A Formal Definition of Time in LOTOS

Guy Leduc and Luc Léonard

Research Associate and Research Assistant of the
National Fund for Scientific Research (Belgium)
Université de Liège, Institut d'Electricité Montefiore, B 28, B-4000 Liège 1, Belgium
Tel: + 32 41 562691 Fax: + 32 41 562989 E-mail: leduc@montefiore.ulg.ac.be

Abstract

A time extended version of LOTOS, denoted ET-LOTOS, is proposed for the modelling of real-time behaviours. The language is first presented informally and applied to many small examples of sequential and concurrent systems. Then the formal semantics is given in two steps: on Basic ET-LOTOS (i.e. a simplified version of ET-LOTOS without data types) and then on full ET-LOTOS. Several equivalence relations are defined and many properties (equivalence laws, expansion theorems, …) are presented. The upward compatibility of ET-LOTOS w.r.t. LOTOS is also discussed. Finally, a larger example is used to demonstrate the usefulness of the language.

1. Introduction

The need to formally specify time-dependent systems is not new. Most protocols are based on time-out mechanisms that are essential for the safety of their behaviour. In 1976 Merlin and Farber [MeF 76] proposed a timed extension to Petri Nets that was precisely designed to specify recovery mechanisms based on time-outs.

Since then several new protocol mechanisms, as well as corresponding service facilities, have appeared and have strengthened this need. Isochronous data transfers, rate control, multimedia synchronization are some new examples.

In recent years, many quantitative timed extensions of well-known asynchronous process algebras have been proposed, as well as new timed process algebras. CSP [Hoa 85], CCS [Mil 89], ACP [BeK 85], LOTOS [ISO 8807, BoB 87] have been extended and new process algebras have been proposed: e.g. Timed CSP [ReR 88, DaS 89, Ree 90], extensions to CCS [MoT 90, HaJ 90, Wan 91, Han 91], extensions to ACP [Gro 90a, BaB 91], extensions to LOTOS [QuF 87, QAF 90, BLT 90, QFA 92, BoL 92a, BoL 92b, Led 92, MFV 93, MFO 93, LeL 93a, LeL 93b, CCE 93, BLT 93, LéL 94, BLT 94], ATP [NRS 90, NiS 91, NSY 91], TPL [HeR 90], PADS [Azc 90]. An overview and synthesis on Timed Process Algebras may be found in [NiS 92].

The time extended version of LOTOS that we propose in this paper for the modelling of quantitative timed behaviours has been inspired by some of the above-mentioned languages. It is an extended version of [LeL 93a, LeL 93b, LéL 94].

In this paper we justify ET-LOTOS from pragmatic and theoretical points of view. The first part is presented as a tutorial and the use of ET-LOTOS is shown on a collection of small but realistic and difficult examples. In a second part, we analyse the mathematical properties of ET-LOTOS. The formal semantics is presented, first for Basic ET-LOTOS, and then for Full ET-LOTOS. In the last part, we give the specification of a subset of the Tick-Tock case study [LLD 94].

2. Sequential Timed Behaviours

Let us consider the classical Time-out mechanism. In standard LOTOS, it is usually specified by expressions like `send; (ack; send_next [] i; re-send)`. In such a case the internal event `i` is intended to model the expiration of a time-out and thus model the recovery mechanism. However, there is no way to specify the exact time of occurrence of `i`.

ET-LOTOS is precisely defined to allow the specification of precise temporal requirements. For this purpose we first need to introduce our *time domain*, which is a set of timed values, denoted D, that satisfies certain properties.

Some authors have chosen a discrete time domain, i.e. a domain which is isomorphic to the natural numbers $N \cup \{\infty\}$ and equipped with basic operations like + and the order $\leq$. We consider that such a time domain is not easy to use in practice because it requires the selection of a reference grain of time in the specifications, which is not always possible. For example the specification of a system that has no known upper bound on the throughput it can generate, which will require arbitrary small delays between successive transmissions. Furthermore, when a refinement of the specification is envisaged, at a lower abstraction level with a smaller grain of time, this requires a complete change of the time values of the specification. For these reasons we decided to design ET-LOTOS in such a way that it can also support a dense time domain: neither its syntax, nor its semantics depend on the time domain. A time domain is *dense* if it is always possible to find a time value between any two given time values. Examples of dense time domains are domains that are isomorphic to the rational numbers or to the real numbers. To be able to give the operational semantics of ET-LOTOS in terms of Labelled Transition Systems (LTS), we will restrict ourselves to countable time domain, such as the rational numbers.

In fact, with ET-LOTOS, it is possible to define his/her own time domain provided that some constraints are fulfilled. The standard library can be extended with a data type of sort `time` and some usual operations. If D is the time domain, + the addition in D, 0 the neutral element of D, (D,+,0) will be a commutative monoid, (D,$\leq$) will be a total order and ∞ will be the absorbent element for + and such that $\forall\ d \in D.\ d \leq \infty$. A formal definition can be found in [MFV 93].

2.1. Two basic constructs: the life reducer and the delay operator

2.1.1. Time-out

This simple and classical example will allow us to illustrate two basic needs for extension: the maximum duration of an action *offer* and the prefixing by a delay. Indeed if we analyse the behaviour of a transmitter, we notice that after having transmitted a message, it is willing to wait for an acknowledgement *during a certain time* (concept of duration of an offer) or to retransmit the message *after this time*. In our formalism this description can be translated literally and leads to the process of figure 1. Note that, in all our examples, we use a simplified LOTOS syntax without the keywords “process”, “endproc”, and without process parameters when this does not create ambiguities.

```
SENDER := req?s:sdu; SEND(s)
SEND(s) := transmit!s; (ack{waiting-time}; SENDER
                        []
                        Δ^waiting-time SEND(s))
```

Figure 1. Time-out

Action-prefix is thus extended: the expression `g{time}`, where `time` ∈ D, means that `g` will not be offered after `time`, or in other words, can only occur in the interval `[0,time]`. This temporal attribute `{time}` is called the life reducer. The precise semantics of `g{time}` is as follows: if, after a delay `time`, `g` has not occurred, the process `g{time};...` behaves like `stop`, which means that the `g` offer is removed without executing the subsequent behaviour. However, this does not preclude other alternative behaviours, specified by way of the operator [], to be executed. Thus, the life reducer does not enforce the execution of `g` within the interval `[0,min]`. It states that `g` cannot occur outside this interval. When no life reducer is present, we are back in the LOTOS semantics where the action may occur at any time. In other words, the default value of the life reducer is ∞ for the observable actions. For internal actions, we will see later on that the default value will be 0.

The delay operator Δ^{time} expresses that the subsequent behaviour will be delayed by `time`.

Note that in ET-LOTOS no internal action is necessary to model the time-out in figure 1.

Our approach is not the only possible one. One could have generalized the action-prefix by introducing *two* attributes or a time interval such as `g{min,max}`. We did not retain this option for a simple reason: our combination is more flexible. Δ is not necessarily attached to an action, like the value `min` in `g{min,max}`, but can be applied to a whole process. However we will admit the shorthand notation `g{min,min+d}` for Δ^{min} `g{d}`.

If we come back to figure 1, we see that Δ prefixes the whole process `send`, without having to look at the internal structure of it.

Figure 2 depicts another example where `accept_data` and `accept_exp_data` are prefixed by different delays. This is a system that takes more time to accept a new ordinary data than to accept a new expedited data. This example also illustrates that the expiration of a delay does not resolve a choice. Indeed, if it was not the case, process `accept_data` could never be executed because the expiration of the small delay would resolve the alternative as a side effect. The expiration of a delay is thus not interpreted as an internal action.

```
SP := Δ^high-delay ACCEPT_DATA >> SP
      []
      Δ^small-delay ACCEPT_EXP_DATA >> SP
```

Figure 2. Δ does not resolve the choice

2.1.2. Isochronism

The pair (life reducer, Δ operator) is also very flexible to express an isochronous behaviour. Figure 3 depicts a process that accepts `DataReq` primitives at regularly spaced time instants.

```
ISOCHRONOUS1 := DataReq{0}; Δ^period ISOCHRONOUS1
```

Figure 3. Isochronism

Note that the offer `DataReq{0}` has a zero duration: it is punctual in time. The environment of this process has to be ready to interact at that precise instant for the action to occur. Otherwise the process will start behaving like `stop`. In other words, this process is not designed to be executed in a non co-operative environment. One can of course design a more robust process, as depicted on figure 4, which now has a recovery mechanism when `DataReq` does not occur: it will resynchronize on the next action after the delay `period`.

```
ISOCHRONOUS2 := DataReq{0}; Δ^period ISOCHRONOUS2
                []
                Δ^period ISOCHRONOUS2
```

Figure 4. Isochronism with fault tolerance

Note again that none of these processes enforce the occurrence of the `DataReq` primitive. However figure 3 shows a "confident" process that trusts on the environment, which is supposed to interact, whereas figure 4 shows a process that considers that an absence of interaction is not abnormal, since this case is explicitly described. If we want to design a communication service provider that only accepts `DataReq` primitives at precise instants, the second approach is better because the service user may not have data to send at each time slot. When shall we specify "confident" processes ? If the designer of the service provider is also the designer of the service user, (s)he may know that the user is always ready to send. This case is more plausible with a `DataInd` primitive for example. Another example may be more illuminating. Suppose that the interaction with the environment consists in emitting light or output some coffee, then the designer may trust on the environment: it will not stop light emission, and even the absence of a cup will not stop the coffee.

From these examples it appears that the designer has the flexibility to specify fault tolerance or not. A major point of disagreement between the proposed timed extensions is certainly the way this concept of "confidence" in the environment is expressed in the semantics. In [BLT 93] the authors consider that action necessity (i.e. the requirement that an observable action be executed within a certain time interval) is essential. This leads to a semantics with a mathematical artefact: time is blocked when the environment is not ready to interact with a process that offers an immediate necessary action. We are not in favour of this option. One may want to express some confidence in the environment, but this does not mean that one can constraint the environment to deserve this confidence. The blockage of time does not solve anything and turns out to be inconvenient in practice. For example, if the examples proposed in figures 1 and 4 are so simple, this is precisely because time is not blocked when the life reducers expire.

In reality, the sole criterion to be taken into account is much less constraining: it is necessary to *give precedence to the interaction* w.r.t. the progression of time without interaction. This essential requirement will be discussed later on when parallel composition is taken into account. Let us note already that in figures 3 and 4, this will lead to the following interpretation: if the environment and the process are ready to execute `DataReq`, then this action will necessarily occur immediately. Indeed, one could not admit that time would pass without this occurrence, which has become autonomous - internal - since the environment is taking part in it.

