Mapping RT-LOTOS specifications into Time Petri Nets

Tarek Sadani[1,2], Marc Boyer[3], Pierre de Saqui-Sannes[1,2], and Jean-Pierre Courtiat[1]
tsadani@ensica.fr, mboyer@enseeiht.fr, desaqui@ensica.fr, courtiat@laas.fr

[1] LAAS-CNRS, 7 av. du colonel Roche, 31077 Toulouse Cedex 04, France
[2] ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France
[3] IRIT-CNRS/ENSEEIHT, 2 rue Camichel, 31000 Toulouse, France

Abstract. RT-LOTOS is a timed process algebra which enables compact and abstract specification of real-time systems. This paper proposes and illustrates a structural translation of RT-LOTOS terms into behaviorally equivalent (timed bisimilar) finite Time Petri nets. It is therefore possible to apply Time Petri nets verification techniques to the profit of RT-LOTOS. Our approach has been implemented in RTL2TPN, a prototype tool which takes as input an RT-LOTOS specification and outputs a TPN. The latter is verified using TINA, a TPN analyzer developed by LAAS-CNRS. The toolkit made of RTL2TPN and TINA has been positively benchmarked against previously developed RT-LOTOS verification tool.

1 Introduction

The acknowledged benefits of using Formal Description Techniques (FDTs) include the possibility to verify a model of the system under design against user requirements. These benefits are even higher for systems which are both concurrent and submitted to stringent temporal constraints.

In this paper, formal verification is addressed in the context of RT-LOTOS, a timed extension of the ISO-based LOTOS [1] FDT. RT-LOTOS [2] shares with LOTOS and other process algebras its capability to specify systems as a collection of communicating processes. The paper proposes a transformational approach from RT-LOTOS to Time Petri Nets (TPNs) which, by contrast, are typical example of non compositional FDT. Therefore, it is proposed to embed TPNs into so-called *components* that can be composed relying on different patterns. The latters are carefully defined so as they ensure a very tight relation between the obtained composite TPN and its corresponding RT-LOTOS behavior. This work can be seen as giving a TPN semantics to RT-LOTOS denotationally. A prototype tool implements the translation patterns. It has been interfaced with TINA [3], the Time Petri Net Analyzer developed by LAAS-CNRS. We show that an automatically generated reachability graph of a TPN can be used to reason about and check the correctness of RT-LOTOS specifications.

The paper is organized as follows. Section 2 introduces the RT-LOTOS language. Section 3 introduces the Time Petri net (TPN) model. Section 4 discusses the theoretical foundations of RT-LOTOS to TPNs mapping. Section 5 addresses practical issues, including tool development and verification results obtained for well-established benchmarks. Section 6 surveys related work. Section 7 concludes the paper.

2 RT-LOTOS

The Language of Temporal Ordering Specifications (LOTOS, [1]), is a formal description technique, based on CCS [4] and extended by a multi-way synchronization mechanism inherited from CSP [5]. In LOTOS, process definitions are expressed by the specification of behavior expressions which are constructed by means of a restricted set of powerful operators making it possible to express behaviors as complex as desired. Among these operators, action prefixing, choice, parallel composition and hiding play a fundamental role.

RT-LOTOS [2] extends LOTOS with three temporal operators: a *deterministic delay* ($\triangle^t$), a *latency*(Ω^t) which enables description of temporal indeterminism and a *time limited offer* $g\{t\}$. The main difference between RT-LOTOS and other timed extensions of LOTOS lies in the way a non-deterministic delay may be expressed. The solution developed for RT-LOTOS is the latency operator. Its usefulness and efficiency have been proved in control command applications and hypermedia authoring [6].

RT-LOTOS formal syntax: Let PV be the set of process variables and X range over PV. Let GV be the set of the user-definable gates. Let g,$g'_1 \ldots g'_n \in GV$, let also L be any (possibly empty) subset of GV noted $L = g'_1 \ldots g'_n$ and i the internal action.

The formal syntax of RT-LOTOS is recursively given by:

`P` ::= `stop` | `exit` | `X[L]` | g`;P` | $g\{t\}$`;P` | $i\{t\}$`;P` | $\triangle^t$`P` | Ω^t`P` | `P [] P` | `P|[L]|P` | `hide L in P` | `P` $\gg$ `P` | `P[>P`

The syntax of a process definition being "process X[$g'_1 \ldots g'_n$]:=P_X endproc". Two alternative syntaxes have been defined for expressing time delays: *delay(t)* which is identical to $\triangle^t$, and latencies, namely *latency(t)* meaning Ω^t.

