16

Verifying ET-LOTOS programs with KRONOS *

C. Daws, A. Olivero and S. Yovine [a] † ‡

[a]VERIMAG, Miniparc-Zirst, Rue Lavoisier - 38330 Montbonnot St. Martin, France

ET-LOTOS is a timed extension of LOTOS proposed for modeling real-time systems. KRONOS is a tool that checks whether an automaton extended with clocks (called timed automaton) satisfies a real-time requirement expressed as a formula of the logic TCTL. This paper shows that real-time systems described in a reasonable subset of ET-LOTOS can be verified with KRONOS by compiling them into timed automata. We illustrate the practical interest of our approach with a case study: the Tick-Tock protocol.
Keywords: I.1 / II.2: ET-LOTOS, I.3, I.6 / III.1: Timed Automata / III.2: TCTL, I.7 / I.8: Tick-Tock protocol, I.10: Real-time

1. Introduction

The development of services and protocols requires timing mechanisms such as timeouts, watchdogs, delays and isochronous data transfers. In the last few years, many FDTs with constructs to model the above mechanisms have been proposed. These "timed" FDTs are mainly based on extensions of asynchronous process algebras, see [11, 7] for an overview. FDTs need supporting tools to allow automatic verification of requirements.

ET-LOTOS [8] is a timed extension of LOTOS that allows the modeling of real-time behaviors. KRONOS [12, 10] is a tool that checks whether a real-time system described by a timed automaton (automaton extended with clocks) satisfies a requirement expressed as a formula of the logic TCTL [1].

The need of verification methods and tools for ET-LOTOS has been stated in [8]. The main issue of this work is to show that real-time systems described in ET-LOTOS can be verified with the tool KRONOS by compiling them into timed automata. A similar approach has also been successfully applied to the algebra of timed processes ATP [12]. To illustrate the practical interest of our approach we apply it to a case study: the Tick-Tock protocol.

This paper is organized as follows. In section 2, we review the description of the Tick-Tock protocol. The ET-LOTOS specification of the protocol is given in section 3. We also review here the syntax and semantics of ET-LOTOS. Section 4 is devoted to timed automata. We illustrate the main ideas of the translation method of ET-LOTOS into timed automata with the example of the protocol. In section 5 we formalize the translation method. In section 6 we use the tool KRONOS to check that the ET-LOTOS description of the protocol satisfies the timing requirements.

*This work has been partially supported by ESPRIT-BRA REACT-P 6021.
†E-mail: {Conrado.Daws,Alfredo.Olivero,Sergio.Yovine}@imag.fr
‡VERIMAG is a joint laboratory of CNRS, INPG, Université Joseph Fourier, and Verilog SA, associated with IMAG.

D. Hogrefe et al. (eds.), *Formal Description Techniques VII*

2. A case study: the Tick-Tock protocol

The Tick-Tock protocol [9] is a case study for the assessment of timed FDTs. Although this protocol intends to be a realistic system, all details that are not relevant to timing requirements are simplified. However, it exhibits most of the essential timing constraints that need to be described by timed FDTs.

The Tick-Tock protocol is composed of three entities called *sender*, *receiver* and *service*. *Service* interacts with *sender* and *receiver* through their SAPs, respectively called Ss-SAP and Sr-SAP. Figure 1 shows a sketch of the system. To keep things simple, we restrict ourselves to the specification of *service*.

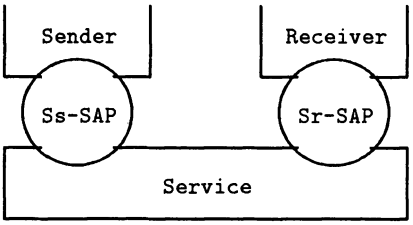

Figure 1. Sketch of the protocol.

Description of service

Service transmits data from *sender* to *receiver*. Exchanges through the corresponding SAPs are instantaneous and atomic, and carry a data *cell* as parameter. *Service* satisfies the following timing requirements.

Isochronism. The behavior of *service* is *isochronous*, that is, a cell from *sender* is only accepted at precise, punctual instants with a given period of π time units. *Sender* may neglect an opportunity of emission. Just one cell can be exchanged at any instant.

Transmission delays. *Service* always proposes a cell to *receiver* between τ_{min} and τ_{max} time units after its emission.

Spacing between deliveries. There is always a delay of at least α time units between two successive offers of cells at Sr-SAP.

Immediate acceptance. A cell offered by *service* to *receiver* must be immediately accepted by *receiver*, otherwise the *service* losses the cell immediately.

Loss-free transmission. The previous requirement describes the only way a cell received by *service* from *sender* can be lost: no cell is lost during its transit through *service*.

3. Specification in ET-LOTOS

ET-LOTOS [8] is an extension of LOTOS that allows the specification of timing requirements. In order to illustrate the meaning of the temporal operators of ET-LOTOS

we first specify the behavior of *service*. We then recall the semantics of ET-LOTOS which is given in terms of labeled transition systems. The formal semantics of full ET-LOTOS, as well as its compatibility with respect to LOTOS, is given in [8].

3.1. Specification of Service

Our specification is basically the one given in [8] with slight changes. The specification of *service* is given in a constrained-oriented style, that is, each requirement is given as an ET-LOTOS process, and the overall behavior of the system corresponds to the parallel composition of all of them. Internal synchronizations are hidden using the `hide` operator.

Figure 2 shows the general structure of *service*. It is composed of four processes, namely, `Isochronous`, `TransDels`, `SpacingDeliveries` and `ImmAccept`.