2.2. Nondeterministic delay - Internal Time Choice

The Δ operator allows prefixing by a well-defined delay. If we want to model a transmission medium that can introduce a propagation delay chosen internally and nondeterministically in an interval [min,max], Δ is not adequate because it can only impose the constraint on `min`, e.g. `in; Δ^min out{max-min}; …` In this example `out` can be executed after a delay `min` as soon as the environment is willing to. By contrast, what we want to express is that the medium alone (not the environment) be able to select the first time instant in [min,max] at which `out` can occur. This is an internal nondeterministic time choice.

```
TRANSM := DataReq?s:sdu; Δ^min i{max-min}; DataInd!s; stop
```

Figure 5. Nondeterministic delay

Consider figure 5. This process introduces, by way of the Δ^{min} operator, a minimum delay `min` between the request and the corresponding indication. Moreover it can introduce an additional delay chosen arbitrarily in the interval [0, max-min]. This is modelled by the internal event `i` equipped with the life reducer `{max-min}`. As for the observable actions, this life reducer limits the possible times of occurrence of i to the interval [0, max-min]. Moreover, since the i action is autonomous, it will occur in this interval at an arbitrary time instant which is *not* chosen by the environment but by the process itself. Let us insist on the difference between `i{delay};a;stop` and Δ^{delay} `a;stop`: In the first case, `a` may start to be offered at any time in [0, delay], whereas in the second case, `a` starts to be offered at 'delay'.

The fact that we are faced with a new form of nondeterminism justifies the occurrence of an `i` action. With Δ, the delay is determined and no `i` occurs.

There is a last property to mention about `i` which gives the desired effect in figure 5. It is about the *necessity* for i to occur in the interval associated with the life reducer. In other words, in `i{delay};P` the `i` action *shall necessarily* occur within [0, delay]. This was not true for observable actions whose occurrences are subject to the willingness of the environment. This concept of necessity or urgency associated with `i` is very interesting in practice. Besides the above example, we will illustrate its usefulness in three other cases.

When no life reducer is associated with `i`, we will consider that the default value is 0. One will say that such an `i` is urgent. By contrast, in `i{∞}`, `i` is said to be unconstrained. Remember that the default life reducer associated with an observable action is not 0, but ∞. There are two reasons for this difference. The main reason will be delayed until section 3.5. The second reason is that the ET-LOTOS specifications are likely to contain more `i{0}` than `i{∞}`.

The fact that a nondeterministic delay introduced via `i{d}` necessarily expires with the occurrence of an internal action may be uncomfortable in choice contexts. However, after an in-depth study, this can only be avoided in two cases: either if time is nondeterministic like in [Gro 90a], or if we add a special action in the semantics like in [LeL 93a]. We consider that these two approaches either lead to a less intuitive model, or to a more complex semantics. For these reasons we finally adopted the current proposal.

In [MFV 93] the introduction of a nondeterministic delay requires the use of the generalized choice. For example: `choice t:time [] [t isin interval] -> i; i[t]; P`. The first `i` that occurs immediately enforces the choice of a `t` value at time 0.

2.3. On the necessity to perform the internal action

In this section, we give three examples that show the advantage of having an internal action that can be required to necessarily occur.

2.3.1. The double watchdog

The first example is the watchdog presented in [NiS 91]. To model a watchdog, X. Nicollin and J. Sifakis introduce a new operator in ATP. We will show that such an operator is not needed in ET-LOTOS. Consider figure 6. It is a login procedure. Two `prompt` actions asks successively the user to input a name and a password within less than `logtime` time units. When this time has elapsed or if the password is incorrect, the system restarts, otherwise it accepts the request. The user can thus have several attempts to log in. However there is a maximum delay of `maxtime` to successfully log in. It is clear that there are two embedded watchdogs parametrized with `logtime` and `maxtime`.

```
SESSION :=
(LOGIN_PROCEDURE [> Δ^maxtime i; stop) >> SESSION_PHASE
where
LOGIN_PROCEDURE :=
PHASE1 >> accept correct_login:bool in
          ([correct_login]-> exit) [] ([not(correct_login)]-> LOGIN_PROCEDURE)
where
PHASE1 :=
prompt?name:usr_name; prompt?pwd:password; exit(correct(name,pwd))
[> Δ^logtime i; PHASE1
```

Figure 6. Login procedure

The reason why a specific watchdog operator is not needed in ET-LOTOS are: LOTOS already possess the disabling operator [> which, coupled with a delay and the possibility to enforce the occurrence of `i`, has the same expressive power.

Note that in ET-LOTOS the process `exit` can also have an associated life reducer, which is by default equal to ∞ (as in figure 6). This parameter ensures that successful termination shall occur soon enough or never. We simply consider the successful termination action δ as any other observable action.

2.3.2. Time-out

We propose an alternative solution to the time-out proposed in section 2.1 but in a more general case. Consider figure 7 which depicts the time-out of figure 1 with the difference that we suppose the existence of a process `Receive_ack` that we want to use as is. This process represents the normal behaviour, whereas the process `send` represents the error recovery process.

```
SENDER := req?s:sdu; SEND(s)
SEND(s):= send!s; (RECEIVE_ACK  []  Δ^waiting-time i; SEND(s))
```

Figure 7. Time-out - Variant

Here the presence of `i` implies that the system shall necessarily resolve the choice directly after the expiration of the delay $\Delta^{waiting\_time}$.

2.3.3. Throughput control

The third example is a service facility extracted from the OSI95 transport service [BLL 94]. It is the part of the service provider that controls the throughput at which data are accepted by the provider to fulfil the (negotiated) quality of service (QoS). In a first step, we consider three criteria:

- The provider does not allow the user to transmit data above a certain throughput, by spacing the `DataReq` primitives by a least `min` time units.
- The provider need not be ready to accept a `DataReq` immediately after `min` units.
- The provider should however offer a minimum data throughput to the user. If, at any time, the provider cannot offer this minimum throughput, it shall disconnect. This will be specified by a maximum delay `max` between two `DataReq` offers. Note however that if the `DataReq` primitive does not occur because the user has nothing to transmit, the provider shall not disconnect. The provider only disconnects if it is responsible for not fulfilling the QoS requirement.

Figure 8 depicts the ET-LOTOS description. We notice the presence of two delay operators Δ^{min} and Δ^{max}, which do not resolve the choice when they expire. We also notice two internal actions:

the first one being unconstrained, and the second one urgent. They are necessary to model respectively the second and the third criteria.

```
THROUGHPUT-CONTROL1 := Δ^min i{∞}; DataReq; THROUGHPUT-CONTROL1
                       []
                       Δ^max i; DisInd; stop
```

Figure 8. Throughput control with disconnection

In OSI95 there is a fourth criterion:

- The provider shall indicate the user that it cannot (temporarily) sustain a threshold throughput `Thres` negotiated somewhere between the minimum and maximum values.

Figure 9 depicts the solution.

```
THROUGHPUT-CONTROL2 :=
Δ^thres ReportInd; stop
[> (Δ^min i{∞}; DataReq; THROUGHPUT-CONTROL2
    []
    Δ^max i; DisInd; stop)
```

Figure 9. Throughput control with indication and disconnection

2.4. Time measurement

It is sometimes useful to measure the time that elapses between an action offer and the corresponding action occurrence. To realise that we propose to enhance the action prefix with a special attribute `@t`, "at t", that declares a time variable which will precisely store this waiting time. This solution has been proposed by Wang Yi [Wang 91] to allow his Timed CCS to have an expansion theorem in presence of dense time. We could motivate its introduction in ET-LOTOS similarly, but other practical reasons will be introduced instead.

2.4.1. Isochronism

We will consider a variant of the example presented in section 2.1.2. Here we replace the punctual offer by an offer of maximum duration `d` (`d < period`). Moreover we will count the successful offers and the unsuccessful offers to allow for an adaptation of the period. Figure 10 shows the use of the attribute `@t`. The new period is calculated by the function `new` which is not detailed.

```
behaviour ISOCHRONOUS(0,0,period,0) where
ISOCHRONOUS(ok,nok,period,t1):=
Δ^period-t1 (a@t{d}; ISOCHRONOUS (ok+1,nok,new(period,ok,nok),t)
            []
            ISOCHRONOUS (ok, nok+1, new(period,ok,nok),0))
```

Figure 10. Isochronism - Variant

The attribute `@t` is a particular form of variable declaration whose scope is defined like for any classical variable declaration in LOTOS.

2.4.2. Measurement of time between failures

We model a system that can crash at any time and restarts only if the time elapsed since the last crash is not below a lower bound `min`. Figure 11 depicts this example.

```
P:= Normal_behaviour [> (i@t; [t > min] -> P)
```

Figure 11. Measurement of time between failures

2.4.3. Spacing non consecutive actions

This example is proposed in [BLT 93] and stated as follows: "in a sequence of four consecutive actions `a`, `b`, `c` and `d`, the last action should occur a fixed amount of time, `delay`, after the first one has occurred."

According to us, the term 'should' is misleading because there exist cases in which `d` may not occur at all: namely when the environment does not offer the sequence abcd. Therefore we consider that the sentence should be at least rephrased as follows: "In a sequence of four consecutive actions `a`, `b`, `c` and `d`, the last action shall occur a fixed amount of time, `delay`, after the first one has occurred, **or shall not occur at all**." This sentence can however be interpreted in two ways:
• `b` and `c` themselves cannot occur after this `delay` has elapsed
• `b` and `c` may possibly occur after this `delay` has elapsed, but then `d` will not occur at all.

In the first interpretation we propose two solutions (Figure 12). A third constraint-oriented one is presented in section 3.4.

```
P1 := a; b@t1{delay}; c@t2{delay-t1}; d@t3 [t1+t2+t3 = delay]; stop

P2 := a; b@t1{delay}; c@t2{delay-t1}; Δ^delay-t1-t2 d{0}; stop
```

Figure 12. Spacing non consecutive actions - first interpretation

In the second interpretation we propose two solutions (Figure 13). A third constraint-oriented one is presented in section 3.4.

```
P1 := a; b@t1; c@t2; d@t3 [t1+t2+t3 = delay]; stop

P2 := a; b@t1; c@t2; Δ^delay-t1-t2 d{0}; stop
```

Figure 13. Spacing non consecutive actions - second interpretation

Note that in both `P1` solutions, the timed variables `ti` (i = 1..3) are used in a selection predicate.

2.4.4. Beat detection

This example is proposed in [BLT 93] and stated as follows: "in a sequence of `a` actions, the unconstrained spacing between the first two actions is detected and imposed as the spacing for the rest of the sequence."

The problem with this description is in the chosen term "imposed". For this statement to have a sense, one has to admit that, after the first two `a` actions, the environment is always ready to perform action `a` when our process decides to perform it. Then the solution follows:

```
P := a; a@t; BEAT (t)
     where BEAT (t:time) := Δ^t a{0}; BEAT (t)
```

Figure 14. Beat detection

2.4.5. Global timer for keeping the absolute time

This example is proposed in [BLT 93] and stated as follows: "The process `Timer` keeps the absolute value of time, i.e. the amount of time passed since system startup, and is always ready to offer this value to other processes wishing to read it".

The solution makes use of a selection predicate referring to a timed variable:

```
TIMER[g] (t:time) := g?t1:time@t2 [t1=t+t2]; TIMER[g] (t1)
```

Figure 15. Global timer for keeping the absolute time

A process that wants to use the timer shall use the action `g?t:time`. Moreover, `g` will surely be an internal gate of the system and thus autonomous.