RT-LOTOS operational semantics in the classical Plotkin's SOS style can be found in [7].

3 Time Petri nets

Petri nets were, to our knowledge, the first theoretical model augmented with time constraints [8, 9], and the support of the first reachability algorithm for timed system [10, 11].

The basic idea of time Petri nets (TPN [8, 9]) is to associate an interval $I_s(t) = [a, b]$ (static interval) with each transition t. The transition *can* be fired

if it has continuously been enabled during at least a time units, and it *must* fire if continuous enabling time reaches b time units[4].

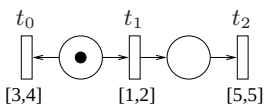


Fig. 1. Priority from urgency

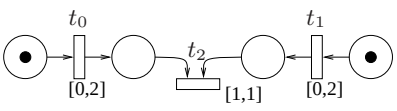


Fig. 2. Synchronization

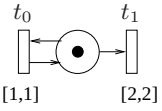


Fig. 3. Continuous enabling

Figure 1 is a first example: in the initial marking, only t_0 and t_1 are enabled. After one time unit delay, t_1 is firable. Because t_1 reaches its upper interval always before t_0 becomes enable ($3 > 2$), then t_0 can never be fired. t_2 is fired five time units after the firing of t_1. Figure 2 illustrates the synchronization rule: t_0 (resp. t_1) is fired at an absolute date $\theta_0 \leq 2$ (resp. $\theta_1 \leq 2$), and t_2 is fired at $max(\theta_0, \theta_1)+1$. Figure 3 illustrates another important point: continuous enabling. In that TPN, transition t_1 will *never* be fired, because, at each time unit, t_0 is fired, removing the token and putting it back immediately. Then, t_1 is at most 1 time unit continuously enabled, never 2 time units.

4 Translation definition

This section gives the translation from RT-LOTOS terms into TPNs. This mapping can also be understood as the definition of an alternative TPNs semantics of RT-LOTOS. It is well known that process algebras (e.g. RT-LOTOS) heavily rely on the concept of compositionality, whereas Petri nets (and their timed versions) lack of compositionality at all. To make the translation possible, a core idea behind our approach is to view a TPN as a composition of number of TPN components. The following section introduces the concept of *TPN Component* as basic building block.

4.1 Time Petri net Component

A *Component* encapsulates a labeled TPN which describes its behavior. It is handled through its interfaces and interactions points. A component performs an action by firing the corresponding transition. A component has two sets of labels: *Act* which is the alphabet of the component and $Time = \{\text{tv}, \text{delay}, \text{latency}\}$. These three labels are introduced to represent the intended temporal behavior of a component. The tv (for "temporal violation") label represents the expiration of time limited-offering. A delay or latency label represents the expiration of a deterministic delay or a non deterministic delay, respectively.

[4] This urgency when the deadline is reached is called "strong semantics".

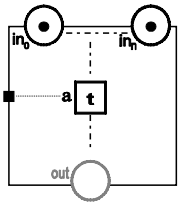


Fig. 4. Component example

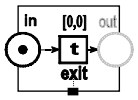


Fig. 5. The `exit` pattern

A component is graphically represented by a box containing a TPN. The black-filled boxes at the component boundary represent interaction points. For instance, the component $C_{\mathtt{P}}$ in the Figure 4 is built from a RT-LOTOS term `P`. During its execution, it may perform the observable action `a`. The in_i (initially marked places) represent the component input interface, and the *out* place denotes its output interface. A token in the *out* place of a component means that the component has successfully completed its execution. A component is *activated* by filling its input places. A component is *active* if at least one of its transitions is enabled. Otherwise, the component is *inactive*.

Definition 1 (Component).
Let $Act = A_o \cup A_h \cup \{exit\}$ be an alphabet of actions, where A_o is a set of observable actions *(with $i \notin A_o$, $exit \notin A_o$), $A_h = \{i\} \times A_o$ is the set of hidden actions (If $\mathtt{a}$ is an observable action, $i_{\mathtt{a}}$ denotes a hidden action).*