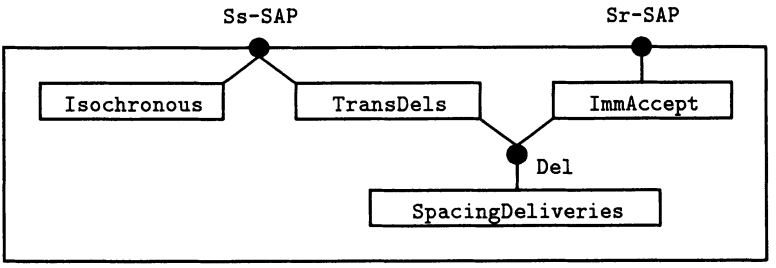

Figure 2. Structure of *service*.

Notice that `TransDels`, `SpacingDeliveries` and `ImmAccept` synchronize on the internal gate `Del`. The gate `Del` becomes enabled only when *service* is ready to deliver a cell to *receiver*. In this case, the cell must be transmitted immediately or lost.

Isochronism. The isochronous behavior of *service* is:

```
process Isochronous [Ss-SAP] : noexit :=
Ss-SAP{0}; Δ^π Isochronous
[]
Δ^π Isochronous
endproc
```

Process `Isochronous` is a choice between two possible behaviors, and this choice is resolved immediately. The action-prefix `Ss-SAP{0}` means that the offer `Ss-SAP` of *service* to *sender* has a zero duration, that is, either the *sender* is ready to interact at this punctual instant and the action occurs, or the offer is removed without executing the subsequent behavior. So, either the offer is accepted immediately, or the process behaves like `stop`. In the first case, a new occurrence of `Isochronous` is called after a delay π, which is expressed by the operator Δ^{π}. In the second case, a new communication through `Ss-SAP` will be offered after a delay of π units.

This example illustrates the two basic timed operators of ET-LOTOS, namely, the extended prefixing g{t}, called *life reducer*, and the *delay* operator Δ^t. The expression g{t} means that g is only offered during the interval [0,t], that is, if g has not occurred after a delay t, the offer is removed and the process behaves like stop. It is important to note that the attribute {t} *does not force* g to be executed before t. A main assumption is that observable actions, i.e., actions different from i, may occur at any time: that is, if no time attribute {t} is associated with an offer g, the execution of g may be delayed without bound. The delay operator Δ^t expresses that the subsequent behavior will be delayed by t. We write g{min,max} as a shorthand notation for $\Delta^{\texttt{min}}$ g{max-min}.

Transmission delays. This requirement is expressed by:

```
process TransDels [Ss-SAP,Del] :  noexit :=
Ss-SAP; i{τmin, τmax}; Del; stop
|||
TransDels [Ss-SAP, Del]
endproc
```

An important property of the action i is that it *must* occur within the interval associated with it. By default, when no time parameter is present, we assume that it is equal to 0, and we say that i is *urgent*. So, TransDels introduces a minimum delay τ_{min} after the reception of a cell through Ss-SAP and an additional and non-deterministic delay less than or equal to $\tau_{max} - \tau_{min}$. Since i must occur, the offer Del is enabled within the interval $[\tau_{min}, \tau_{max}]$. However, Del may not be executed immediately because of the minimal delay imposed between two consecutive deliveries. We will see later in section 6 that under some assumptions on the parameters, the Del will never occur after τ_{max}.

Spacing between deliveries. The following ET-LOTOS process ensures that two successive offers of Del are spaced out at least by α time units:

```
process SpacingDeliveries [Del] :  noexit :=
Del; Δ^α SpacingDeliveries
endproc
```

Immediate acceptance. This requirement is specified as follows:

```
process ImmAccept [Del,Sr-SAP] : noexit :=
Del;
( Sr-SAP{0}; ImmAccept[Del,Sr-SAP]
[]
ImmAccept[Del,Sr-SAP] )
endproc
```

The behavior of ImmAccept is as follows: either it outputs the cell immediately through Sr-SAP, or if the output is not possible, it looses the cell and waits for the next one.

Service. The above processes synchronize on the internal gate Del. As Del is an internal action, it must be hidden. Process Service is defined as follows:

```
process Service [Ss-SAP] : noexit :=
```

```
( Isochronous [Ss-SAP]
|[ Ss-SAP ]|
( hide Del in
( TransDels [Ss-SAP,Del]
|[ Del ]|
SpacingDeliveries [Del]
|[ Del ]|
ImmAccept [Del,Sr-SAP] ) ) )
endproc
```

In ET-LOTOS, hidden actions must occur as soon as they become enabled. This behavior is ensured by the semantics of the `hide` operator. However, notice that even if `Del` is forced to occur as soon as possible, it is not clear whether it will always happen within the required interval of time. In section 6 we use the tool KRONOS to prove that this requirement is fulfilled.

3.2. Semantics of ET-LOTOS

The formal meaning of ET-LOTOS is given in terms of labeled transition systems. There are two kind of transitions, namely discrete and timed. A discrete transition corresponds to the execution of an action, and is therefore labeled with the name of the action. $P \xrightarrow{a} P'$ means that P may engage in action a and then behave like P'. Timed transitions correspond to time elapses. The set of possible timed values is called the *time domain.* If D is the time domain, $+$ the addition in D, 0 the neutral element of D, then $(D, +, 0)$ is a commutative monoid, $(D, \leq)$ is a total order and ∞ is the absorbent element for $+$. D can be discrete or dense. Hereafter we assume that D is the set of rational numbers. Now, $P \xrightarrow{d} P'$ means that P may idle during a time d and then behave like P'.

Table 1 shows the formal operational semantics of a subset of ET-LOTOS. We restrict ourselves to the subset containing all the constructs used in the example modeled above. The semantics of full ET-LOTOS is given in [8]. The semantics is given as a set of axioms and inference rules, with $d \in D - \{0\}$, $d_1 \in D$, $a \in G \cup \{\delta, \mathtt{i}\}$ and $\Gamma \subseteq G$, where G is the set of observable actions. We have also $a \neq \delta$ in (AP1), (AP2), (AP3). The left and right columns correspond to discrete and timed transitions, respectively.

