TE-LOLA: A Timed Extended LOLA Prototype

Gualberto Rabay*and Juan Quemada
Dept. of Telematics Engineering
Technical University of Madrid
28040 Ciudad Universitaria, Madrid
grabay,quemada@dit.upm.es

Abstract

TE-LOTOS (Time Extended LOTOS) is the current ISO proposal for extending Standard LOTOS with quantitative time. In order to support TE-LOTOS analysis and verification a prototype tool called TE-LOLA has been implemented. TE-LOLA is based on LOLA, its untimed predecessor, which has been successfully applied to LOTOS specifications. In this paper we describe TE-LOLA functionalities and show its application to a small example.

1 Introduction

TE-LOTOS (Time Extended LOTOS) [8] is the current ISO proposal for extending Standard LOTOS [4] with quantitative time. TE-LOTOS is the timed part of a more ambitious proposal, called E-LOTOS, that intends to introduce both features in the behaviour part and in the data part as well. TE-LOTOS language is the result of a mature research process that started in 1987 with the work of Fernandez and Quemada [15]. Further research towards a timed extension to LOTOS continued and the main contributions to current proposal are [13, 6, 7, 9, 14]. Real-time features were introduced by definition of timed events which may happen in concrete instants of time and the addition of a new delay operator. TE-LOTOS is upward compatible with LOTOS.

LOLA [17, 11] is a Full Standard LOTOS exploration and analysis tool which is one of the components of the DIT/UPM Tool Set. TE-LOLA (Timed Extended Lotos LAboratory) is a LOLA based prototype that supports full TE-LOTOS. TE-LOLA has almost the same functionalities as LOLA and it is able to check properties, to check testing equivalence, to transform and to execute specifications in a step by step mode.

The next sections are organized as follows. In Section 2 we give a short description of TE-LOTOS model including its syntax and semantics. In Section 3 we describe the main functionalities of TE-LOLA and comment some implementation details. In Section 4 we show an example of the use of TE-LOLA applied to a simple protocol specification analysis. Finally in Section 5 we present some conclusions and future work.

*On leave from Telebras. Supported by grant 200584/89-8 from CNPq/Brazil.

2 TE-LOTOS Model in Brief

In this section we briefly introduce TE-LOTOS model and we assume that the reader is familiar with Standard LOTOS (see [1] for a tutorial). For further details and a complete formal description of TE-LOTOS the interested reader should see the ISO Proposal [8] and additionally, the contribution papers we mentioned in the previous section.

2.1 Time Domain and Data Types

TE-LOTOS data part is described by the Abstract Data Type language ACT ONE. The time domain is defined as a set of values from a given data sort. The definition of this data type is left free to the user, but it must observe some properties (existence of a total order relation, a bottom element, an infinity element, and a commutative and associative sum operation) as defined in the TE-LOTOS Proposal. TE-LOTOS allows discrete and dense time domain descriptions as well, but it must be countable in order to be able to describe its operational semantics as a Labelled Transition System.

2.2 Syntax

TE-LOTOS syntax has minor changes with respect to Standard LOTOS. A time attribute was attached to internal actions, observable actions and terminations while other operators are kept unchanged. Only a new delay operator was added. In Table 1 we present the syntax of a representative subset of the language in a BNF expression with these new elements in boldface.

Table 1: TE-LOTOS Syntax

$$
\begin{aligned}
P := \; & stop \,|\, exit(e_1..e_n)\{\mathbf{T}\} \; |gd_1..d_n \; \{\mathbf{t\ in\ T}\}[SP]; P \\
& |i \; \{\mathbf{t\ in\ T}\} \; ; P \,|\, \mathbf{Wait(t1)}; P \,|\, P[]P \,|\, P|[A]|P \\
& |hide\, A \; in \; P \,|\, P >> \; accept \, x_1 : s_1..x_n : s_n \; in \; P \\
& |P[> P \,|\, [SP]-> P \,|\, let \, x_1 = tx_1..x_n = tx_n \; in \; P \\
& |P \,|\, choice \; x_1 : s_1, ..x_n : s_n[]P
\end{aligned}
$$

The time attribute is optional and consists of a time variable t and a time interval T as represented in the syntax table. Time variables can appear only in internal and observable action prefix. The time interval has the form $t^-..t^+$ but it can be represented as t^- in case t^- is equal to t^+. If T is omitted in an observable action, it is equivalent to $T = 0..\infty$. In the case of internal action omission of T, it is equivalent to a time interval $T = 0..0$.