2.5. Conclusion on sequential processes

Up to now all our examples do not involve the synchronization between processes. The parallel composition will be discussed in section 3. Let us briefly summarize the simple timed extensions that we have proposed:

- A delay operator Δ that prevents the process to execute actions for a while. Its semantics is such that no internal action occurs when the delay expires.
- An optional life reducer associated with action prefix. There is no requirement to actually execute the action within the specified interval except if the action is `i`.
- An optional timed variable declaration associated with action prefix. It allows the measurement of the time spent between action offer and action occurrence.

We have shown that the separation between the delay and the life reducer is very flexible.

3. Extension to parallel processes

In this section we study the description of more complex systems composed of several communicating processes. We will see from several examples that in practice it appears quite sufficient to impose *urgency on internal synchronizations* between processes to specify real-time distributed systems.

3.1. Symmetric Time-out

Our first example is the symmetric time-out proposed by T. Bolognesi in [BLT 93]. The system is composed of two processes that behave as follows:

- They first execute some private tasks of unknown duration (represented by actions `d1` and `d2`)
- Then they propose to synchronize with the other one (via gate `sync`) as soon as possible (i.e. as soon as the other one is ready)
- But they are not willing to wait for synchronization more than a certain delay (resp. `t01` and `t02`)

This example is presented as a difficult problem because the communicating processes cannot determine when the other one becomes ready to interact. In particular, the other process can be ready before. It thus becomes impossible for a process to impose the time of occurrence of the interaction. Therefore, the urgency requirement on the interaction can only be solved at a more global level in which the two processes are seen as a pair of composed processes.

To realize that, we have adopted the *maximal progress* hypothesis, but limited to the *internal interactions*. This means that the semantics of the `hide` operator is such that every hidden action becomes urgent. In other words, if a synchronization can occur, it shall not be postponed: it shall occur now unless another competing action occurs. We can summarize the discussion by saying that hidden actions have a higher priority than passage of time (without occurrence of the hidden action), but not a higher priority than other actions.

This form of maximal progress on the hidden action `sync` allows us to write a very simple ET-LOTOS specifications of the symmetric time-out (figure 16). Note also the advantage of having a life reducer in this case.

```
SYMMETRIC_TIMEOUT [d1,d2] (to1,to2:time) :=
hide sync in (PROCESS[d1,sync](to1) |[sync]| PROCESS[d2,sync](to2))
where
PROCESS [d,sync] (t:time) := d; sync{t}; PROCESS [d,sync](t)
```

Figure 16. The symmetric time-out

There is an alternative to this maximal progress approach. It has been proposed in [BLT 93] and consists in adding a specific operator that can urge observable actions.

3.2. Synchronization between a sender and a receiver

Figure 17 gives another less abstract specification of the time-out of figure 1. Here we isolate a `Timer` process that implements the time constraints and interacts with the `Sender` via three interaction points: `set` (to start the timer), `reset` (to stop the timer) and `timeout` (to indicate that time has elapsed).

We can see that the three interactions `set`, `reset` and `timeout` are internal. Moreover, the time-out behaves correctly because the interactions between `Sender` and `Timer` occur as soon as they are made possible by both processes. Stated otherwise, internal interactions are urgent.

```
SYSTEM := hide set,reset,timeout in (SENDER
                                     |[set,reset,timeout]|
                                     TIMER)
where
SENDER := req?s:sdu; SENDER2 (s)
SENDER2 (s:sdu) := send!s; set!waiting-time; (ack; reset; SENDER
                                              []
                                              timeout; SENDER2 (s))
TIMER := set?t:time; (reset; TIMER
                      []
                      Δ^t timeout; TIMER)
```

Figure 17. A less abstract time-out

There are other formalisms that do not have maximal progress, nor an extra urgency operator (e.g. TIC [QFA 92], ATP [NRS 90]), but can nonetheless specify correctly this example. This is due to the fact that, for every interaction, one knows in advance the process that will be the first to offer the interaction: the `Sender` for a `set` or a `reset`, the `Timer` for a `timeout`. One can then impose urgency locally in one of the processes for every interaction.

3.3. Urgent termination

The maximal progress on hidden actions extends naturally to synchronizations on the successful termination action δ. To illustrate the benefits induced by this characteristic, we specify an example

inspired by the Lip-Synchronization algorithm from [Reg 93]. The specification is given in figure 18. It is composed of two parallel processes, each of them being responsible for a data stream (either voice or video). Each process is supposed to receive a quasi-isochronous stream (i.e. with some jitter) and retransmit the stream in an isochronous way. To achieve that, the processes introduce an additional transit delay equal to the maximum expected jitter. However if the jitter in the input stream is higher than expected such that a data does not arrive soon enough, then the process will not be able to resynchronize and must signal the error.

The fact that synchronization on a hidden δ is urgent allows us to model urgent termination and give a simple specification of this system (figure 18). Each `Control` process resynchronizes its input stream until it is unable to do so due to the too late arrival of the next frame. In this case, it offers to terminate with an error code (`exit (ers)`). This termination will interrupt the process that handles the other stream thanks to the `[> exit` construct. By symmetry this structure allows a simultaneous and immediate stop initiated by any of the two processes. Note that the two processes `exit (video)` and `exit (sound)` cannot synchronize because `video` ≠ `sound`.

This example is similar to the symmetric time-out (section 3.1), because we cannot anticipate which process will initiate the disconnection.

```
SYNC_CONTROL [sound_in,video_in,sound_out,video_out,error]:=
  ((CONTROL [sound_in,sound_out] (sound,speriod,sjitter) [> exit(video))
   |||
   (CONTROL [video_in,video_out] (video,vperiod,vjitter) [> exit(sound))
  ) >> accept err:ersource in error!err; stop
where
  CONTROL [in,out] (ers:ersource, per,jit:time):=
       in@t{jit}; Δ^jit-t out{0}; Δ^per-jit CONTROL [in,out] (ers,per,jit)
       []
       Δ^jit exit(ers)
```

Figure 18. Resynchronization of multimedia streams

3.4. Spacing non consecutive actions

The problem stated in section 2.4.3 can also be solved in a constraint-oriented style as shown in figure 19. The two processes `P3` correspond to the two interpretations explained in 2.4.3.

```
P3 := a; b; c; d; stop
      ||
      a; (b{delay}; stop
          |||
          c{delay}; stop
          |||
          Δ^delay d{0}; stop)

P3 := a; b; c; d; stop
      |[a,d]|
      a; Δ^delay d{0}; stop
```

Figure 19. Spacing non consecutive actions - CO style descriptions

3.5. Conclusions on parallelism

A single concept has been introduced in this section: urgency on explicitly hidden actions. Let us note that this is a limited version of the maximal progress property adopted by [HaJ 90, HeR 90, MFV 93, ReR 88, Reg 93, Wan 91].

We can now better justify our choice of the default value 0 for the life reducer associated with `i`, which contrasts with the default value (∞) associated with the life reducers of observable actions. The reason is that we wanted to be consistent with the fact that hiding actions necessarily induces urgency. More precisely, if we want to preserve strong equivalence between processes like `i; exit` and `hide a in (a;exit |[a]| a;exit)`, it is important that the `i`'s without life reducer have in fact a life reducer equal to 0.

As mentioned earlier some formalisms are less expressive than ours because they can only impose urgency locally (i.e. when a process decides the occurrence time for all parallel processes) [NRS 90, QFA 92].

4. Formal semantics and properties of ET-LOTOS

In this section we present the formal operational semantics of ET-LOTOS. It will be given by a set of axioms and inference rules for every operator. We will adopt the presentation in two columns like in [MoT 90] in order to clearly separate the timed transitions from the standard actions.

4.1. Syntax and semantics of Basic Timed LOTOS

Notations

G denotes the countable set of observable gates. $L = G \cup \{\delta\}$ denotes the alphabet of observable actions where δ is the special action denoting successful termination (δ does not belong to G). $A = L \cup \{i\}$ is the alphabet of actions where the symbol i is reserved for the unobservable internal action (i does not belong to L). Capital Greek letters such as Γ will be used to denote subsets of G. D denotes the countable time domain which is the alphabet of timed actions. $D_0 = D - \{0\}$.

The collection of Basic ET-LOTOS behaviour expressions is defined by the following BNF expression where $a \in G \cup \{i\}$, d and $t \in D$, $\Gamma \subseteq G$ and $\tilde{X}$ represents a vector of process names X:

P ::= Q where $\tilde{X} := \tilde{Q}$ [1]

Q ::= stop | exit{d} | a@t{d};Q | Δ^d Q | Q[]Q | Q|[Γ]|Q | hide Γ in Q | Q>>Q | Q[>Q | X

Remark: in a@t{d};Q we let both @t and {d} be optional, and use the convention that, if omitted, d=∞ when $a \neq i$ and d=0 when a=i. Similarly {d} is optional in exit{d}, and exit means implicitly exit{∞}.

$P \xrightarrow{a} P'$, with $a \in A$, means that process P may engage in action a and, after doing so, behave like process P'. $P \xrightarrow{a}$ means $\exists\, P'.\ P \xrightarrow{a} P'$. $P \not\xrightarrow{a}$, with $a \in A$, means $\nexists\, P'.\ P \xrightarrow{a} P'$, i.e. P cannot perform action a. $P \xrightarrow{d} P'$, with $d \in D$, means that process P may idle (i.e. not execute any action in A) during a period of d units of time and, after doing so, behave like process P'. $P \not\xrightarrow{d}$, with $d \in D$, means that $\nexists\, P'.\ P \xrightarrow{d} P'$, i.e. P cannot idle during a period of d units of time.

In the following inference rules, $d \in D_0$, $d_1 \in D$ et $a \in A$ by default. Of course, according to the syntax defined above, we have also implicitly $a \neq \delta$ in AP1, AP2, AP3, TM1, TM2, TM3.

[1] For convenience, we suppose, without lack of generality, that there is a single where-clause that gathers all the process declarations of the specification.