4. Extended automata with clocks

In this section we model timing requirements by automata extended with a set of *clocks.* Clocks are variables that count the time elapsed between events. These automata are called *timed automata.* Many different definitions of timed automata have been given, see for instance [1, 12, 13]. Here we adopt the definition given in [10].

A timed automaton has a finite set of clocks. The values of the clocks increase uniformly with time. Timing constraints are expressed by associating enabling conditions with transitions. A transition is enabled, that is, may be executed, if the condition is satisfied by the current values of the clocks. A clock can be reset with any transition. At any instant the value of a clock is equal to the time elapsed since the last time it was reset.

The semantics of timed automata and ET-LOTOS is given in terms of the same class of labeled transition systems. Besides, the parallel composition operator of ET-LOTOS can also be defined on timed automata.

Table 1
Operational semantics of ET-LOTOS.

		(S)	$\texttt{stop} \xrightarrow{d} \texttt{stop}$
(AP1)	$a\{d_1\};P \xrightarrow{a} P$	(AP2)	$a\{d_1+d\};P \xrightarrow{d} a\{d_1\};P$
		(AP3)	$a\{d_1\};P \xrightarrow{d} \texttt{stop} \quad (a \neq i, d > d_1)$
(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)	$\dfrac{P \xrightarrow{d} P'}{\Delta^0 P \xrightarrow{d} P'}$
(CH1)	$\dfrac{P \xrightarrow{a} P'}{P[]Q \xrightarrow{a} P'}$ (+CH1')	(CH2)	$\dfrac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{P[]Q \xrightarrow{d} P'[]Q'}$
(P1)	$\dfrac{P \xrightarrow{a} P'}{P\|[\Gamma]\|Q \xrightarrow{a} P'\|[\Gamma]\|Q} \quad (a \notin \Gamma \cup \{\delta\})$ (+P1')	(P3)	$\dfrac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{P\|[\Gamma]\|Q \xrightarrow{d} P'\|[\Gamma]\|Q'}$
(P2)	$\dfrac{P \xrightarrow{a} P', Q \xrightarrow{a} Q'}{P\|[\Gamma]\|Q \xrightarrow{a} P'\|[\Gamma]\|Q'} \quad (a \in \Gamma \cup \{\delta\})$		
(H1)	$\dfrac{P \xrightarrow{a} P'}{\texttt{hide}\Gamma\texttt{in}P \xrightarrow{a} \texttt{hide}\Gamma\texttt{in}P'} \quad (a \notin \Gamma)$	(H3)	$\dfrac{P \xrightarrow{d} P', \forall a \in \Gamma.P \xrightarrow{a}\!\!\!\!\!\!/}{\texttt{hide}\Gamma\texttt{in}P \xrightarrow{d} \texttt{hide}\Gamma\texttt{in}P'}$
(H2)	$\dfrac{P \xrightarrow{a} P'}{\texttt{hide}\Gamma\texttt{in}P \xrightarrow{i} \texttt{hide}\Gamma\texttt{in}P'} \quad (a \in \Gamma)$		
(In1)	$\dfrac{[g/h]P \xrightarrow{a} P', Q[h] := P}{Q[g] \xrightarrow{a} P'}$	(In2)	$\dfrac{[g/h]P \xrightarrow{d} P', Q[h] := P}{Q[g] \xrightarrow{d} P'}$

4.1. Specification of Service

We specify the behavior of *service* to illustrate the syntax and semantics of timed automata. For each ET-LOTOS process we show a timed automaton that describes the same behavior. *Service* is modeled as the parallel composition of the following timed automata.

Isochronism. The timed automaton of process `Isochronous` is shown in figure 3. This automaton has two control *locations*, namely 0 and 1, and one clock, namely x. The initial location is 0 and the initial value of x is 0, which is denoted by the small arrow. At location 0, the automaton behaves like follows: either it takes the transition `Ss-SAP` when the value of x is equal to 0, or it waits until the value of x is π to reset x to 0 in order to allow a new offer of `Ss-SAP`. The label Δ means that the transition corresponds to the delay operator of ET-LOTOS. At location 1, the automaton waits π time units and then moves to location 0.

In both locations the value of x *cannot increase beyond* π, which forces the transitions to be taken when the value of x reaches π. Consequently, the time spent in one visit at each location is less or equal than π. This behavior is ensured by associating with each

location a predicate called the *time-can-progress* predicate, noted **tcp**, that determines the amount of time the automaton may stay at the location. The **tcp** predicate is formally defined in section 4.2.

Transmission delays. Recall the definition of process `TransDels` given before. Clearly, this process cannot be represented by a finite automaton since the use of recursion through the parallel operator generates an unbounded number of instances. Instead, we consider the following definition of process `TransDels`:

```
process TransDels [Ss-SAP,Del] :  noexit :=
Ss-SAP; i{τmin, τmax}; Del; TransDels [Ss-SAP, Del]
endproc
```

Remark 4.1 *Notice that in this case,* service *has no capacity to buffer cells and it can only accept one cell at a time. If buffering of cells is required, we can put in parallel as many of these processes as the size of the buffer.*

Figure 4 shows the timed automaton corresponding to process `TransDels`. Location 0 is the initial location with z equal to 0. The system may rest at locations 0 and 2 a time without bound, but it must leave location 1 at some time within the interval $[\tau_{min}, \tau_{max}]$. We assume that the **tcp** predicate ensures that the value of z cannot increase beyond τ_{max} at location 1.

