this surface, g is possible, but at the end, it no longer is. Arrow 3 represents the transition $\xrightarrow{0}$.

For instance, the picture above could describe the process: P:= g{5};A [] Δ^{10} g;B. In the sequel, we will denote P_d the state reached by P after idling for a time d. Hence P_0 := P. *Arrow 1* could be: P_2:= g{3};A [] Δ^8 g;B $\xrightarrow{1}$ P_3:= g{2};A [] Δ^7 g;B. *Arrow 2*: P_4:= g{1};A [] Δ^6 g;B $\xrightarrow{1}$ P_5:= g{0};A [] Δ^5 g;B. *Arrow 3*: $P_5 \xrightarrow{0} P_{5'}$:= Δ^5 g;B. *Arrow 4*: $P_{5'} \xrightarrow{2} P_7$:= Δ^3 g;B. *Arrow 5*: P_6:= Δ^4 g;B $\xrightarrow{2}$ P_8:= Δ^2 g;B. *Arrow 6*: $P_{5'} \xrightarrow{5} P_{10}$:= Δ^0 g;B. *Arrow 7*: P_9:= Δ^1 g;B $\xrightarrow{1}$ P_{10}. *Arrow 8*: $P_{10} \xrightarrow{1} P_{11}$:= g;B.

This picture also helps understand why we oblige a process to accomplish a $\xrightarrow{0}$ transition at the expiration of a life reducer. Without the break caused by the $\xrightarrow{0}$ transition, arrow 6 would be leaving from state P_5, where g is offered, to state P_{10} where it is offered again, whereas the offer is interrupted somewhere in the meantime. This situation would contradict property 1, and hence, property 5. We will see in section 5.3 why it is important that these properties be valid.

On the other hand, introducing such an obligation at the expiration of a delay would have been a useless complication, and even a drawback. For instance, with the process $i;P\ []\ \Delta^0\ i;Q$, as i has the priority on any timed transition, even $\xrightarrow{0}$, the first i would always gain the choice. Thus, this would introduce an undesired distinction between processes prefixed by Δ^0 or not.

5.3 Usage of the atomicity properties

Having these properties allows an easy definition of "high level" operators that have more complex semantics than **hide** (for **hide** property 1 was enough). For example, let us consider $\lfloor P \rfloor_g^d(Q)$, that we introduced in Timed LOTOS and whose inference rules are given in the table below. As soon as P begins to propose g, $\lfloor P \rfloor_g^d(Q)$ starts a timer, and if g remains proposed without being accepted for a time d, it gives the control to process Q.

The $\lfloor P \rfloor_g^d(Q)$ operator is defined via an intermediate operator, denoted $\lfloor P \rfloor_g^{d,d'}(Q)$, according to the following definition: $\lfloor P \rfloor_g^d(Q) := \lfloor P \rfloor_g^{d,d}(Q)$. The second argument d', referred to as the counter, is used to count down the remaining time until the expiration of the time-out on g.

The rules in the first column of the table are quite simple to understand:

- **Tg1** expresses that, when g is not offered at the same time as action a, the execution of action a resets the counter to d.
- **Tg2** expresses that, when g is offered at the same time as action a, the execution of action a has no effect on the counter.
- **Tg3** expresses that the execution of g resets the counter to d.

The rules for the passing of time are of more concern. $\lfloor P \rfloor_g^d(Q)$ must carefully observe P to know how the offer of g evolves in time. In particular, two points are of importance:

- It must notice the moment when P begins to offer g, in order to start the count.
- It must also make sure of the continuity of the offer of g: if g stops being offered for a period of time, the timer must be reset.

So the problem is how $\lfloor P \rfloor_g^d(Q)$ can get this information easily. The atomicity properties ease its task a lot. Let us imagine that we have the time additivity. So, for instance, a process P may idle beyond the moment when an action becomes possible. It may also idle from a moment where an action was possible to another such moment, but without noticing that an interruption of the offer occurred in the meantime. In fact, in this case, the behaviour of P gives no information to $\lfloor P \rfloor_g^d(Q)$. It cannot draw conclusions just by observing the state from which it came or to which it arrived. To decide if it agrees to idle, it should test all the states in the meantime. That is what can be avoided when the atomicity properties are fulfilled, which is the case in ET-LOTOS. In ET-LOTOS, $\lfloor P \rfloor_g^d(Q)$ knows that there is no risk to miss the moment when P begins to offer g, because it cannot idle beyond it (property 1, or 3 and 4). And for the same reason, it also knows that there is no risk to miss an interruption of the offer (property 5). If we had not obliged each process to accomplish a $\xrightarrow{0}$ transition at the expiration of a life reducer, it would have been impossible to distinguish between the transitions described by arrows 6 and 1 on the picture of section 5.2 only from an observation of the initial and final states. Making the difference would have required the test of a state in the meantime.

In fact, with the atomicity properties, the rules below show that the observation of the initial state is enough to decide how $\lfloor P \rfloor_g^d(Q)$ must behave.