2.3 Semantics

The current TE-LOTOS Proposal has two different semantic models as described respectively in its body and in the Annex 1. The first follows [10], where the alphabet of actions is extended with a separate set of time actions. In this model a process evolves executing actions or time transitions. The operational semantics is represented by separate axioms and inference rules for actions and time transitions.

The alternative semantic model was first defined in [15] and the alphabet of actions is replaced by a timed alphabet. In this case a process evolves executing these timed actions. This composed time-action transition system abstracts from time transitions and therefore has a more compact representation as it can be observed in Figure 1.

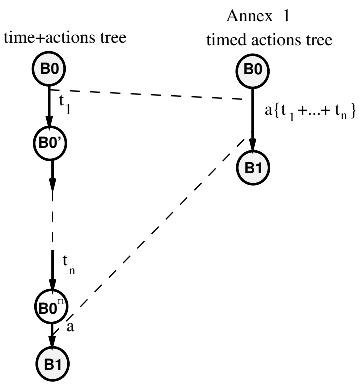

Figure 1: Time/action tree versus timed-actions tree

The second semantic model has been adopted for the tool design because it generates a simpler implementation of the simulation part concerning the computing of offers as the user interface as well. Following this decision from now on we will be referring to the semantics as described in the Annex 1 of the TE-LOTOS proposal. This represents no restriction of use since the two models have the same Expansion Theorem and were proved to be equivalent in [19].

Let B be any finite behaviour expression. Then the operational semantics of TE-LOTOS is defined by a labelled transition system which is defined as follows.

Definition 2.1

We define the transition system of B as the labelled transition system $\langle \mathcal{B}, \mathcal{E} * \mathcal{T}, TR, B_0 \rangle$ where $\mathcal{B}$ is the set of states, $\mathcal{E} * \mathcal{T}$ is the Cartesian product of gate labels and time instants where each pair represents a timed event, $B_0 = B$ is the initial state, and TR is the set of labelled transitions.

□

The set of labelled transitions is derived from applying a set of axioms and inference rules as defined in [8]. Here we will discuss the semantics in a less formal way.

Time Attribute. The time intervals indicate the earlier and latter instant of time of execution of the action after the execution of the preceding event. In $a\{1\}; b\{2\}; stop$, a can occur 1 instant of time after being enabled and b can occur 2 instants of time after the occurrence of a. The time variables are used to register the precise instant of time of the occurrence of an action and then can be used in the specification as any other variable. An observable action can have its execution disabled if time advances beyond upper bound value of its time interval as we will discuss in the next item.

Concurrency. TE-LOTOS uses interleaving semantics and therefore time is implicitly global. In other words, when a process in a parallel composition executes an (timed) action, all inactive components must be aware of the passage of the time. This is accomplished by a semantic operator called Age that updates the temporal attributes of those processes. For instance

$a\{1\}; b\{3\}; stop \,|[[b]|\, c\{3\}; b\{1\}; stop$

is transformed into $b\{3\}; stop \,|[b]|\, c\{2\}; b\{1\}; stop$ after the execution of $a1$ and the corresponding aging of the behaviour expression on the right side of the parallel operator. Using the same reasoning, one possible execution can be derived from this behaviour:

$b\{3\}; stop \,|[b]|\, c\{2\}; b\{1\}; stop \xrightarrow{c\{2\}}$
$b\{1\}; stop \,|[b]|\, b\{1\}; stop$

$b\{1\}; stop\,|[b]|\, b\{1\}; stop \overset{b\{1\}}{\rightarrow} stop$

As we could see in the above example, synchronization is possible only if time intervals of participating actions have no empty intersection. Therefore, time imposes an additional constraint to synchronization beside untimed constraints such as name matching, offer list compatibility, guard conjunction and true predicate evaluation.

Urgency. TE-LOTOS uses *urgency*, or *maximal progress*, for internal events. When urgent events are enabled, time can not advance. Internal events produced by the use of the *hide* operator use the lower bound of its time interval as the urgency instant. For instance

$hide\ a\ in\ a\{2..4\}; stop|||b\{3..5\}; stop$

forces the execution of the hidden a in 2 instants of time.

In the case of internal action prefix, the model adopts the upper bound of the time interval as the urgency instant. For instance, in $i\,\{1..6\}; B$ the internal action must happen at any time between 1 and 6.