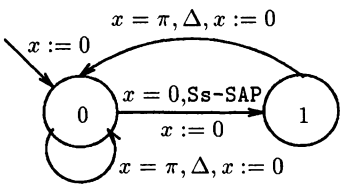

Figure 3. Isochronism.

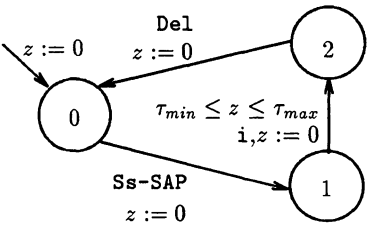

Figure 4. Transmission delays.

Spacing between deliveries. The timed automaton of figure 5 describes the same behavior than the ET-LOTOS process `SpacingDeliveries`. At location 0, the system may take the transition `Del` at any time. Location 1 models the delay of α required between two consecutive offers of `Del`. At this location, the **tcp** predicate ensures that the value of y cannot increase beyond α, so the transition must be taken when the value of y reaches α. The Δ-transition models the expiring delay.

Immediate acceptance. Figure 6 shows the timed automaton of `ImmAccept`. After a transition `Del`, the value of w is reset to 0 and a `Sr-SAP` is offered. Since the enabling condition of the transition `Sr-SAP` is $w = 0$, either it is taken immediately, or it becomes disabled and the opportunity is lost until the next `Del`. The `tcp` predicate is true in both locations.

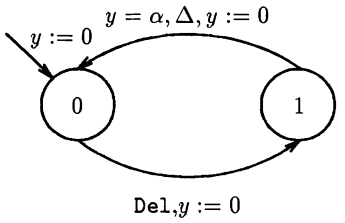

Figure 5. Spacing between deliveries.

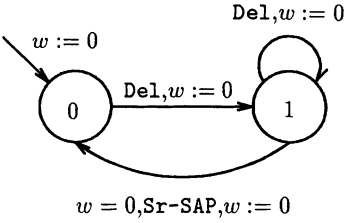

Figure 6. Immediate acceptance.

4.2. Operational semantics

Formally, a timed automaton is a tuple $T = \langle S, X, E, s^0, \mathsf{tcp} \rangle$ where:

1. S is a finite set of vertices called locations.

2. X is a finite set of variables called clocks. A valuation v of clocks is a function that assigns a value $v(x) \in D$ to each clock $x \in X$. V denotes the set of valuations.

   A *state* q is a pair (s, v) consisting of a location $s \in S$ and a valuation $v \in V$. We write $\mathcal{Q}$ for the set of states.

3. E is a finite set of edges called transitions. Each transition $e = (s, l, \psi, \rho, s')$ consists of a source location $s \in S$, a target location $s' \in S$, a label l, a guard ψ, and a set $\rho \subseteq X$ of clocks to be reset to zero simultaneously with the transition.

   A guard is any boolean combination of atoms of the form $x \# c$ where $x \in X$, $c \in \mathsf{N}$ and $\#$ is any binary relation in the set $\{<, \leq, >, \geq, =\}$.

   The transition e is *enabled* at a state (s, v) if v satisfies the guard ψ. We write $v[\rho := 0]$ for the valuation v' such that $v'(x) = 0$ for $x \in \rho$ and $v'(x) = v(x)$ otherwise. The state $(s', v[\rho := 0])$ is the discrete successor of state (s, v) by transition e.

4. s^0 is the initial location.

5. For each $s \in S$, `tcp` is the *time-can-progress* predicate that determines for each state (s, v) and time value $d \in D$, if the system may let time progress by an amount of d at location s. In this case, the values of the clocks increase by d.

   We write $v + d$ for the valuation v' such that $v'(x) = v(x) + d$ for all clocks $x \in X$. The state $(s, v + d)$ is the timed successor of state (s, v).

The operational semantics of a timed automaton is a labeled transition system where the set of states is $\mathcal{Q}$ and the transition relation $\rightarrow$ is defined by the rules in table 2.

Table 2
Operational semantics of timed automata.

(T1)	$\dfrac{e = \langle s, l, \psi, \rho, s' \rangle \in E,\ \psi(v)}{(s, v) \xrightarrow{l} (s', v[\rho := 0])}$	(T2)	$\dfrac{\mathsf{tcp}(s)(v, d), d \in D - \{0\}}{(s, v) \xrightarrow{d} (s, v + d)}$

Note For instance, the tcp predicate of location 0 of the timed automaton of figure 3 is $\mathsf{tcp}(0)(v, d) \equiv v(x) + d \leq \pi$. From this definition and rule (T2) above it is easy to see that the amount of time the automaton may stay at location 0 is less than or equal to π. For a detailed description of the tcp predicate see [10].

5. The principle of the translation method

The example above suggests a general method for translating ET-LOTOS specifications into timed automata. In this section we show the principle of this method for the subset of ET-LOTOS considered in section 3. Our translation method is based on the one developed for the timed process algebra ATP presented in [12, 13]. We use the label Δ to represent transitions corresponding to expiring events of the delay operator, l denotes an element of $G \cup \{\delta, \mathtt{i}, \Delta\}$.

Stop. The timed automaton corresponding to the process `stop` is

$$\langle \{s\}, \{x\}, \emptyset, s, \mathsf{tcp} \rangle$$

where $\mathsf{tcp}(s)$ is the predicate *true*, that is, $\mathsf{tcp}(s)(v, d)$ is true for all valuations v of x and for all time values $d \in D$. This means that `stop` is a process that only lets time pass (rule (S) in table 1).