	(S) $\mathrm{stop} \xrightarrow{d} \mathrm{stop}$
(AP1) $a\{d_1\}; P \xrightarrow{a} P$	(AP2) $a\{d_1+d\}; P \xrightarrow{d} a\{d_1\}; P$ (AP3) $a\{d_1\}; P \xrightarrow{d} \mathrm{stop}$ $(a \neq i, d > d_1)$
(TM1) $a@t\{d_1\}; P \xrightarrow{a} [0/t]\,P$	(TM2) $a@t\{d_1+d\}; P \xrightarrow{d} a@t\{d_1\}; [t+d/t]\,P$ (TM3) $a@t\{d_1\}; P \xrightarrow{d} \mathrm{stop}$ $(a \neq i, d > d_1)$
(D1) $\dfrac{P \xrightarrow{a} P'}{\Delta^0 P \xrightarrow{a} P'}$	(D2) $\Delta^{d_1+d} P \xrightarrow{d} \Delta^{d_1} P$ (D3) $\dfrac{P \xrightarrow{d} P'}{\Delta^0 P \xrightarrow{d} P'}$
(Ex1) $\mathrm{exit}\{d_1\} \xrightarrow{\delta} \mathrm{stop}$	(Ex2) $\mathrm{exit}\{d_1+d\} \xrightarrow{d} \mathrm{exit}\{d_1\}$ (Ex3) $\mathrm{exit}\{d_1\} \xrightarrow{d} \mathrm{stop}$ $(d > d_1)$
(Ch1) $\dfrac{P \xrightarrow{a} P'}{P \;[]\; Q \xrightarrow{a} P'}$ (+ Ch1')	(Ch2) $\dfrac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q'}{P \;[]\; Q \xrightarrow{d} P' \;[]\; Q'}$
(PC1) $\dfrac{P \xrightarrow{a} P'}{P \;\|[\Gamma]\|\; Q \xrightarrow{a} P' \;\|[\Gamma]\|\; Q}$ $(a \notin \Gamma \cup \{\delta\})$ (+ PC1') (PC2) $\dfrac{P \xrightarrow{a} P',\ Q \xrightarrow{a} Q'}{P \;\|[\Gamma]\|\; Q \xrightarrow{a} P' \;\|[\Gamma]\|\; Q'}$ $(a \in \Gamma \cup \{\delta\})$	(PC3) $\dfrac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q'}{P \;\|[\Gamma]\|\; Q \xrightarrow{d} P' \;\|[\Gamma]\|\; Q'}$
(H1) $\dfrac{P \xrightarrow{a} P'}{\mathrm{hide}\ \Gamma\ \mathrm{in}\ P \xrightarrow{a} \mathrm{hide}\ \Gamma\ \mathrm{in}\ P'}$ $(a \notin \Gamma)$ (H2) $\dfrac{P \xrightarrow{a} P'}{\mathrm{hide}\ \Gamma\ \mathrm{in}\ P \xrightarrow{i} \mathrm{hide}\ \Gamma\ \mathrm{in}\ P'}$ $(a \in \Gamma)$	(H3) $\dfrac{P \xrightarrow{d} P',\ P \xrightarrow{a} \forall\ a \in \Gamma}{\mathrm{hide}\ \Gamma\ \mathrm{in}\ P \xrightarrow{d} \mathrm{hide}\ \Gamma\ \mathrm{in}\ P'}$
(En1) $\dfrac{P \xrightarrow{a} P'}{P >> Q \xrightarrow{a} P' >> Q}$ $(a \neq \delta)$ (En2) $\dfrac{P \xrightarrow{\delta} P'}{P >> Q \xrightarrow{i} Q}$	(En3) $\dfrac{P \xrightarrow{d} P',\ P \xrightarrow{\delta} }{P >> Q \xrightarrow{d} P' >> Q}$
(Di1) $\dfrac{P \xrightarrow{a} P'}{P \;[>\; Q \xrightarrow{a} P' \;[>\; Q}$ $(a \neq \delta)$ (Di2) $\dfrac{Q \xrightarrow{a} Q'}{P \;[>\; Q \xrightarrow{a} Q'}$ (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) $\dfrac{[g_1/h_1,\ldots g_n/h_n]\ P \xrightarrow{a} P',\ Q[h_1,\ldots h_n]:=P}{Q[g_1,\ldots g_n] \xrightarrow{a} P'}$	(In2) $\dfrac{[g_1/h_1,\ldots g_n/h_n]\ P \xrightarrow{d} P',\ Q[h_1,\ldots h_n]:=P}{Q[g_1,\ldots g_n] \xrightarrow{d} P'}$

Table 1 Operational semantics of Basic ET-LOTOS

We will not detail all these rules. Let us note that the left column basically contains the LOTOS inference rules (except of course TM1 and D1 which deal with timed constructs). TM1 expresses that the variable t is instantiated by 0 when the action occurs. D1 indicates that a zero delay has no effect on executable actions. Concerning the second column, S means that `stop` is a process that lets time pass. Ch2, PC3 et Di4 impose that time passes at the same pace in all processes. AP2 indicates

that a life reducer decreases when time passes without reaching the time limit. AP3 indicates that time may pass beyond the life reducer limit provided that the associated action offer disappear (as well as the subsequent behaviour). TM2 and TM3 are similar to AP2 and AP3. In these rules is explained how the `t` variable declared in the attribute `@t` is instantiated. D2 indicates that prefixing by a delay allows the passing of time until the delay elapses (i.e. equals 0). D3 completes D1 to indicate that a zero delay has no effect on a process. Ex2 and Ex3 are the transpositions of AP2 and AP3 to the δ action. H3 expresses the maximal progress property on the hidden actions by giving priority to a hidden action w.r.t. the passing of time. En3 is similar to H3 because >> implicitly hides the δ action, which makes the successful terminations urgent.

An additional shorthand notation

We also introduce the notations $a\{d_1,d_2\};P$ to mean $\Delta^{d1}a\{d_2\text{-}d_1\};P$ and $a@t\{d_1,d_2\};P$ for $\Delta^{d1}a@t\{d_2\text{-}d_1\};[t+d_1/t]P$. The last definition means that $@t$ starts to count when the control arrives at $a@t\ \{d_1,d_2\};P$, and not when a actually begins to be offered. We made this choice because we think it is more intuitive, and because it complements the basic construction $\Delta^{d1}a@t\{d_2\text{-}d_1\};P$, where t does not include the delay d_1.

Let us outline some interesting features of the semantic rules defined above:
- The LOTOS rules are kept unchanged (in left column)
- The alphabet A of actions is kept as is (e.g. no additional time stamps in action labels). We have just extended it with timed actions from a separate set D.
- We do not need auxiliary operators or functions. This last point will however not remain true in Full ET-LOTOS due to the generalized choice operator.

Blockage of time

In this semantic model, unguarded specifications[1] block the progression of time. Consider for example the specification “S := S”. No timed transition can be derived according to the operational semantics.

Usually unguarded specifications are not really useful. A counter-example is however “Ps where Ps := P ||| Ps”.

To refer explicitly to processes that block time, we define the pathological process, *block*, which has no axiom and no inference rule.

4.2. Syntax and semantics of Full Timed LOTOS

In full ET-LOTOS, the data types are described in the Abstract Data Type language ACT ONE. Some new operators such as guards are also added.

The collection of Full ET-LOTOS behaviour expressions is defined by the following BNF expression where $a \in G$, d and $t \in D$, $\Gamma \subseteq G$, $\tilde{X}$ represents a vector of process names, SP is a selection predicate (a Boolean expression or an equation) and the x_i’s (resp. tx_i’s) are variables (resp. terms) of sorts s_i’s :

[1] “P where $X_1 := P_1, \ldots X_n := P_n$” is a guarded ET-LOTOS specification if, by recursively substituting a finite number of times the expressions P_i’s for the process identifiers X_i’s occurring in P and in the P_i’s themselves, it is possible to obtain an expanded ET-LOTOS specification “Q where $X_1 := Q_1, \ldots X_n := Q_n$” where Q and the Q_i’s are guarded expressions, i.e. if all instantiations of Q_i’s in X_j’s are preceded by at least an action (observable or not) or a (non-zero) delay.

P ::= Q where $\tilde{X}$:= $\tilde{Q}$

Q ::= stop | exit{d} | a@t{d}[SP];Q | i@t{d};Q | Δ^d Q | Q[]Q | Q|[Γ]|Q | hide Γ in Q | Q>>Q
| Q[>Q | X | [SP] -> Q | let x_1=tx_1, … x_n=tx_n in Q | choice x_1:s_1, … x_n:s_n [] Q

The mapping function between a full ET-LOTOS specification and a (structured) Labelled Transition System (LTS) is rather complex. It involves several phases that we recall hereafter.

First phase : The flattening mapping

The purpose of the *flattening mapping* is to produce a *canonical LOTOS specification*, *CLS* for short, where all identifiers are unique and defined at one global level. This function is partial since only static semantically correct specifications have a well-defined canonical form CLS.

CLS is a 2-tuple <CAS, CBS> composed of:

(i) a *canonical behaviour specification CBS*, i.e. a set of process definitions *PDEFS* with an initial process definition $pdef_0 \in PDEFS : CBS = <PDEFS, pdef_0>$.

A *process definition* is a pair consisting of a process variable *p* and a behaviour expression *B* : *pdef* = <*p*, *B*>.

(ii) a *canonical algebraic specification CAS*, i.e. an algebraic specification <*S, OP, E*> (*S* is a set of sorts, *OP* is a set of operations and *E* is the set of conditional equations defined on the signature <*S, OP*>) such that the signature <*S, OP*> contains all sorts and operations occurring in *CBS*.

Second phase : The derivation system of a data representation and the interpretation of CAS

This phase consists in generating a *derivation system*, denoted *DS*, from the data representation *CAS* = <*S, OP, E*>. This derivation system is composed of axioms and inference rules generated by the conditional equations of *E*.

A congruence relation between ground terms (terms which do not contain variables) is induced by *CAS* : two ground terms t_1 and t_2 are called *congruent* w.r.t. *CAS*, simply denoted $t_1 = t_2$, iff
DS $\vdash t_1 = t_2$, i.e. it is possible to prove $t_1 = t_2$ from the axioms and the inference rules of the derivation system *DS*.

[t] denotes the set of all ground terms congruent to *t* w.r.t. *CAS*, i.e. intuitively *[t]* is the object represented by *t* or any of its equivalent representations.

The semantic interpretation of *CAS* = <*S, OP, E*> is the many-sorted algebra $Q(CAS) = <D_Q, O_Q>$, called the *quotient term algebra*, where

(i) D_Q is the set $\{Q(s) \mid s \in S\}$,
where $Q(s) = \{[t] \mid t$ is a ground term of sort $s\}$ for each $s \in S$; and

(ii) O_Q is the set of functions $\{Q(op) \mid op \in OP\}$,
where the $Q(op)$ are defined by $Q(op)([t_1], \ldots [t_n]) = [op(t_1, \ldots t_n)]$.

In this algebra, the terms with different representations but modelling the same object are collapsed.

Third phase : Mapping of CLS onto a LTS

The purpose of this last phase is the generation of a LTS. This generation is based on a transition derivation system .

The *transition derivation system* of a canonical ET-LOTOS specification CLS = <*CAS,CBS*> is composed of axioms and inference rules like those provided hereafter.

If P_1 and P_2 are two behaviour expressions, $P_1 \xrightarrow{av_1 \ldots v_n} P_2$ means intuitively that P_1 may accept the composite event $av_1 \ldots v_n$ and further behave like P_2. In these expressions, it is required that P_1 and P_2 be closed, i.e. they do not contain free variables.

Table 1 gives the transition derivation system of Basic ET-LOTOS. We will simply generalize some rules and give the rules for operators that were not part of Basic ET-LOTOS.

4.2.1. Action-prefix with attributes and selection predicates

The rules for action-prefix are basically the same as TM1, TM2, TM3 (or their version without the attribute @t given as AP1, AP2, AP3). Like in full LOTOS, a selection predicate can be added to the action-prefix when the name of the action is not i.