- **Tg4** expresses that the counter is reset when one idles from a state where g was not offered.
- **Tg5** expresses that the counter is decreased when one idles from a state where g was offered.
- **Tg6** expresses that, when g is enabled, the timer expires if the counter has reached the value 0. The premise is needed to avoid the expiration of the timer when the g offer has just disappeared whereas the counter has not been reset yet. This rule is interesting, because the expi-

ration of the timer introduces a new case, where simultaneously some offers (from P) may disappear and other offers (from Q) appear. In order to keep the atomicity properties valid, the expiration is signalled by a zero-time transition, like at the expiration of a life reducer.

In the rules below, we always have: $d,d',d_1 \in D$.

(Tg1) $\dfrac{P \xrightarrow{a} P',\ P \not\xrightarrow{g}}{\lfloor P \rfloor_g^{d,d'}(Q) \xrightarrow{a} \lfloor P' \rfloor_g^{d,d}(Q)}$ $(a \in L^{i,\delta} \setminus \{g\})$	(Tg4) $\dfrac{P \xrightarrow{d_1} P',\ P \not\xrightarrow{g}}{\lfloor P \rfloor_g^{d,d'}(Q) \xrightarrow{d_1} \lfloor P' \rfloor_g^{d,d}(Q)}$
(Tg2) $\dfrac{P \xrightarrow{a} P',\ P \xrightarrow{g}}{\lfloor P \rfloor_g^{d,d'}(Q) \xrightarrow{a} \lfloor P' \rfloor_g^{d,d'}(Q)}$ $(a \in L^{i,\delta} \setminus \{g\})$	(Tg5) $\dfrac{P \xrightarrow{d_1} P',\ P \xrightarrow{g}}{\lfloor P \rfloor_g^{d,d'+d_1}(Q) \xrightarrow{d_1} \lfloor P' \rfloor_g^{d,d'}(Q)}$ $(d'+d_1 > 0)$
(Tg3) $\dfrac{P \xrightarrow{g} P'}{\lfloor P \rfloor_g^{d,d'}(Q) \xrightarrow{g} \lfloor P' \rfloor_g^{d,d}(Q)}$	(Tg6) $\dfrac{P \xrightarrow{g}}{\lfloor P \rfloor_g^{d,0}(Q) \xrightarrow{0} Q}$

5.4 Comparison with Timed LOTOS

In Timed LOTOS [LeL93a] a special event θ was needed to define $\Delta_g^I(P)$ and $\lfloor P \rfloor_g^d(Q)$. θ was signalling the expiration of a delay, or of a time-out. It was giving us the persistency and reverse persistency properties. These properties are stronger than the atomicity properties of ET-LOTOS. At this time, we were believing that they were necessary. It is now clear that atomicity properties are sufficient. Reverse persistency implies that while idling, a process never gains the ability to perform a new action (except θ). Such a change can only happen by way of the occurrence of events. It is not verified in ET-LOTOS. For instance: $\Delta^1\ g;B \xrightarrow{1} \Delta^0\ g;B$ is a counter-example. But in fact, it is enough that the process never idles beyond the first instant of the offer, what is ensured by properties 1, 3, 4.

As regards the persistency, the $\xrightarrow{0}$ transition that we introduced at the expiration of a life reducer is in fact similar to a transition made by θ. But it presents some advantages. $0 \in D$, whereas θ is an event. It was not possible to give θ an inferior priority than *i*, it is done naturally for 0. Basically, it would not have been possible to define correctly punctual offers with θ instead of 0. We know that, at the same time, such punctual offers may either occur, or turn into stop. With a $\xrightarrow{0}$ transition the first opportunity has the priority, whereas with a $\xrightarrow{\theta}$ transition both have the same priority, and the occurrence of $\xrightarrow{\theta}$ could prevent a possible interaction.

So, we see that the mechanism we introduced to break the time additivity is as interesting as the introduction of θ for the definition of high level operators, and even allows us to preserve the new constructs like the life reducer, that we introduced with the simple semantics and that appear to be of paramount importance in the expressiveness of ET-LOTOS. A possible drawback could be that the second semantics generates, in almost every state, a loop of zero-time transitions. Determining whether this would really be annoying requires further study.

6 CONCLUSION

We have presented ET-LOTOS, that responds to many of the features we wanted to improve in Timed LOTOS. Its application to a case study was conclusive. What makes its strength is the pair 'life reducer + delay operator'. Especially, the life reducer is interesting because it allows the expression of action offers of all kinds of duration, from zero to infinity. Coupled with *i*, it gives a way to express easily non deterministic delays. A delay operator complements the life reducer, which cannot be used to express precise delays. Having such a distinct delay operator (which is not attached as a label to an action) is an important advantage. This operator delays a process without having to delay each of its actions one by one. In particular, if a given process P has been instantiated, one can delay it simply: $\Delta^d\ P$, without having to rewrite P.