Prefixing. Let P be the term $a\{d_1\}; P_1$ and $T_1 = \langle S_1, X_1, E_1, s_1^0, \mathsf{tcp}_1 \rangle$ be the timed automaton of P_1. The timed automaton of P is

$$\langle S_1 \cup \{s\}, X_1 \cup \{x\}, E_1 \cup \{(s, a, x \leq d_1, X_1, s_1^0)\}, s, \mathsf{tcp} \rangle$$

where the initial location $s \notin S_1$ is a new one, and $x \notin X_1$ is a new clock, and the new transition represents that the gate a is enabled for at most d_1 time units (rules (AP1) and (AP2) in table 1).

For all locations $s_1 \in S_1$, $\mathsf{tcp}(s_1)$ is like $\mathsf{tcp}_1(s_1)$. For the initial location $s \in S$, $\mathsf{tcp}(s)$ depends on whether a is an observable action or $a = \mathtt{i}$. If $a \neq \mathtt{i}$, then $\mathsf{tcp}(s)$ is *true* which means that time can progress without bound at the initial location s (rule (AP3) in table 1). If $a = \mathtt{i}$, then $\mathsf{tcp}(s)(v, d)$ is true iff $v(x) + d \leq d_1$, which means that the value of x cannot increase beyond d_1, expressing the necessity for the transition to be taken within the interval $[0, d_1]$.

Delay. Let P be the term $\Delta^{d_1} P_1$ and $T_1 = \langle S_1, X_1, E_1, s_1^0, \mathsf{tcp}_1 \rangle$ be the timed automaton of P_1. The timed automaton of P is

$$\langle S_1 \cup \{s\}, X_1 \cup \{x\}, E_1 \cup \{(s, \Delta, x = d_1, X_1, s_1^0)\}, s, \mathsf{tcp} \rangle$$

where the initial location $s \notin S_1$ is a new one, $x \notin X_1$ is a new clock, and the new transition represents that the system moves to the initial state of T_1 after idling d_1 time units at the initial location s. For all locations $s_1 \in S_1$, $\mathsf{tcp}(s_1)$ is $\mathsf{tcp}_1(s_1)$. For the initial location s, $\mathsf{tcp}(s)(v, d)$ is true iff $v(x) + d \leq d_1$, which means that the value of x cannot increase beyond d_1, forcing the transition Δ to be executed when the value of x is equal to d_1.

Nondeterministic choice. Let P be the term $P_1 [] P_2$ and $T_i = \langle S_i, X_i, E_i, s_i^0, \mathsf{tcp}_i \rangle$ be the timed automaton of P_i, for $i \in \{1, 2\}$. The timed automaton of P is

$$\langle S_1 \cup S_2 \cup S_1 \times S_2, X_1 \cup X_2, E \cup E_1 \cup E_2, (s_1^0, s_2^0), \mathsf{tcp} \rangle$$

where the set of transitions E is

$$\begin{aligned}
E \;=\; & \{((s_1, s_2), a, \psi, \rho, s_1') \mid (s_1, a, \psi, \rho, s_1') \in E_1\} && (1)\\
& \cup \{((s_1, s_2), a, \psi, \rho, s_2') \mid (s_2, a, \psi, \rho, s_2') \in E_2\} && (2)\\
& \cup \{((s_1, s_2), \Delta, \psi, \rho, (s_1', s_2)) \mid (s_1, \Delta, \psi, \rho, s_1') \in E_1\} && (3)\\
& \cup \{((s_1, s_2), \Delta, \psi, \rho, (s_1, s_2')) \mid (s_2, \Delta, \psi, \rho, s_2') \in E_2\} && (4)
\end{aligned}$$

(1) and (2) express that a choice is resolved by the execution of an action (rules (CH1) and (CH1') in table 1), and (3) and (4) express that the expiration of a delay does not resolve a choice (rule (CH2) in table 1). For all locations $s_i \in S_i$, $i = 1, 2$, $\mathsf{tcp}(s_i)$ is $\mathsf{tcp}_i(s_i)$. For all locations $(s_1, s_2) \in S_1 \times S_2$, $\mathsf{tcp}(s_1, s_2)$ is $\mathsf{tcp}_1(s_1) \wedge \mathsf{tcp}_2(s_2)$ which means that time passes at the same pace in both processes (rule (CH2) in table 1).

Parallel composition. Let P be the term $P_1 |[\Gamma]| P_2$ and $T_i = \langle S_i, X_i, E_i, s_i^0, \mathsf{tcp}_i \rangle$ be the timed automaton of P_i, for $i \in \{1, 2\}$. The timed automaton of P is

$$\langle S_1 \times S_2, X_1 \cup X_2, E, (s_1^0, s_2^0), \mathsf{tcp} \rangle$$

where the set of transitions E is

$$\begin{aligned}
E \;=\; & \{((s_1, s_2), l, \psi, \rho, (s_1', s_2)) \mid (s_1, l, \psi, \rho, s_1') \in E_1, l \notin \Gamma \cup \{\delta\}\} && (5)\\
& \cup \{((s_1, s_2), l, \psi, \rho, (s_1, s_2')) \mid (s_2, l, \psi, \rho, s_2') \in E_2, l \notin \Gamma \cup \{\delta\}\} && (6)\\
& \cup \{((s_1, s_2), l, \psi_1 \wedge \psi_2, \rho_1 \cup \rho_2, (s_1', s_2')) \mid (s_i, l, \psi_i, \rho_i, s_i') \in E_i, i = 1, 2, l \in \Gamma \cup \{\delta\}\} && (7)
\end{aligned}$$

(5), (6) and (7) correspond to rules (P1), (P1') and (P2) in table 1, respectively. For all locations $(s_1, s_2) \in S_1 \times S_2$, $\mathsf{tcp}(s_1, s_2)$ is $\mathsf{tcp}_1(s_1) \wedge \mathsf{tcp}_2(s_2)$ which means that time passes at the same pace in both processes (rule (P3) in table 1).