The rules for i are thus unchanged (refer to AP1, AP2, TM1 and TM2 in table 1). We just give hereafter the extended rules TM1, TM2 and TM3 which are valid when **name (a) ≠ i**. The rules AP1, AP2 and AP3 are extended similarly.

(TM1) $ad_1 \ldots d_n@t\{d\}[SP(y_1,\ldots y_m,t)];P \xrightarrow{av_1 \ldots v_n} [ty_1/y_1,\ldots ty_m/y_m,0/t]P$ if $DS \mid\!- SP(ty_1,\ldots ty_m,0)$

where $v_i = [t_i]$ if $d_i = !t_i$

$v_i \in Q(s_i) = \{[t] \mid t \text{ is a ground term of sort } s_i\}$ if $d_i = ?x_i : s_i$

$\{y_1,\ldots y_m\} = \{x_i \mid d_i = ?x_i : s_i\}$

$[ty_j] = v_i$ if $y_j = x_i$ and $d_i = ?x_i : s_i$

(TM2) $ad_1 \ldots d_n@t\{d'+d\}[SP(y_1,\ldots y_m,t)];P \xrightarrow{d} ad_1 \ldots d_n@t\{d'\}[SP(y_1,\ldots y_m,t+d)]; [t+d/t]\ P$

(TM3) $ad_1 \ldots d_n@t\{d'\}[SP(y_1,\ldots y_m,t)];P \xrightarrow{d} stop \quad (d > d')$

Table 2 Operational semantics of full action-prefix

In full ET-LOTOS the shorthand notation *a@t{d1,d2}[SP]; P* means $\Delta^{d1}a@t\{d_2\text{-}d_1\}[[t+d_1/t]SP]; [t+d_1/t]P$.

4.2.2. Guard and let

These operators are defined without any problem (Table 3).

(G1) $\dfrac{P \xrightarrow{a} P'}{[SP] \text{ -> } P \xrightarrow{a} P'}$ if $DS \mid\!- SP$

(G2) $\dfrac{P \xrightarrow{d} P'}{[SP] \text{ -> } P \xrightarrow{d} P'}$ if $DS \mid\!- SP$

(G3) $[SP] \text{ -> } P \xrightarrow{d} stop$ if $\neg\ DS \mid\!- SP$

(L1) $\dfrac{[tx_1/x_1, \ \ldots \ tx_n/x_n]P \xrightarrow{a} P'}{\text{let } x_1=tx_1, \ \ldots \ x_n=tx_n \text{ in } P \xrightarrow{a} P'}$

(L2) $\dfrac{[tx_1/x_1, \ \ldots \ tx_n/x_n]P \xrightarrow{d} P'}{\text{let } x_1=tx_1, \ \ldots \ x_n=tx_n \text{ in } P \xrightarrow{d} P'}$

Table 3 Operational semantics of the guard and let operators

4.2.3. Generalized choice

The generalized choice creates special problems in ET-LOTOS like in any other timed extension of full LOTOS. The nature of this problem lies in the possibly infinite "synchronization" on timed actions between all the branches of the choice operator [BLT 93].

The first rule is the classical rule GC1 (table 4) and a first definition of the second rule is GC2 (table 4). The auxiliary operator 'Age' takes as arguments a behaviour expression P and a time value d, and returns the behaviour expression P' such that $P \xrightarrow{d1} \xrightarrow{d2} \ldots \xrightarrow{dn} P'$, where $d = \sum_{i=1}^{n} d_i$, or *block* if no P' exists. The formal semantics of Age is given below.

$$\text{(GC1)}\quad \frac{[tx_1/x_1,\ \ldots\ tx_n/x_n] \xrightarrow{a} P\ \ P'}{\text{choice}\ \ x_1{:}s_1,\ \ldots\ x_n{:}s_n\ [] \xrightarrow{a}\ P'}$$

where tx_i are ground terms with $[tx_i] \in Q(s_i)$

$$\text{(GC2)}\quad \frac{[tx_1/x_1,\ \ldots\ tx_n/x_n] \xrightarrow{d} P \qquad \forall\ tx_i.\ [tx_i] \in Q(s_i),\ i =}{\text{choice}\ \ x_1{:}s_1,\ \ldots\ x_n{:}s_n\ [] \xrightarrow{d}\ \text{choice}\ \ x_1{:}s_1,\ \ldots\ x_n{:}s_n\ []\ \text{Age}\ (c$$

Table 4 Operational semantics of the generalized choice operator

The auxiliary Age operator is defined inductively on the syntax of ET-LOTOS behaviour expressions as follows.

Age (d, stop) = stop
Age (d, i@t{d'}; P) = i@t{d'-d}; [t+d/t] P if d ≤ d'
= block otherwise
Age (d, $ad_1, \ldots d_n$@t{d'} [SP]; P) = $ad_1, \ldots d_n$@t{d'-d} [[t+d/t]SP]; [t+d/t] P if d ≤ d'
= stop otherwise
Age (d, exit ($d_1, \ldots d_n$){d'}) = exit ($d_1, \ldots d_n$){d'-d} if d ≤ d'
= stop otherwise
Age (d, $\Delta^{d'}$ P) = $\Delta^{d'-d}$ P if d ≤ d'
= Age (d-d', P) otherwise
Age (d, P [] Q) = Age (d, P) [] Age (d, Q)
Age (d, P |[Γ]| Q) = Age (d, P) |[Γ]| Age (d, Q)
Age (d, P [> Q) = Age (d, P) [> Age (d, Q)
Age (d, P >> Q) = Age (d, P) >> Q if δ ∉ Out (P) (Out is defined below)
= block otherwise
Age (d, hide Γ in P) = hide Γ in Age (d, P) if Out (P) ∩ Γ = ∅
= block otherwise
Age (d, [SP] -> P) = [SP] -> Age (d, P)
Age (d, let $x_1=tx_1, \ldots x_n=tx_n$ in P) = Age (d, $[tx_1/x_1, \ldots tx_n/x_n]$ P)
Age (d, choice $x_1{:}s_1, \ldots x_n{:}s_n$ [] P) = choice $x_1{:}s_1, \ldots x_n{:}s_n$ [] Age (d, P)
Age (d, X_i where $X_1 := P_1, \ldots X_n := P_n$) = Age (d, P_i)
if P_i where $X_1 := P_1, \ldots X_n := P_n$ is a guarded spec.
= block otherwise

Block appears in the definition above because we preferred to give a complete definition of Age, but the premise of GC2 ensures that we are never in the cases in which block appears (see also section 4.2.6).

Out (P) is defined as the set of gates at which actions are immediately possible. It is formally defined inductively on the syntax of behaviour expressions as:

Out (stop) = ∅
Out (i@t{d}; P) = ∅
Out ($ad_1, \ldots d_n$@t{d} [SP]; P) = {a} if DS |– [0/t] SP for some instantiation of the free variables
= ∅ otherwise
Out (exit ($d_1, \ldots d_n$){d}) = {δ}
Out (Δ^d P) = Out (P) if d = 0
= ∅ otherwise
Out (P [] Q) = Out (P) ∪ Out (Q)
Out (P |[Γ]| Q) = (Γ ∩ Out (P) ∩ Out (Q)) ∪ ((Out (P) ∪ Out (Q)) — Γ)
Out (P [> Q) = Out (P) ∪ Out (Q)
Out (P >> Q) = Out (P) — {δ}
Out (hide Γ in P) = Out (P) — Γ
Out ([SP] -> P) = Out (P) if DS |– SP for some instantiation of the free variables
= ∅ otherwise
Out (let x_1=tx_1, … x_n=tx_n in P) = Out ([tx_1/x_1, … tx_n/x_n] P)
Out (choice x_1:s_1, … x_n:s_n [] P) = Out (P)
Out (X_i where X_1 := P_1, … X_n := P_n) = Out (P_i)
if P_i where X_1 := P_1, … X_n := P_n is a guarded spec.
= ∅ otherwise

The problem of rule GC2 is that the premise is infinite when $Q(s_i)$ is so for some i. To avoid this problem, an auxiliary function, denoted gta, is proposed which takes a (closed) behaviour expression P as argument and evaluates the greatest timed arc that P can execute.

Thanks to this gta function, the GC2 rule can be replaced by the following:

(GC2') choice x_1:s_1, … x_n:s_n [] P $\xrightarrow{d}$ choice x_1:s_1, … x_n:s_n [] Age (d, P)
if 0 < d ≤ gta (choice x_1:s_1, … x_n:s_n [] P)

gta (S) is defined inductively on the syntax of (closed) ET-LOTOS behaviour expressions:

gta (stop) = ∞
gta (i@t{d}; P) = d,
gta ($ad_1, \ldots d_n$@t{d} [SP]; P) = ∞
gta (exit ($d_1, \ldots d_n$){d}) = ∞
gta (Δ^d P) = d
gta (P [] Q) = min (gta (P), gta (Q))
gta (P |[Γ]| Q) = min (gta (P), gta (Q))
gta (P [> Q) = min (gta (P), gta (Q))
gta (P >> Q) = gta (P) if δ ∉ Out (P)
= 0 otherwise
gta (hide Γ in P) = gta (P) if Out (P) ∩ Γ = ∅
= 0 otherwise
gta ([SP] -> P) = gta (P) if DS |– SP
= ∞ otherwise

gta (let x_1=tx_1, … x_n=tx_n in P) = gta ([tx_1/x_1, … tx_n/x_n] P)
gta (choice x_1:s_1, … x_n:s_n [] P) = min {gta ([tx_1/x_1, … tx_n/x_n] P) | [tx_i] ∈ Q(s_i), i = 1, … n},
gta (X_i where X_1 := P_1, … X_n := P_n) = gta (P_i) if P_i where X_1 := P_1, … X_n := P_n is a guarded spec.
= 0 otherwise

Propositions
(i) Out (P) = {name (a) ∈ L | P $\xrightarrow{a}$ }
(ii) gta (P) is the greatest d such that P $\xrightarrow{d}$.
The proofs are easily carried out by structural induction on guarded expressions; the base cases being stop, exit, action-prefix and delay. Finally, unguarded specifications block time.

Another solution to solve the problem faced with the generalized choice is proposed in [BLT 93]. There the authors use a harsher solution that consists in adding restrictions on the possible syntax that P can have in expressions like choice x_1:s_1, … x_n:s_n [] P. In [BLT 93] the required condition is that the timing operators should be guarded in P.

4.2. Properties

4.2.1. Consistency of the semantics

The first obvious requirement to be fulfilled is the consistency of the proposed operational semantics. Such consistency is indeed easily falsified in the presence of inference rules with negative premises.

Propositions 4.1
The operational semantics of ET-LOTOS specifications is consistent.

Proof
The transition system specification given in tables 1, 2, 3 and 4, which is intended to define an operational semantics of ET-LOTOS, is stratifiable according to definition 2.3.1 of [Gro 90b]. The following function *S* can be proved to be a stratification: *S (P $\xrightarrow{a}$ P') := rk (a) where rk (a) = 0 if a ∈ A and rk (a) = 1 if a ∈ D*. Therefore, applying theorem 2.4.2 of [Gro 90b], the semantics is consistent. ❒

4.2.2. Time determinism

The timed transitions are deterministic. This means that ∀ P. if P $\xrightarrow{d}$ P' and P $\xrightarrow{d}$ P" then P' = P".