3 TE-LOLA Description

Most of the functionalities of TE-LOLA are inherited from LOLA and extended with time features. The main differences with respect to the untimed versions are:

1. *Synchronization:* It is necessary to compute the intersection of the time intervals of participating actions and verify if it is not empty.
2. *Aging:* In compositions with the parallel or disabling operator it is necessary to age time attributes of actions from inactive processes.
3. *Urgency:* The computation of the labelled transition system is done in two steps. First, it computes all possibles events from current state and then, it applies the urgency criterion pruning those branches that do not observe it.

For efficiency reasons, TE-LOLA uses a discrete time domain as described in a data time library that is distributed with the tool. This data type description has a set of equations particularly defined for evaluation of symbolic expressions.

3.1 Functionality

There are 3 types of transformations supported in TE-LOLA according to its objective:

Expansions. The *Expansion* transformation produces a compressed version of the TE-LOTOS transition system representing its Extended Finite State Machine (EFSM). These transformations use a generalized version of the Timed Expansion Theorem [8]. Its main applications are state exploration, deadlock detection, implementation derivation, etc. The supported expansions are: Expand, Free Expand, Parameterized Timed Expand and Interleaved Expand.

- **Expand.** This *Expansion* transforms a LOTOS behaviour into another equivalent one that only has observable and internal action prefixes, choices, sum expressions, exits, stops, guards and processes. The resulting behaviour is strongly/weakly equivalent to the original one. This expansion compresses the state space due to duplicate behaviour detection mechanisms. To terminate this expansion, it is necessary to use only finite data type sorts, bounded dynamic process creation. In the timed model termination also depends on the time interval limits and on the type of composition between recursive processes. The expansion of any number of interleaving recursive processes having events with time intervals may not terminate.
- **Timed Parameterized Expand.** This expansion is based on the *Parameterized Expansion* [18] and differs from the *Expand* in keeping variables symbolically. It also uses an algorithm that parameterizes aging expressions. It achieves a greater state space compression and can expand recursive behaviours with infinite data types and time intervals.
- **Free Expand.** This expansion is the simplest and consequently the fastest one. It does not have any mechanism of detection and, therefore, it should not be applied to recursive behaviours.
- **Interleaved Expand.** The *interleaved expansion* [5, 16] produces a system representation where interleaved behaviours are kept without expanding. This representation allows a significant state space reduction and targets the optimization of validation and verification algorithms and the production of more efficient implementations.

Testing. Testing transformations [12] calculate the response of a system specification with respect to a test (*must, may* or *reject*) by state exploration of the parallel composition between the test and the behaviour under test. The evaluation follows the Nicola and Hennessy [2] testing equivalence definitions. If the test has actions with intervals, the verdict can be inconclusive due to unevaluated guards that appear in timed expansions.

Simulation/Debugging. The simulation mode allows the step by step symbolic execution of the specification.

It is also possible to evaluate expressions according to rewrite rules defined in the specification.

- **Step.** In this mode the possible transitions at each step are available in a menu. Many facilities are available such as free variables value assignment, simulation traces printing, step undoing and options to move through the specification and simulate any of its components. In case there is a time interval in the selected transition, the user is prompted to enter a concrete value of time. If the entered time value violates urgency constraints, the tool issues a warning and reoffers the current menu.
- **Rewrite.** This mode allows evaluation of data types expressions using rewriting rules as defined in the data type part of the specification.

3.2 Input Language

TE-LOLA uses another component (TOPO) from the DIT/UPM toolset as a front-end that checks the syntax and the static semantics of Standard LOTOS specifications. The final syntax of E-LOTOS has not been defined yet, therefore front-end tools are not available. For this reason current input language for TE-LOLA is Standard LOTOS and an *ad hoc* solution was adopted in order to represent time constraints.

First, two special gates named `time` and `wait` are added to gate lists of process definition, process instantiation and specification whenever it is necessary to introduce time constraints or the delay operator, respectively. Second, time constraints are expressed by prefixing actions with the special gate *time* whose offer list represents the time attributes. Introduction of delays is done in a similar way by prefixing the delayed behaviour with the gate *wait* and an offer that represents the delay value. For example, to represent the TE-LOTOS expression $a\{x\ in\, t_1..t_2\}; Wait(y); B$ it should be written $time?x : time!t_1!_2!; a; wait!y; B$. In table 2, all possible time constructions from TE-LOTOS and respective representation as those accepted by TE-LOLA are shown. Although it is not a very elegant solution, the procedure adopted is easy to use and fulfils the purpose of checking the time semantics. When a specification is loaded, the first task the tool does is to eliminate all those special gates and to generate TE-LOTOS time attributes.