Hide. Let P be the term $\texttt{hide}\Gamma\texttt{in}P_1$ and $T_1 = \langle S_1, X_1, E_1, s_1^0, \mathsf{tcp}_1\rangle$ be the timed automaton of P_1. The timed automaton of P is

$$\langle S_1, X_1, E, s_1^0, \mathsf{tcp}\rangle$$

where the set of transitions E is

$$E = \{(s_1, l, \psi, \rho, s_1') \mid (s_1, l, \psi, \rho, s_1') \in E_1, l \notin \Gamma\} \quad (8)$$

$$\cup\{(s_1, \mathtt{i}, \psi, \rho, s_1') \mid (s_1, l, \psi, \rho, s_1') \in E_1, l \in \Gamma\} \quad (9)$$

(8) and (9) correspond to rules (H1) and (H2) in table 1. For all locations $s \in S_1$, $\mathsf{tcp}(s)(v, d)$ is the predicate $\mathsf{tcp}_1(s)(v, d) \wedge \forall l \in \Gamma.\neg\mathsf{enable}(s, l)(v)$ where $\mathsf{enable}(s, l)(v)$ is true if an outgoing transition from s with label l is enabled for the valuation v. This definition of $\mathsf{tcp}(s)$ captures exactly the meaning of rule (H3) in table 1.

The syntax-driven construction presented in this section introduces Δ-transitions to represent expirations of delays. These transitions, which are also used in the translation method of ATP [12, 13], play a role similar to the ϵ-transitions used in [4] and are introduced in order to obtain a compositional method of construction. In many cases these transitions may be eliminated by propagating the corresponding timing constraints. The ET-LOTOS program and the automaton have the same behaviors. The proof of correctness is similar to the one given for ATP in [14]. It relies on the definition of an equivalence relation between transition systems that appropriately handles Δ-transitions introduced by the translation method. To keep the presentation simple we do not deal with recursion here. For a formal treatment of recursion see for instance [4, 14].

It is important to note that the complexity of the algorithm is independent of the values of the parameters of the timing constructs of ET-LOTOS.

6. Verifying timing requirements with KRONOS

We address here the problem of verifying whether a given ET-LOTOS specification satisfies a requirement that is expressed in the real-time temporal logic TCTL [1]. In order to do so, we first apply the translation method proposed in section 5 to obtain the timed automaton corresponding to the ET-LOTOS process. Then we use the model-checker KRONOS [12, 10] which checks the validity of the formula on the timed automaton.

6.1. The logic TCTL

The real-time temporal logic TCTL extends the temporal operators $\exists\mathcal{U}$ (*there exists a run*) and $\forall\mathcal{U}$ (*for all runs*) of CTL [3] with timing constraints that allow "quantitative" temporal reasoning. Let $\mathcal{P}$ be a set of basic predicates. The formulas of TCTL are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \exists\mathcal{U}_I \varphi_2 \mid \varphi_1 \forall\mathcal{U}_I \varphi_2$$

where $p \in \mathcal{P}$ is a basic predicate and I is an interval with positive integer end-points (I may be open, closed, bounded or unbounded).

Intuitively, $\varphi_1 \exists\mathcal{U}_I \varphi_2$ means that there exists a run of the system with a finite prefix such that φ_2 holds in the last state at a time t within the interval I and φ_1 continuously holds in all the previous states. $\varphi_1 \forall\mathcal{U}_I \varphi_2$ means that for all runs the above property holds.

For example, we can write $true\exists\mathcal{U}_{<5}p$ to say that along some run, proposition p becomes true before 5 time units.

We will use the typical abbreviations such as $\forall\Diamond_I\varphi$ for $true\forall\mathcal{U}_I\varphi$, $\exists\Diamond_I\varphi$ for $true\exists\mathcal{U}_I\varphi$, $\exists\Box_I\varphi$ for $\neg\forall\Diamond_I\neg\varphi$ and $\forall\Box_I\varphi$ for $\neg\exists\Diamond_I\neg\varphi$. The unrestricted temporal operators of CTL correspond to TCTL operators subscripted by ≥ 0.

The set $\mathcal{P}$ of basic predicates is defined by the following grammar:

$$p \ ::= \ \mathsf{init} \mid \mathsf{enable}(a) \mid \mathsf{after}(a) \mid x \in I$$

where $x \in X$ is a clock of the timed automaton we want to verify, **init** defines the initial state (i.e. the initial location with all clocks set to 0) and for any action a, **enable**(a) defines the set of states having an enabled transition labeled a and **after**(a) defines the set of states reached by a transition labeled a.

6.2. The tool KRONOS

Given a timed automaton T and a TCTL-formula φ, the model-checker KRONOS computes the set of states of the labeled transition system of T that satisfies the formula φ. This set is called the characteristic set of φ and it is denoted $[\![\varphi]\!]$. KRONOS implements the symbolic model-checking algorithm developed in [5]. This algorithm is based on fixpoint characterizations of the operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$ in terms of a binary next operator $\triangleright$. In [5] it is shown that the fixpoints can be computed iteratively and the procedure always terminates. Here, we show how KRONOS iteratively computes the characteristic set of some of the TCTL-formulas.

Reachability. The set of states from where it is possible to reach a state satisfying the formula φ within the interval I is expressed in TCTL by the formula $\exists\Diamond_I\varphi$. The characteristic set of this formula is iteratively computed as $\bigcup_i Y_i[z := 0]$ with

$$\begin{aligned} Y_0 &= [\![\varphi \wedge z \in I]\!] \\ Y_{i+1} &= Y_i \cup (true \triangleright Y_i) \end{aligned}$$

The variable z is a new clock added to count the time elapsed since the beginning of the run. So, the algorithm starts with the set of states that satisfy $\varphi \wedge z \in I$ and computes the predecessors and so on. At the end, the value of z is reset to 0.