A weakness of ET-LOTOS lies in its inability to fully support a 'pure' constraint oriented style. This point is under study. To our knowledge, no other formalism could do better. But we have shown how the semantics could be designed to allow an easy definition of 'high level' operators, that could provide an answer.

As regards the high level operators, we must compare our approach with [BLT90]. They present an elegant mechanism of aged actions that allows a simplification of the definition of high level operators. However, we think that this mechanism is not sufficient. The problem is that aged actions introduce, as a side effect, a difference between different occurrences of the same action proposed by a given process. For example, a process like: $g\{5\};P\ []\ \Delta^5\ g;P$, would not be considered equivalent to $g;P$, because in the former the age of g is reset at time 5. We think that this approach is not abstract enough. It is not coherent with the philosophy of LOTOS, where a process should be considered as a black box, with no concern on how interactions are proposed. Moreover the mechanism of aged actions requires that, in their semantics, actions be extended with a time label, that is their age. This complication is not necessary in ET-LOTOS.

ET-LOTOS is compared with T-LOTOS in section 4.3. Let us also mention RT-LOTOS [CCE93], inspired by Timed LOTOS, in which a new operator is proposed to start an exception behaviour when a time constraint on some external action is not matched by the environment.

The presence of negative premises in our semantics is harmless: the semantics is consistent and strong bisimulation is a congruence. The proofs are sketched in [LeL93] for Timed LOTOS and remain valid here. In addition to further improvement to fully support the constraint oriented style, this work should be extended to define an adequate observation equivalence which abstracts away from internal actions *i* and preserves the classical LOTOS laws.

REFERENCES

[BoB 87] T. Bolognesi, E. Brinksma, *Introduction to the ISO Specification Language LOTOS,* Computer Networks and ISDN Systems 14 (1) 25-59 (1987).

[BoL 92] T. Bolognesi, F. Lucidi, *LOTOS-like process algebras with urgent or timed interactions,* in: K. Parker, G. Rose, eds., Formal Description Techniques, IV (North-Holland, Amsterdam, 1992) 249-264.

[Bri 89] E. Brinksma, *Constraint-oriented specification in a constructive formal description technique*, in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg, eds., Stepwise Refinement of Distributed Systems, Models Formalisms, Correctness, LNCS 430 (Springer-Verlag, Berlin, 1989) 130-152.

[CCE 93] J-P. Courtiat, M.S. deCamargo, D-E. Saidouni, *RT-LOTOS: LOTOS temporisé pour la spécification de systèmes temps réel*, in: R. Dssouli, G. v.Bochmann, L. Lévesque, eds., Ingénierie des Protocoles - CFIP'93 (Hermès, Paris, 1993).

[ISO 8807] ISO/IEC-JTC1/SC21/WG1/FDT/C, *IPS - OSI - LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807, February 1989.

[LeL 93a] G. Leduc, L. Léonard, *A timed LOTOS supporting a dense time domain and including new timed operators*, in: M. Diaz, R. Groz, eds., Formal Description Techniques, V (North-Holland, Amsterdam, 1993) 87-102.

[LeL 93b] G. Leduc, L. Léonard, *Comment rendre LOTOS apte à spécifier des systèmes temps réel?*, in: R. Dssouli, G. v.Bochmann, L. Lévesque, eds., Ingénierie des Protocoles - CFIP'93 (Hermès,Paris,93).

[LLD 93] L. Léonard, G. Leduc, A. Danthine, *The Tick-Tock case study for the assessment of Timed FDTs*, to appear in: A. Danthine, ed., The OSI95 Project (Springer-Verlag, Berlin, 1993).

[MFV 92] C. Miguel, A. Fernandez, L. Vidaller, *Extending LOTOS towards performance evaluation*, in: M. Diaz, R. Groz, eds., Formal Description Techniques, V (North-Holland, Amsterdam, 1993) 103-118.

[MFV 93] C. Miguel, A. Fernandez, L. Vidaller, *Assessment of Extended LOTOS*, to appear in: A. Danthine, ed., The OSI95 Project (Springer-Verlag, Berlin, 1993).

[MoT 90] F.Moller, C. Tofts, *A temporal calculus of communicating systems,* in: J.C.M. Baeten, J.W. Klop, eds., CONCUR '90, Theories of Concurrency: Unification and Extension, LNCS 458 (Springer - Verlag, Berlin Heidelberg New York, 1990) 401-415.

[NiS 91] X. Nicollin, J. Sifakis, *An Overview and Synthesis on Timed Process Algebras,* in: K.G. Larsen, A. Skou, eds., Computer-Aided Verification, III (LNCS 575, Springer-Verlag, Berlin Heidelberg New York, 1992) 376-398. Also in: LNCS 600.

[Wan 91] Y. Wang, *CCS + Time = an Interleaving Model for Real Time Systems,* in: J. Leach Albert, B. Mounier, M. Rodríguez Artalego, eds., Automata, Languages and Programming, 18 (LNCS 510, Springer-Verlag, Berlin Heidelberg New York, 1991) 217-228.