Table 2: TE-LOLA Input Syntax

TE-LOTOS syntax	TE-LOLA syntax
$i; B$	$i; B$
$i\{t\ in\ t_1..t_2\}; B$	$time?t : time!t_1!t_2; i; B$
$i\{t_1..t_2\}; B$	$time!t_1!t_2; i; B$
$i\{t_1\}; B$	$time!t_1; i; B$
$gd[SP]; B$	$gd[SP]; B$
$gd\{t\ in\ t_1..t_2\}[SP]; B$	$time?t : time!t_1!t_2; gd[SP]; B$
$gd\{t_1..t_2\}[SP]; B$	$time!t_1!t_2; gd[SP]; B$
$gd\{t_1\}[SP]; B$	$time!t_1; gd[SP]; B$
$Wait(t); B$	$wait!t; B$
$P[g_1, .., g_n](..)$	$P[g_1, .., g_n, time, wait](..)$
$exit(E_1, ..., E_n)$	$exit(E_1, ..., E_n)$
$exit(E_1, ..., E_n)\{t_1..t_2\}$	$time!t_1!t_2; exit(E_1, ..., E_n)$
$exit(E_1, ..., E_n)\{t_1\}$	$time!t_1; exit(E_1, ..., E_n)$

4 Example of Use

The analysis of timed LOTOS specifications is more complex than the untimed one for many reasons. First, it is necessary to carry out extra functions as mentioned in Section 3, and second, the generated state space is almost always much bigger due to the generation of time variables in the aging process, even considering elimination of states that violate time constraints. For example, applying the Timed Parameterized Expansion to a timed version of the Bit Alternate Protocol produces 57 transitions and 33 states against 12 transitions and 9 states resulting from the application of LOLA to the equivalent untimed version.

Due to space limitations and readability we choose a timed *Bank* specification to illustrate the application of the Timed Parameterized Expand and the Testing Expand.

```
SPECIFICATION credit[bank,work,sleep,time] : noexit

LIBRARY  time, boolean  ENDLIB

 TYPE Money IS Boolean
  sorts money
 OPNS  0          :               -> money
      inc, dec : money        -> money
      _eq_     : money, money -> bool
      noi, nod : money        -> bool
 EQNS  FORALL x,y : money
 OFSORT money
      inc(dec(x))=x; dec(inc(x))=x;
      OFSORT bool
      nod(0) = true; nod(inc(x)) = nod(x);
      noi(0) = true; noi(dec(x)) = noi(x);
```

```
      0 eq 0 = true;
      inc(x) eq y = x eq dec(y);
      dec(x) eq y = x eq inc(y);
      nod(x) => 0 eq inc(x) = false;
      noi(x) => 0 eq dec(x) = false;
      ENDTYPE

  TYPE BankOperation IS
   SORTS BANKOP
   OPNS  borrow, pay : -> BankOp
  ENDTYPE

 BEHAVIOUR
  Bank[bank,sleep,time]
     (inc(inc(0 Of Money)),0 Of Money, 0 of time)
 |[bank]|
  Client[bank,work]

  WHERE
    PROCESS Client [ bank, work ] : NOEXIT :=
        bank !borrow; Client[bank,work]
     [] bank !pay   ; Client[bank,work]
     [] work        ; Client[bank,work]
    ENDPROC

    PROCESS Bank [bank,sleep,time]
             (Max,debt:Money,delay:time) : NOEXIT :=
       time!delay!4;
       bank !borrow [not(debt eq Max)];
       Bank[bank,sleep,time](Max,inc(debt),2)
     []
       time?y:time!0 of time!4;
       bank !pay    [not(debt eq 0 Of Money)] ;
       Bank[bank,sleep,time](Max,dec(debt),y)
     []
       sleep;
       Bank[bank,sleep,time](Max,debt,delay)
    ENDPROC

ENDSPEC
```

This specification represents the transactions that can happen between a client and a bank represented by their respective processes. They communicate through the gate *bank* using two kinds of transactions: *borrow* and *pay*. Observe the use of time variable *delay* in process definition and instantiations. The Bank can give money to the client in the time interval between *delay* and 4 instants of time and the client can do it consecutively only after 2 instants of time. The Bank accepts payments at any instants between 0 and 4 instants of time and records this value to be used as a delay the next time the client can borrow money. The Bank can *sleep* at any time and the Client can *work* at any time. The process Client has no time restrictions.