Invariance. To verify if the formula φ is an invariant of the system we check if the initial state is contained in the characteristic set of the formula $\forall\Box\varphi$. This characteristic set can be iteratively computed as $\bigcap_i Y_i$ with

$$\begin{aligned} Y_0 &= [\![\varphi]\!] \\ Y_{i+1} &= Y_i \cap \neg(true \triangleright \neg Y_i) \end{aligned}$$

Bounded-time response. The real-time response property asserting that a given event must occur within a certain time bound c is expressed in TCTL by a formula of the form $\forall\Diamond_{\leq c}\varphi$. It is shown in [5] that this formula is equivalent to $\neg(\neg\varphi\exists\mathcal{U}_{>c}true)$, that is, φ must occur within c time units if there exists no run for which the negation of φ continuously

holds for a time greater than c. The characteristic set can be iteratively computed as $\neg \bigcup_i Y_i[z := 0]$ with

$$\begin{aligned} Y_0 &= [\![z > c]\!] \\ Y_{i+1} &= Y_i \cup ([\![\neg\varphi]\!] \triangleright Y_i) \end{aligned}$$

where the variable z is a new clock added to count the time elapsed since the beginning of the run.

6.3. Requirements of Service

Isochronism. This requirement states that offers at Ss-SAP occur at punctual instants with a period π. Isochronism is specified by a set of formulas. The first formula

$$\mathsf{init} \Rightarrow \forall\Box(\mathsf{enable(Ss\text{-}SAP)} \Rightarrow \forall\Diamond_{=\pi}\mathsf{enable(Ss\text{-}SAP)}) \tag{10}$$

states that always an offer at Ss-SAP is inevitably followed by another offer exactly π time units later. The second formula

$$\mathsf{init} \Rightarrow \forall\Box(\mathsf{enable(Ss\text{-}SAP)} \Rightarrow \neg\exists\Diamond_{(0,\pi)}\mathsf{enable(Ss\text{-}SAP)}) \tag{11}$$

states that an offer at Ss-SAP will never follow the previous one before a time π. The third formula

$$\neg(\mathsf{enable(Ss\text{-}SAP)}\exists\mathcal{U}_{>0}\mathsf{enable(Ss\text{-}SAP)}) \tag{12}$$

says that there exists no run where enable(Ss-SAP) holds for some time greater than zero, or equivalently, offers at Ss-SAP have zero duration.

Transmission delays. This requirement is specified by the following formula:

$$\mathsf{init} \Rightarrow \forall\Box(\mathsf{after(Ss\text{-}SAP)} \Rightarrow \forall\Diamond_{[\tau_{min},\tau_{max}]}\mathsf{enable(Sr\text{-}SAP)}) \tag{13}$$

that is, always an offer at Sr-SAP must occur within the interval $[\tau_{min}, \tau_{max}]$ after its reception from *sender*.

Spacing between deliveries. There is always a delay of at least α time units between two successive offers of cells at Sr-SAP. The formula

$$\mathsf{init} \Rightarrow \forall\Box(\mathsf{enable(Sr\text{-}SAP)} \Rightarrow \neg\exists\Diamond_{(0,\alpha)}\mathsf{enable(Sr\text{-}SAP)}) \tag{14}$$

states that there exists no run starting at a state satisfying after(Sr-SAP) where an offer at Sr-SAP is enabled before a time α.

Immediate acceptance. A cell offered by *service* to *receiver* must be immediately accepted or lost. The formula

$$\neg(\mathsf{enable(Sr\text{-}SAP)}\exists\mathcal{U}_{>0}\mathsf{enable(Sr\text{-}SAP)}) \tag{15}$$

says that offers at Sr-SAP have zero duration.

Table 3
Sizes.

buffer size	trans	states	clocks
1	64	24	5
2	240	72	6
3	864	216	7

Table 4
Results for a buffer of size 1.

	A			B			C			D		
	$\tau_{min} = 50\ \alpha = 90$			$\tau_{min} = 10500\ \alpha = 12000$			$\tau_{min} = 13\ \alpha = 1050$			$\tau_{min} = 75\ \alpha = 50$		
	$\tau_{max} = 80\ \pi = 100$			$\tau_{max} = 13666\ \pi = 15000$			$\tau_{max} = 813\ \pi = 1000$			$\tau_{max} = 120\ \pi = 100$		
	iter	time	eval	iter	time	eval	iter	time	eval	iter	time	eval
10	8	3.61 s	true	8	3.66 s	true	36	5.83 s	false	16	6.75 s	false
11	7	0.53 s	true	7	0.51 s	true	6	0.33 s	true	7	0.63 s	true
12	1	0.03 s	true	1	0.03 s	true	1	0.05 s	true	1	0.06 s	true
13	12	4.46 s	true	8	6.05 s	true	19	4.96 s	false	9	9.86 s	true
14	7	1.81 s	true	7	1.56 s	true	8	2.25 s	true	6	1.56 s	true
15	1	0.15 s	true	1	0.16 s	true	1	0.13 s	true	1	0.16 s	true

We show here the results obtained with the tool KRONOS to verify whether the ET-LOTOS specification of *service* given in section 3 satisfies the TCTL-formulas given above.

Table 3 shows the number of states, transitions and clocks of the timed automata of service for different sizes of the buffer (see remark 4.1). Tables 4, 5, 6 show the results obtained with KRONOS for timed automata with buffer size equal to 1,2 or 3 respectively. Column "iter" gives the number of iterations required for computing the characteristic set of each formula using the algorithms sketched in section 6.2, column "time" gives the running times measured in seconds (on a 16 MB SUN 4 Sparc Station) and column "eval" shows the result of the evaluation.