The proof is obvious from the definition of the semantics.

4.2.3. Time density

It is obvious that the timed transitions are closed under the relation ≤, or more formally that:
if P $\xrightarrow{d}$ P' then ∀ d' ≤ d. ∃ P". P $\xrightarrow{d'}$ P".

Furthermore, if P $\xrightarrow{d}$ P' then ∀ d' < d. ∃ d". P $\xrightarrow{d'}$ P" $\xrightarrow{d"}$ P' and d = d' + d".

The proofs are easily carried out by structural induction.

4.2.4. Reverse persistency

If $P \xrightarrow{d} P'$ and $P \not\xrightarrow{a}$ and $P' \xrightarrow{a}$ then $\forall\, d' < d.\ P \xrightarrow{d'} P'' \not\xrightarrow{a}$

The proof is easily carried out by structural induction. The only non vacuous base case being Δ.

This means that no new action offer can appear when a process executes a timed arc $\xrightarrow{d}$ except at the end. This property allows us to write rules H3 and En3 as proposed in table 1. Indeed, without this property, we would have had to strengthen the negative premise of H3 (resp. En3) so that $P \not\xrightarrow{a}$ (resp. $P \not\xrightarrow{\delta}$) hold in every intermediate state along $P \xrightarrow{d} P'$.

4.2.5. Non additivity and non persistency

The timed transitions *are not* additive. This means that $P \xrightarrow{d1} P'$ and $P' \xrightarrow{d2} P''$ does not necessarily imply $\exists\, P''.\ P \xrightarrow{d1+d2} P''$.

The persistency property is not valid. This means that $P \xrightarrow{d} P'$ and $P \xrightarrow{a}$ does not necessarily imply $P' \xrightarrow{a}$

4.2.6. Blockage of time

We already mentioned that in our model, unguarded specifications block the progression of time. More precisely, if S is unguarded then gta (S) = 0.

Also Age (d, P) blocks time in some cases. However Age (d, P) only appears as an auxiliary operator which is needed to define the semantics of the generalized choice operator, and in this case Age (d, .) is never applied to a process P such that d > gta (P), and therefore never blocks time.

4.2.7. Strong bisimulation

The second important requirement is that strong bisimulation be a congruence. This is very important in order to be able to replace a part of an ET-LOTOS description by another strongly bisimilar process without changing the semantics of the description, i.e. the overall description remains strongly bisimilar to the original one.

We have first to define the meaning of strong bisimulation in our context. This is very simple because our underlying model is the usual LTS. Of course this LTS will generally be infinite states and infinitely branching with D as a dense time domain, but this does not matter here.

Consider a LTS = $\langle S, A \cup D, T, s_0 \rangle$.
A relation $\underline{R} \subseteq S \times S$ is a strong bisimulation iff $\forall \langle B_1, B_2 \rangle \in \underline{R}$,$\forall\, \alpha \in A \cup D$, we have

(i) if $B_1 \xrightarrow{\alpha} B_1'$, then $\exists\, B_2'$ such that $B_2 \xrightarrow{\alpha} B_2'$ and $\langle B_1', B_2' \rangle \in \underline{R}$
(ii) if $B_2 \xrightarrow{\alpha} B_2'$, then $\exists\, B_1'$ such that $B_1 \xrightarrow{\alpha} B_1'$ and $\langle B_1', B_2' \rangle \in \underline{R}$

This is the classical definition of a strong bisimulation, where timed arcs from *D* are considered as any other transitions. The strong bisimulation equivalence between two LTS is then defined as follows.

Definition 4.2
Two LTSs $Sys_1 = \langle S_1, A \cup D, T_1, s_{0_1} \rangle$ and $Sys_2 = \langle S_2, A \cup D, T_2, s_{0_2} \rangle$ are strong bisimulation equivalent, denoted $Sys_1 \sim Sys_2$, iff

$\exists$ a strong bisimulation relation $\underline{R} \subseteq S_1 \times S_2$, such that $<s_{0_1}, s_{0_2}> \in \underline{R}$

Proposition 4.3
In ET-LOTOS strong bisimulation ~ is a congruence.

Proof
It can be checked very easily that all the rules of tables 1, 2, 3 and 4 are in the *ntyft/ntyxt* formats and are well founded (refer to definitions 4.2 and 4.3 of [Gro 90b]). Moreover, this transition system specification is stratifiable (refer to the proofs of propositions 4.1); and therefore, applying theorem 4.4 from [Gro 90b], strong bisimulation is a congruence. ❒

Laws for strong bisimulation equivalence

All the laws listed in section B.2.2 (items a to k) of ISO 8807 (appendix B) are valid laws for strong bisimulation in ET-LOTOS. The proofs are straightforward.

New equivalence laws can be added whose proofs are easy.

Urgency			
i@t; P	~ i; [0/t] P		
Δ^d P [] i{d'}; Q	~ i{d'}; Q	if d' < d	
Δ^d P [> i{d'}; Q	~ i{d'}; Q	if d' < d	
a{d}; P [] i{d'}; Q	~ a{d'}; P [] i{d'}; Q	if d' ≤ d	
a{d}; P [> i{d'}; Q	~ a{d'}; P [> i{d'}; Q	if d' ≤ d	
exit{d} [] i{d'}; Q	~ exit{d'} [] i{d'}; Q	if d' ≤ d	
Time determinacy			
Δ^0 P	~ P		
Δ^d P [] $\Delta^{d'}$ Q	~ $\Delta^{d'}$ (P [] $\Delta^{d-d'}$ Q)	if d' ≤ d	
Δ^d P |[Γ]| $\Delta^{d'}$ Q	~ $\Delta^{d'}$ (P |[Γ]| $\Delta^{d-d'}$ Q)	if d' ≤ d	
Others			
a{d}; P [] a{d'}; P	~ a{d}; P	if d' ≤ d and name(a) ≠ i	
exit{d} [] exit{d'}	~ exit{d}	if d' ≤ d	
exit{d} |[Γ]| exit{d'}	~ exit{d'}	if d' ≤ d	
a@t{d} [SP]; P	~ stop	if $\exists$ t'. DS |– [t'/t] SP	(name (a) ≠ i)
a@t [0≤t≤d]; P	~ a{d}; P	if t is not free in P	(name (a) ≠ i)

4.2.8. Strong timed bisimulation

The strong timed bisimulation is an equivalence that is a bit coarser than the strong bisimulation. In this new equivalence, we abstract away from breaks between successive timed arcs. Modulo this equivalence we will get back the time additivity property (see later on).

Definitions 4.4
Let d ∈ D, a ∈ A:

$P \xrightarrow[*]{d} Q$ iff $P \xrightarrow{d1} \xrightarrow{d2} \ldots \xrightarrow{dn} Q$ where $d = \sum_{i=1}^{n} d_i$

$P \xrightarrow[*]{a} Q$ iff $P \xrightarrow{a} Q$

Consider a LTS = <S, A ∪ D, T, s_0> .
A relation $\underline{R} \subseteq S \times S$ is a strong timed bisimulation iff $\forall <B_1, B_2> \in \underline{R}$,$\forall \alpha \in A \cup D$, we have

(i) if $B_1 \stackrel{\alpha}{\Rightarrow}_* B_1'$, then $\exists\, B_2'$ such that $B_2 \stackrel{\alpha}{\Rightarrow}_* B_2'$ and $<B_1', B_2'> \in \underline{R}$

(ii) if $B_2 \stackrel{\alpha}{\Rightarrow}_* B_2'$, then $\exists\, B_1'$ such that $B_1 \stackrel{\alpha}{\Rightarrow}_* B_1'$ and $<B_1', B_2'> \in \underline{R}$

Two LTSs $Sys_1 = <S_1, A \cup D, T_1, s_{0_1}>$ and $Sys_2 = <S_2, A \cup D, T_2, s_{0_2}>$ are strong timed bisimulation equivalent, denoted $Sys_1 \simeq Sys_2$, iff

$\exists$ a strong timed bisimulation relation $\underline{R} \subseteq S_1 \times S_2$, such that $<s_{0_1}, s_{0_2}> \in \underline{R}$

Propositions 4.5

(i) $\simeq$ is a congruence

(ii) $P \sim Q$ implies $P \simeq Q$

The proofs are straightforward.

Equivalence laws

Time additivity

$\Delta^d\ \Delta^{d'}\ P \simeq \Delta^{d+d'}\ P$

$\Delta^d\ stop \simeq stop$

Persistency

$a@t;\ P\ []\ \Delta^d\ a@t;\ [t+d/t]\ P \simeq a@t;\ P$

Expansion theorems

In this section, the behaviour expressions are in the following general format $\sum_{i\ \in\ I} \Delta^{d_i}\ a_i@t_i\{r_i\};\ P_i$.

It is thus assumed that the elements of the summation (which means choice) can always be enumerated by some (possibly infinite) suitably chosen index set I.

Let $P := \sum_{i\ \in\ I} \Delta^{d_i}\ a_i@t_i\{r_i\};\ P_i$ and $Q := \sum_{j\ \in\ J} \Delta^{d'j}\ b_j@t'_j\{r'_j\};\ Q_j$

$$\Delta^d\ P \simeq \sum_{i\ \in\ I} \Delta^{d+d_i}\ a_i@t_i\{d_i\};\ P_i$$

$$P\ |[\Gamma]|\ Q \simeq \sum_{i\ \in\ I} \{\Delta^{d_i}\ a_i@t_i\{r_i\};\ (P_i\ |[\Gamma]|\ Age\ (d_i+t_i, Q))\ \mid\ name\ (a_i) \notin \Gamma\}$$
$$[]\ \sum_{j\ \in\ J} \{\Delta^{d'j}\ b_j@t'_j\{r'_j\};\ (Age\ (d'_j+t'_j, P)\ |[\Gamma]|\ Q_j)\ \mid\ name\ (b_j) \notin \Gamma\}$$
$$[]\ \sum \{\Delta^x\ c@y\{z\};\ ([y+x-d_i/t_i]\ P_i\ |[\Gamma]|\ [y+x-d'_j/t'_j]\ Q_j))\ \mid\ c=a_i=b_j,\ name\ (c) \in \Gamma,$$
$$x = max\ (d_i, d'_j),\ z = min\ (d_i+r_i-x, d'_j+r'_j-x),\ z \geq 0,\ i \in I, j \in J\}$$

$$P\ [>\ Q \simeq Q\ []\ \sum_{i\ \in\ I} \Delta^{d_i}\ a_i@t_i\{r_i\};\ (P_i\ [>\ Q)$$

$$hide\ \Gamma\ in\ P \simeq \sum_{i\ \in\ I} \{\Delta^{d_i}\ a_i@t_i\{r_i\};\ hide\ \Gamma\ in\ P_i\ \mid\ name\ (a_i) \notin \Gamma\}$$
$$[]\ \sum_{i\ \in\ I} \{\Delta^{d_i}\ i;\ hide\ \Gamma\ in\ [0/t_i]\ P_i\ \mid\ name\ (a_i) \in \Gamma\}$$

4.2.9. Weak timed bisimulation

Definitions 4.6

Let $d \in D$, $a \in L$ and ε the empty transition:

$P \overset{d}{\Rightarrow} Q$ iff $P (\overset{i}{\rightarrow})^* \overset{d1}{\underset{*}{\rightarrow}} (\overset{i}{\rightarrow})^* \overset{d2}{\underset{*}{\rightarrow}} (\overset{i}{\rightarrow})^* \ldots \overset{dn}{\underset{*}{\rightarrow}} (\overset{i}{\rightarrow})^* Q$ where $d = \sum_{i=1}^{n} d_i$

$P \overset{a}{\Rightarrow} Q$ iff $P (\overset{i}{\rightarrow})^* \overset{a}{\rightarrow} (\overset{i}{\rightarrow})^* Q$

$P \overset{\varepsilon}{\Rightarrow} Q$ iff $P (\overset{i}{\rightarrow})^*$

Consider a LTS = $<S, A \cup D, T, s_0>$.