The expansion of this specification produces the following behaviour :

```
specification credit [testok,bank,work,sleep] : noexit

behaviour

   duplicate0 [bank,work,sleep]
          (0 of time,0 of money,inc(inc(0)))

where

process duplicate0 [bank,work,sleep]
  (delay_187:time, debt_186:money, max_185:money)
                     : noexit :=
 [delay_187 gt 4 = false]->
  bank  ! borrow {delay_187..4}
   [not(debt_186 eq max_185) = true];
  duplicate0 [bank,work,sleep]
   (2,inc(debt_186),max_185)
 []
  bank  ! pay {y_188 in 0 of time..4}
   [not(debt_186 eq 0) = true];
   duplicate0 [bank,work,sleep]
    (y_188,dec(debt_186),max_185)
 []
  sleep {0 of time..inf} ;
  duplicate0 [bank,work,sleep]
    (delay_187,debt_186,max_185)
   []
     work {time_189 in 0 of time..inf} ;
     duplicate1 [bank,work,sleep]
      (time_189,delay_187,debt_186,max_185)
endproc

process duplicate1 [bank,work,sleep]
 (time_189:time, delay_187:time,
    debt_186:money, max_185:money) : noexit :=
 [delay_187 - time_189 gt (4 - time_189) = false]->
  bank  ! borrow {delay_187 - time_189..4 - time_189}
   [not(debt_186 eq max_185) = true];
  duplicate0 [bank,work,sleep] (2,inc(debt_186),max_185)
[]
  bank! pay{y_190 in 0 of time..4 - time_189}
   [not(debt_186 eq 0) = true];
  duplicate0 [bank,work,sleep]
  (y_190 + time_189,dec(debt_186),max_185)
[]
  sleep {0 of time..inf} ;
  duplicate0 [bank,work,sleep]
  (delay_187,debt_186,max_185)
[]
  work {time_203 in 0 of time..inf} ;
  duplicate1 [bank,work,sleep]
   (time_203 + time_189,delay_187,debt_186,max_185)
endproc

endspec
```

Whenever TE-LOLA finds duplicate behaviours, it generates duplicate process instantiations. This expansion generates 8 transitions and 7 duplicated states that are collapsed into the 2 duplicated processes that appear in the results. Observing the resulting behaviour we can see that process duplicate1 has an additional parameter, a time variable, that records the passage of time in order to achieve correct aging of actions.

Free Expand and Expand can not be applied to this example since due to process recursivity they do not terminate. For testing the specification the user can define a battery of tests and execute them in batch. The tool composes them with the behaviour under test and executes one by one. For illustration purpose let us define a test for this specification that we will call BankMust. The test is a TE-LOTOS process with a success event, named *testok*, that serves to indicate successful termination.

```
PROCESS  BankMust
 [testok ,bank, work, sleep ,time]:noexit:=
time!1;
    bank !borrow ;
time!4;
    bank !borrow ;
time!2;
    bank !pay   ;
time!5;
   work;
time!6;
    sleep;
time!3;
    bank !borrow;
time!0 of time;
    testok;
STOP
ENDPROC
```

The composition and execution of this test is achieved with the command *test* followed in this case by two parameters that indicate the success event and the name of the test process. After the execution, the tool gives a response to the test accompanied by some statistics:

```
te-lola> test testok bankmust
testexpand -1 testok bankmust

 Building the rewrite rules tree.
 Building done.
 Rewriting expressions in the specification.
 Rewriting done.
 Analysing unguarded conditions.
 Analysis done.

  Composing behaviour and test :

     bankmust [testok,bank,work,sleep]
   |[bank,work,sleep]|
       bank [bank,sleep]
       (inc(inc(0)),0 of money,0 of time)
     |[bank]|
       client [bank,work]

    Analysed states       = 7
    Generated transitions = 7
    Duplicated states     = 0
    Deadlocks             = 0
    Time Deadlocks        = 0

    Process Test = bankmust
    Test result  = MUST PASS.

                   successes = 1
                       stops = 0
                       exits = 0
               cuts by depth = 0
```

5 Summary and Conclusions

A prototype tool for the analysis, transformation and verification of Time Extended LOTOS was implemented. Current version is available for SunOS and Solaris, although portability to MS-DOS is also straightforward. A graphical version, using the same user interface as X-LOLA, is available for SunOS. TE-LOLA has almost the same functionality as its predecessor LOLA and has been applied to some small case studies. Nevertheless, being a prototype, the tool has still some limitations and needs improvements that we list in the following:

1. **Input Language:** Not accepting TE-LOTOS syntax as input is the main limitation of the tool. However, this is easily solved by the use of special gates as it is done currently. Extending the front-end tools is the next step to eliminate this restriction.