We have evaluated the formulas for different values of $\tau_{min}, \tau_{max}, \alpha$ and π. Notice that

Table 5
Results for a buffer of size 2.

	E			F			G		
	$\tau_{min} = 75$	$\alpha = 50$		$\tau_{min} = 50$	$\alpha = 150$		$\tau_{min} = 75$	$\alpha = 50$	
	$\tau_{max} = 120$	$\pi = 100$		$\tau_{max} = 75$	$\pi = 100$		$\tau_{max} = 220$	$\pi = 100$	
	iter	time	eval	iter	time	eval	iter	time	eval
10	15	527.45 s	true	33	45.70 s	false	22	69.15 s	false
11	9	6.93 s	true	7	2.60 s	true	9	3.15 s	true
12	1	0.20 s	true	1	0.21 s	true	1	0.23 s	true
13	17	664.53 s	true	19	92.06 s	false	13	149.83 s	true
14	7	16.98 s	true	8	16.38 s	true	7	10.20 s	true
15	1	0.41 s	true	1	0.41 s	true	1	0.41 s	true

Table 6
Results for a buffer of size 3.

	H		
	$\tau_{min} = 75$	$\alpha = 50$	
	$\tau_{max} = 220$	$\pi = 100$	
	iter	time	eval
10	23	1278.71 s	`true`
11	10	30.35 s	`true`
12	1	2.40 s	`true`
13	13	2582.61 s	`true`
14	8	78.48 s	`true`
15	1	3.51 s	`true`

the values of the parameters affect the validity of the formulas:

- Let b be the size of the buffer. We can see that when $b.\pi < \tau_{max}$, like for instance in columns D and G, the formula 10 is not satisfied. For these values, *service* may not be ready to accept a new cell at `Ss-SAP` because the buffer is full. The formula is verified if we increment the size of the buffer such that $b.\pi \geq \tau_{max}$, like in columns E and H.

- When $\alpha > \pi$, like in columns C and F, the requirement *transmission delay* (formula 13) is not verified. Consider two consecutive emissions separated by π. In the worst case, the first one is delivered in τ_{max}. Then *service* must deliver the second one after $\tau_{max} + \alpha$ to satisfy the spacing between deliveries but before $\pi + \tau_{max}$ as specified by the requirement *transmission delay*. But this is impossible if $\alpha > \pi$. Notice that formula 10 is false whenever formula 13 is false, because the buffer may become full in this case.

Also notice that formulas 10 and formula 13 are harder to verify than the others. Besides, changes of the values of the parameters do not significantly change the running times, except when these changes affect the truth value of the formula.

7. Conclusion and further work

This paper shows that real-time systems described in a reasonable subset of ET-LOTOS can be verified with the tool KRONOS by compiling them into timed automata. It is important to note that the complexity of the translation method and the size of the constructed automaton is independent of the values of the time parameters of the timing constructs of the language. We have also shown that the model-checker KRONOS allows the verification of real-time systems with arbitrarily large time delays.

The practical application of our approach is illustrated with a case study: the *service* entity of the Tick-Tock protocol, specially designed for the assessment of real-time FDTs and tools. We are working on the verification of the complete Tick-Tock protocol described in [9].

In order to keep things simple, we have restricted ourselves to a subset of ET-LOTOS. Nevertheless, our method can be extended to full ET-LOTOS. An efficient compiler is

envisaged by combining the techniques used by the tool CÆSAR [4] and the ATP compiler [12]. The main problem here is how to deal with the @t ("at t") operator. This operator declares a time variable t used to store the time of occurrence of an action. The variable t can be considered as an *integrator* [6, 2], i.e., a clock that can be stopped. The problem is that unrestricted use of integrators leads to undecidability of verification.

Finally, we are currently implementing optimizations to improve on the tool KRONOS in order to be able to cope with larger systems.

Acknowledgements We are grateful to Joseph Sifakis for the helpfull discussions.

REFERENCES

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th LICS*, pages 414–425. IEEE, 1990.
2. A. Bouajjani, R. Echahed, and J. Sifakis. On model-checking for real-time properties with durations. In *8th. Symp. on Logic in Computer Science*. IEEE, 1993.
3. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS* 8(2):244–263, 1986.
4. H. Garavel and J. Sifakis. Compilation and verification of LOTOS specifications. In *Proc. 10th PSTV*, Canada, 1990. IFIP, North-Holland.
5. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th LICS*, pages 394–406. IEEE, 1992.
6. Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In *Workshop on Theory of Hybrid Systems*, pages 179–208. LNCS 736, Springer-Verlag, 1993.
7. G. Leduc and L. Léonard. A timed LOTOS supporting dense time domain and including new timed operators. In *Formal Description Techniques V*, pages 87–102. North-Holland, 1993.
8. G. Leduc and L. Léonard. A formal definition of time in LOTOS. In J. Quemada, editor, *Revised draft on enhancements to LOTOS*. Annex G of document ISO/IEC JTC1/SC21/WG1/Q48.6, 1994.
9. G. Leduc, L. Léonard, and A. Danthine. The Tick-Tock case study for the assessment of timed FDTs. In *The OSI95 transport service with multimedia support on HSLAN's and B-ISDN*, 1994.
10. X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Workshop on Theory of Hybrid Systems*, pages 149–178. LNCS 736, Springer-Verlag, 1993.
11. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proc. 3rd Workshop on Computer-Aided Verification*, pages 376–398, Denmark, 1991. LNCS 575, Springer-Verlag.
12. X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE TSE Special Issue on Real-Time Systems*, 18(9):794–804, 1992.
13. X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.
14. S. Yovine. Méthodes et outils pour la vérification symbolique de systèmes temporisés. Thèse, Institut National Polytechnique de Grenoble, Grenoble, France, 1993.