A relation $\underline{R} \subseteq S \times S$ is a weak timed bisimulation iff $\forall <B_1, B_2> \in \underline{R}$,$\forall \alpha \in L \cup D \cup \{\varepsilon\}$:

(i) if $B_1 \overset{\alpha}{\Rightarrow} B_1'$, then $\exists B_2'$ such that $B_2 \overset{\alpha}{\Rightarrow} B_2'$ and $<B_1', B_2'> \in \underline{R}$

(ii) if $B_2 \overset{\alpha}{\Rightarrow} B_2'$, then $\exists B_1'$ such that $B_1 \overset{\alpha}{\Rightarrow} B_1'$ and $<B_1', B_2'> \in \underline{R}$

Two LTSs $Sys_1 = <S_1, A \cup D, T_1, s_{0_1}>$ and $Sys_2 = <S_2, A \cup D, T_2, s_{0_2}>$ are weak timed bisimulation equivalent, denoted $Sys_1 \approx Sys_2$, iff

$\exists$ a strong timed bisimulation relation $\underline{R} \subseteq S_1 \times S_2$, such that $<s_{0_1}, s_{0_2}> \in \underline{R}$

Proposition 4.7

(i) $\approx$ is a congruence in every context except [] and [>

(ii) $P \simeq Q$ implies $P \approx Q$

The proofs are straightforward.

Equivalence laws

$P \approx i; P$ (Remember that i; P is a shorthand notation for i{0}; P)

$P [] i; P \approx i; P$

$a; (P_1 [] i; P_2) [] a; P_2 \approx a; (P_1 [] i; P_2)$

4.2.10. Upward compatibility

Consider the LOTOS process algebra LOTOS = $(OP, A, R_A^{OP}, \sim)$ where OP is a set of operators, A is the alphabet of actions, R_A^{OP} is the set of operational semantics rules and ~ the strong bisimulation equivalence. Consider ET-LOTOS as the process algebra ET-LOTOS = $(OP', A', R_{A'}^{OP'}, \sim_E)$ where OP' is a superset of OP, A' = A $\cup$ D is a superset of A, $R_{A'}^{OP'}$ is the new set of rules, and $\sim_E$ is the strong bisimulation equivalence in ET-LOTOS ($\sim_E$ denotes our ~ as defined in section 4.2.7).

Our definition of upward compatibility is the one given in [NiS 92] where the following two requirements are stated. They are translated to the LOTOS framework as follows:

- Semantics conservation: $\forall r \in R_A^{OP}$. r is valid in $R_{A'}^{OP'}$ if it is applied on **LOTOS** terms.
  The rules R_A^{OP} remain valid in ET-LOTOS as far as they are applied on LOTOS terms.
- Isomorphism: $\forall P, Q \in$ **LOTOS**. $P \sim Q$ iff $P \sim_E Q$.
  The theory of processes in ET-LOTOS is isomorphic to that of the restriction of ET-LOTOS to constructs of LOTOS.

The semantics conservation is obvious since the first column of ET-LOTOS semantic rules is a superset of the LOTOS semantic rules.

The isomorphism of the (ET-LOTOS, $\sim_E$) and the (LOTOS, ~) theories is only true for guarded specifications. It is easily derived from the following basic two lemmas.

Lemmas 4.8

(i) $\forall P, Q \in$ LOTOS. $\forall a \in A$. $P \overset{a}{\rightarrow} Q$ according to R_A^{OP} iff $P \overset{a}{\rightarrow} Q$ according to $R_{A'}^{OP'}$

(ii) According to $R_{A'}^{OP'}$, $\forall P \in$ LOTOS.

- if $P \xrightarrow{i}$ then $\forall$ d. $P \not\xrightarrow{d}$
- if $P \not\xrightarrow{i}$ then $\forall$ d. $P \xrightarrow{d} P$ for guarded specifications

The proof of (i) is obvious since the axioms and inference rules that are applicable are the same. The proofs of (ii) are carried out by structural induction. For the first part the non vacuous base case is i;P. For the second part the non vacuous base cases are stop and a;P with $a \neq i$.

With unguarded specifications however the isomorphism between the theories is not true any more. For example, in LOTOS, P := stop and Q := Q are strong bisimulation equivalent, whereas in ET-LOTOS, $P \xrightarrow{d}$ but $Q \not\xrightarrow{d}$. Note that discriminating these two processes is considered more as an asset than as a shortcoming.

Moreover, for guarded specifications and as far as we have checked, the LOTOS laws for strong (resp. weak) bisimulation equivalence remain true $\forall$ P, Q $\in$ **ET-LOTOS** (i.e. not only on LOTOS terms), thereby preserving the LOTOS intuition in ET-LOTOS.

5. An Example of the Use of ET-LOTOS

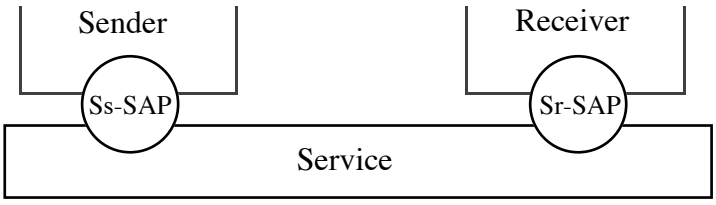
In order to justify our design choices and prove their advantages, we will illustrate here the expressiveness of ET-LOTOS by applying it to the specification of a small system, taken from the Tick-Tock case study [LLD 94], which has been specially designed to assess timed FDTs. Some features have been added to the original version to present a more complete overview of the capabilities of ET-LOTOS.

5.1. 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)

[1] This subsection is mainly made of excerpts from [LLD 94], with slight changes

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.

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*.

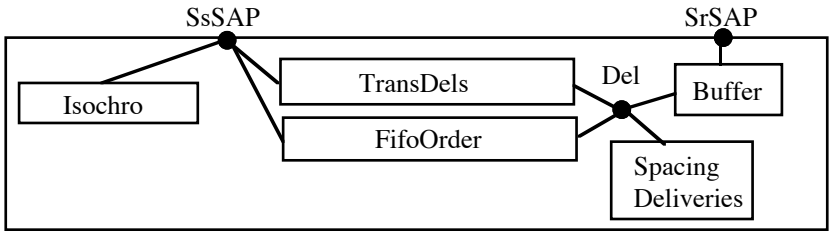
Let us notice that there is no incompatibility between the constraint “transmission delays” and the constraints “spacing between the deliveries” and “FIFO-ordering of cells”, as long as the minimal period of admission η is greater or equal to the minimal delay between two deliveries α. We will suppose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

5.2. ET-LOTOS Specification of *Service*

We have tried to describe *service* in the constraint oriented style [Bri 89], 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 same difficulty was faced in [MFV 94] and, to our knowledge, in other timed extensions too.

The picture below shows the general structure of the specification, represented by process `Service` (without process `Crash`):




```
process Service [SsSAP,SrSAP] : noexit :=
(Isochro [SsSAP] (π,0,0,0,0)
 |[SsSAP]|
 (hide Del in (TransDels [SsSAP,Del]
               |[SsSAP,Del]|
               FifoOrder [SsSAP,Del] (NoCell)
               |[Del]|
               SpacingDeliveries [Del]
               |[Del]|
               Buffer [Del, SrSAP]
))            )
[> 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.
- `Buffer` expresses the immediate acceptation (or loss) constraint.
- `Crash` describes the effects of a crash of the system.

As we can see, `TransDels`, `FifoOrder`, `SpacingDeliveries` and `Buffer` synchronise on the internal gate `Del`. The reason for this will appear more clearly in the individual presentation of each process. This internal 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 others. 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 four processes are ready for it. The occurrence of `Del` thus means that the transmission of the cell must be transmitted (by `Buffer`) immediately or never.

This problem enlightens a weakness of ET-LOTOS, but to our knowledge, no other timed extension of LOTOS could do better. 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 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] (per:time, slot,transm,slot1,transm1:Nat) : noexit:=
SsSAP{0} ?c:cell; Δ^per
 ([not(slot eq 9)] -> Isochro [SsSAP](per,succ(slot),succ(transm),succ(slot1),succ(transm1))
  [] [(slot eq 9) and (transm eq 9)] -> Isochro [SsSAP](min(per/2,η),0,0,0,0)
  [] [(slot eq 9) and not(transm eq 9) and not(slot1 eq 29)] ->
                                        Isochro [SsSAP](per,0,0,succ(slot1),succ(transm1))
  [] [(slot1 eq 29) and not(transm eq 9)] ->
          ([transm1 le 15] -> Isochro [SsSAP](max(per*3/2,ψ),0,0,0,0)
           []
           [transm1 ge 16] -> Isochro [SsSAP](max(per*6/5,ψ),0,0,0,0)
 )         )
[]
Δ^per
([not(slot eq 9)] -> Isochro2 [SsSAP](per,succ(slot),transm,succ(slot1),transm1)
 [] [(slot eq 9) and not(slot1 eq 29)] -> Isochro [SsSAP](per,0,0,succ(slot1),transm1)
 [] [slot1 eq 29] ->
       ([transm1 le 15] -> Isochro [SsSAP](max(per*3/2,ψ),0,0,0,0)
        []
        [transm1 ge 16] -> Isochro [SsSAP](max(per*6/5,ψ),0,0,0,0)
)       )
endproc (* Isochro *)
```

`Isochro` 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 `Isochro` is called after a delay `per`, with the parameters changed to take account of the receipt of a new cell. In the second case, a new occurrence of `Isochro` is called after a delay `per`, with the parameters changed to take account of the rejection of the offer. In particular, in both cases

the new period is calculated. `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.

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

```
process TransDels [SsSAP,Del] : noexit :=
SsSAP?c:Cell ;  i{τmin,τmax} ; Del!c ; stop
|||
TransDels [SsSAP,Del]
endproc (* TransDels *)
```

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 Buffer [Del,SrSAP] : noexit :=
Del?c:cell ; (SrSAP{0}!c ; Buffer [Del,SrSAP]
              []
              Buffer [Del,SrSAP])
endproc (* Buffer *)
```

`Buffer` acts as a one place buffer that is almost always free. It normally outputs immediately through `SrSAP` the cells it inputs by `Del`. If the output is not possible immediately, it looses the cell immediately and waits for the next cell.