2. **Time Domain:** The tool only supports discrete time domain. This was an implementation decision, since existing Natural number data types descriptions allow more readable representations (instead of usual *succ*(*succ*(..)) of time values and operations and were sufficient for the needs we had in that moment. A second reason is that the data part of LOTOS is also being evaluated at the ISO working group that proposes enhancements to the language. Thus the implementation of dense domain functions will be delayed till the definition of the new data types.

As future work we would mention the necessity to better evaluate performance aspects and apply the tool to real specifications cases. Further investigation is being carried out in order to improve current timed expansion algorithms and to define new algorithms for state space reduction.

References

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

[2] R. de Niccola and M. Hennessy. Testing Equivalences for Processes. *Journal of Theoretical Computer Science*, 34(1,2):83–133, Nov 1984.

[3] M. Diaz and R. Groz, editors. *Formal Description Techniques V*. North-Holland, 1993.

[4] ISO. LOTOS a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. IS 8807, TC97/SC21, 1989.

[5] D. Larrabeiti. *Contribución al análisis del espacio de estados de especificaciones LOTOS*. PhD thesis, Univ. Politécnica de Madrid, Febrero 1996.

[6] G. Leduc. An upward compatible time extension to LOTOS. In K.R. Parker and G.A. Rose, editors, *Formal Description Techniques IV*, pages 217–232. North-Holland, 1992.

[7] G. Leduc and L. Léonard. A Timed LOTOS Supporting a Dense Time Domain and Including new Timed Operators. In Diaz and Groz [3], pages 87–102.

[8] G. Leduc, L. Léonard, D. de Frutos, L. Llana, C. Miguel, J. Quemada, and G. Rabay. Time Extended LOTOS. In J.Quemada, editor, *Working Draft on Enhancements to LOTOS, ISO/IEC JTC1/SC21/WG1*, September 1995.

[9] C. Miguel, A. Fernandez, and L. Vidaller. Extending LOTOS Towards Performance Evaluation. In Diaz and Groz [3].

[10] F. Moller and C. Tofts. *A Temporal Calculus of Communicating Systems*, pages 401–415. Number LNCS-458, ISBN 3-540-53048-7 in Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg, New York, 1990.

[11] S. Pavón, D. Larrabeit, and G. Rabay. LOLA - LOtos LAboratory: User Manual(version 3R6). Technical report, E.T.S.I de Telecomunicación - Univ. Politécnica de Madrid, 1995.

[12] S. Pavón and M. Llamas. The Testing Functionalities of LOLA. In J. Mañas, J. Quemada, and E. Vazquez, editors, *Formal Description Techniques III*. North-Holland, 1991.

[13] J. Quemada, A. Azcorra, and D. Frutos. A Timed Calculus for LOTOS. In S. T. Vuong, editor, *Formal Description Techniques II*. North-Holland, 1990.

[14] J. Quemada, A. Azcorra, and D. Frutos. TIC: A TImed Calculus. *Formal Aspects of Computing*, (5:224-252), June 1993.

[15] J. Quemada and A. Fernández. Introduction of Quantitative Relative Time into Lotos. In H. Rudin, editor, *Protocol Specification, Testing and Verification VII*, Amsterdam, May 1987. North–Holland.

[16] J. Quemada, D. Larrabeiti, and S. Pavón. Compressed State Space Representation of LOTOS Specifications. In R.L. Tenney, P.D. Amer, and M. Uyar, editors, *Formal Description Techniques, VI*, pages 19 – 34. North-Holland, October 1994.

[17] J. Quemada, S. Pavón, and A. Fernández. State Exploration by Transformation with LOLA. In *Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, June 1989.

[18] J. Quemada, S. Pavón, and A. Fernández. Transforming LOTOS Specifications with LOLA: The Parameterized Expansion. In K. Turner, editor, *Formal Description Techniques I*. North-Holland, 1989.

[19] G. Rabay and L. Llana. Defining semantic equivalences between time/actions trees and timed-action trees. Technical report, Depto. Informática y Automática, Universidad Complutense de Madrid, November 1995.