```
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 [δ,ϕ], which is expressed with the short notation `i`{δ,ϕ}.

6. Related works

In this section we compare our proposal with some other timed extensions of LOTOS.

The timed LOTOS proposed in [MFV 93, MFO 93] by Miguel has the same expressive power as ours but lacks some flexibility to specify the internal time choice. The reason is that all `i` are necessarily urgent in this proposal, i.e. the authors adopt full maximal progress.

The main difference with ET-LOTOS lies however in the semantics. The authors propose two versions of it which are both somehow open to criticism. The first version of the semantics is given in two steps. In the first step, an unrestricted transition system is generated in a standard manner, and in the second step, a restricted transition system is derived by removing transitions that do not fulfil the chosen urgency conditions. This definition lacks compositionality in the sense that the *restricted* transition system of a composed behaviour cannot be derived from the *restricted* transition systems of its components (but from the *unrestricted* ones). In the second version, a classical one-step semantics is given which has *infinite negative* premises in some inference rules when time is dense. Finally, their semantics is not presented in two columns with a clear separation of concerns between the progression in time and the execution of actions. Both are intertwined because the authors use an alphabet of extended actions that are composed of a usual action and of a time stamp.

The timed LOTOS proposed in [BLT 93] differs from ours on a basic point: observable action necessity. In their proposal the authors can write a process that enforces an *observable* action to occur in a certain interval (or blocks time if the environment does not allow it). We think that this facility is not needed and is even uncomfortable in practice (Refer to the example of section 2.1.2 on isochronism). The authors do not adopt maximal progress because they propose a more general operator that can urge observable actions. If these urgent interactions are then hidden (by a `hide`), this leads to maximal progress. The advantage is the decoupling between urgency and abstraction (hide). It is also more general because it allows the association of a time interval to explicitly hidden actions, instead of pure urgency.

Let us finally mention RT-LOTOS [CCE 93] which has been inspired by a predecessor of ET-LOTOS and is therefore very similar to our model. In RT-LOTOS a new operator is proposed to start an exception behaviour when a time constraint on some external action is not matched by the environment.

7. Conclusion and perspectives

The ET-LOTOS language has given convincing evidences of its suitability to model real-time distributed systems. The mathematical properties of the language are also quite satisfactory.

The development of a timed testing theory and of a translation to a tractable verification model are under study. One such model, called a timed graph model, is presented in [ACD 90]. A timed graph is a state-transition graph extended with a mechanism that allows the expression of constraints on the delays between the state transitions. Constraints are expressed as predicates on state variables representing timers. In addition to the limited state explosion, there exist model checking methods for temporal logics with quantitative temporal operators which are directly applicable to them. In [NSY91] a method for the translation of ATP into timed graph is presented. For Timed PNs, a similar finite representation exists which combines the usual marking with inequalities on time values. Therefore, it is likely that timed graphs or another similar model be more adequate than a usual LTS for verifying ET-LOTOS specifications.

Acknowledgements

We are grateful to T. Bolognesi, J.-P. Courtiat, C. Pecheur, J. Quemada and T. Regan for several interesting discussions on ET-LOTOS and its predecessors.

References

[ACD 90] R. Alur, C. Courcoubetis, D. Dill, *Model-checking for real-time systems*, in: Fifth annual IEEE symposium on logic in computer science, Philadelphia, P.A., June 1990, 414-425.

[Azc 90] A. Azcorra-Saloña, *Formal Modeling of Synchronous Systems,* Ph. D. Thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, Spain, Nov. 1990.

[BaB 91] J.C.M. Baeten, J.A. Bergstra, *Real Time Process Algebra,* Formal Aspects of Computing 3 (2) 142-188, 1991.

[BeK 85] J.A. Bergstra, J. W. Klop, *Algebra of Communicating Processes with Abstraction,* Theoretical Computer Science 37 (1985) 77-121 (North-Holland, Amsterdam).

[BLL 94] Y. Baguette, L. Léonard, G. Leduc, A. Danthine, *The OSI95 Connection-mode Transport Service*, to appear in A. Danthine, ed., The OSI95 Transport Service with Multimedia Support on HSLAN's and B-ISDN (Springer-Verlag, Berlin, 1994).

[BLT 90] T. Bolognesi, F. Lucidi, S. Trigila, *From Timed Petri Nets to Timed LOTOS,* in: L. Logrippo, R. Probert, H. Ural, eds., Protocol Specification, Testing and Verification, X (North-Holland, Amsterdam, 1990) 395-408.

[BLT 93] T. Bolognesi, F. Lucidi, S. Trigila, *Towards Timed Full LOTOS,* in: Proceedings of First AMAST International Workshop on Real-Time Systems, Nov. 1993, Iowa City, Iowa, USA, 28 p.

[BLT 94] T. Bolognesi, F. Lucidi, S. Trigila, *Converging Towards a Timed LOTOS Standard,* to appear in: Computer Standards and Interfaces, 1994.

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

[BoL 92a] 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.

[BoL 92b] T. Bolognesi, F. Lucidi, *Timed Process Algebras with Urgent Interactions and a Unique Powerful Binary Operator,* in: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg, eds., Real-Time: Theory and Practice (LNCS 600, Springer-Verlag, Berlin, 1992).

[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. de Camargo, 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).

[DaS 89] J. Davies, S. Schneider, *An introduction to timed CSP,* Rept. No. PRG-75, Oxford University Computing Laboratory, Programming Research Group, Aug. 1989.

[Gro 90a] J. F. Groote, *Specification and Verification of Real Time Systems in ACP,* in: L. Logrippo, R. Probert, H. Ural, eds., Protocol Specification, Testing and Verification, X (North-Holland, Amsterdam, 1990) 261-274.

[Gro 90b] J. F. Groote, *Transition system specifications with negative premises,* in: J.C.M. Baeten, J.W. Klop, eds., CONCUR '90, Theories of Concurrency: Unification and Extension, LNCS 458 (Springer - Verlag, Berlin, 1990) 332-341.

[HaJ 90] H. Hansson, B. Jonsson, *A calculus for communicating systems with time and probabilities*, in: 11th IEEE Real-Time Systems Symposium, Orlando, Florida, 1990, IEEE Computer Society Press.

[Han 91] H. Hansson, *Time and Probability in Formal Design of Distributed Systems,* Ph. D Thesis, DoCS 91/27, Uppsala University, Dept. of Computer Science, P.O. Box 520, S-75120 Uppsala, Sweden.

[HeR 90] M. Hennessy, T. Regan, *A temporal process algebra,* in: J. Quemada, J. Mañas, E. Vazquez, eds., Formal Description Techniques, III (North-Holland, Amsterdam, 1991) 33-48.

[Hoa 85] C.A.R. Hoare, *Communicating Sequential Processes,* (Prentice-Hall Internat., London, 1985).

[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, Feb. 1989.

[Led 92] G. Leduc, *An upward compatible timed extension to LOTOS,* in: K. Parker, G. Rose, eds., Formal Description Techniques, IV (North-Holland, Amsterdam, 1992) 217-232.

[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).

[LéL 94] L. Léonard, G. Leduc, *An Enhanced Version of Timed LOTOS and its Application to a Case Study,* to appear in A. Danthine, ed., The OSI95 Transport Service with Multimedia Support on HSLAN's and B-ISDN (Springer-Verlag, Berlin, 1994).

[LLD 94] 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 Transport Service with Multimedia Support on HSLAN's and B-ISDN (Springer-Verlag, Berlin, 1994).

[MeF 76] P.M. Merlin, D.J. Farber, *Recoverability of Communication Protocols - Implications of a theoretical Study,* IEEE Transaction on Communication, 24, Sept. 76, 1036-1043.

[MFO 93] C. Miguel, A. Fernández, J. Ortuño, L. Vidaller, *A LOTOS based Performance Evaluation Tool*, Computer Networks and ISDN Systems, Vol. 25, No7 (1993) 791-814.

[MFV 93] C. Miguel, A. Fernández, L. Vidaller, *Extending LOTOS towards performance evaluation*, in: M. Diaz, R. Groz, eds., Formal Description Techniques, V (North-Holland, Amsterdam, 1993) 103-118.

[MFV 94] C. Miguel, A. Fernández, L. Vidaller, *Assessment of Extended LOTOS*, to appear in A. Danthine, ed., The OSI95 Transport Service with Multimedia Support on HSLAN's and B-ISDN (Springer-Verlag, Berlin, 1994).

[Mil 89] R. Milner, *Communication and Concurrency,* (Prentice-Hall International, London, 1989).

[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, *The Algebra of Timed Processes ATP: Theory and Application,* Rept. No. RT-C26, Projet Spectre, LGI-IMAG, Nov. 1991.

[NiS 92] 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.

[NRS 90] X. Nicollin, J.-L. Richier, J. Sifakis, J. Voiron, *ATP : An algebra for timed processes,* in: M. Broy, C.B. Jones, eds., IFIP Working Conference on Programming Concepts and Methods, Sea of Gallilee, Israel (North-Holland, Amsterdam, 1990).

[NSY 91] X. Nicollin, J. Sifakis, S. Yovine, *From ATP to Timed Graphs and Hybrid Systems,* in: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg, eds., Real-Time: Theory and Practice (LNCS 600, Springer-Verlag, Berlin, 1992) 549-572.

[QAF 90] J. Quemada, A. Azcorra, D. Frutos, *A timed calculus for LOTOS,* in: S. T. Vuong, ed., Formal Description Techniques II (North-Holland, Amsterdam, 1990) 195-209.

[QFA 92] J. Quemada, D. de Frutos, A. Azcorra, *TIC: A TImed Calculus*, Formal Aspects of Computing 3, 1992.

[QuF 87] J. Quemada, A. Fernandez, *Introduction of Quantitative Relative Time into LOTOS,* in: H. Rudin, C.H. West, eds., Protocol Specification, Testing and Verification, VII, (North-Holland, Amsterdam, 1987, ISBN 0-444-70293-8) 105-121.

[Ree 90] G.M.Reed, *A Hierarchy of Domains for Real Time Distributed Computing,* in: M. Main, A. Melton, M. Mislove, D. Schmidt, eds., Mathematical Foundations of Programming Semantics (LNCS 442, Springer-Verlag, Berlin Heidelberg New York, 1990) 80-128.

[Reg 93] T. Regan, *Multimedia in Temporal LOTOS: a Lip-Synchronization Algorithm*, in: A. Danthine, G. Leduc, P. Wolper, eds., Protocol Specification, Testing and Verification, XIII (North-Holland, Amsterdam, 1993) 127-142.

[ReR 88] G.M.Reed, A.W. Roscoe, *A Timed Model for Communicating Sequential Processes,* Theoretical Computer Science 58 (1988) 249 - 261 (North-Holland, Amsterdam).

[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.