Formally a component is a tuple $C = \langle \Sigma, Lab, I, O \rangle$ where

- *$\Sigma = \langle P, T, Pre, Post, M_0, IS \rangle$ is a TPN.*
- *$Lab : T \rightarrow (Act \cup Time)$ is a labeling function which labels each transition in Σ with either an action name (Act) or a time-event ($Time = \{tv, delay, latency\}$). Let T^{Act} (resp. T^{Time}) be the set of transitions with labels in Act (resp. $Time$).*
- *$I \subset P$ is a non empty set of places defining the input interface.*
- *$O \subset P$ is the output interface of the component. A component has an output interface if it has at least one transition labeled by `exit`. If so, O is the outgoing place of those transitions. Otherwise, $O = \emptyset$.*

The following invariants apply to all components:

H1 There is no source transition in a component.
H2 The encapsulated TPN is 1-bounded (cf. *safe* nets in [12]). H2 is called the "safe marking" property. It is essential for the decidability of reachability analysis procedure applied to TPNs.
H3 If all the input places are marked, all other places are empty ($I \subset M \Rightarrow M = I$).
H4 If the *out* place is marked, all other places are empty ($O \neq \emptyset \wedge O \subset M \Rightarrow M = O$).

H5 For each transition t such that $Lab(t) \in Act$, if the label is an observable action ($Lab(t) \in A_0$), its time interval is $[0, \infty)$, otherwise[5], it is $[0, 0]$.

4.2 Translation patterns

RT-LOTOS behaviour expressions are inductively defined by means of algebraic operators acting on behaviour expressions. Since the translation is meant to be syntax driven we need to endow TPNs with operators corresponding to RT-LOTOS ones, so as to allow one to construct a composite TPN. In the following, we first define components corresponding to nullary algebraic operators (`stop` and `exit`) and, given each RT-LOTOS operator and its operands (i.e. behaviour expressions), we inductively describe how to obtain a new component starting from the one corresponding to the given RT-LOTOS behaviour expressions. The resulting component corresponds to the RT-LOTOS behaviour expression computed by applying the operator to the given operands.

Due to lack of space, the formalization of some patterns is skipped. A complete formal definition can be found in the extended version of this paper [13].

Notation and definition $f' = f \cup (a, b)$ defines the function $f' : A \cup \{a\} \mapsto B \cup \{b\}$ such that $f'(x) = f(x)$ if $x \in A$ and $f'(a) = b$ otherwise.

Definition 2 (First actions set). *Let C be a component. The set of first actions $\mathcal{FA}(C_P)$ can be recursively built using the following rules*[6]*:*

$$\mathcal{FA}(C_{stop}) = \emptyset \qquad \mathcal{FA}(C_{exit}) = \{t_{exit}\}$$
$$\mathcal{FA}(C_{a;P}) = \{t_a\} \qquad \mathcal{FA}(C_{\mu X.\,(P;X)}) = \mathcal{FA}(C_P)$$
$$\mathcal{FA}(C_{a\{d\}P}) = \{t_a\} \qquad \mathcal{FA}(C_{delay(d)P}) = \mathcal{FA}(C_P)$$
$$\mathcal{FA}(C_{latency(d)P}) = \mathcal{FA}(C_P) \qquad \mathcal{FA}(C_{P;Q}) = \mathcal{FA}(C_P)$$
$$\mathcal{FA}(C_{P|[A]|Q}) = \mathcal{FA}(C_P) \cup \mathcal{FA}(C_Q) \qquad \mathcal{FA}(C_{P>>Q}) = \mathcal{FA}(C_P)$$
$$\mathcal{FA}(C_{P[]Q}) = \mathcal{FA}(C_P) \cup \mathcal{FA}(C_Q) \qquad \mathcal{FA}(C_{P[>Q}) = \mathcal{FA}(C_P) \cup \mathcal{FA}(C_Q)$$
$$\mathcal{FA}(C_{hide\ a\ in\ P}) = h_a(\mathcal{FA}(C_P))$$

Low level Petri net operations The formal definition of the translation patterns uses the following low level Petri nets operators: $\cup, \setminus, \uplus$.

Let $N = \langle P, T, Pre, Post, M_0, IS \rangle$ be a TPN.

Adding a place Let p be a new place ($p \notin P$), Pre_p and $Post_p$ two sets of transitions of T. Then $N' = N \cup \langle Pre_p, p, Post_p \rangle$ is the TPN augmented with place p such that ${}^\bullet p = Pre_p$ and $p^\bullet = Post_p$.

Adding a transition: Let t be a new transition ($t \notin T$), I its time interval, Pre_t and $Post_t$ two sets of places of P. Then $N' = N \cup \langle Pre_t, (t, I), Post_t \rangle$ is the TPN augmented with transitions t such that ${}^\bullet t = Pre_t$ and $t^\bullet = Post_t$.

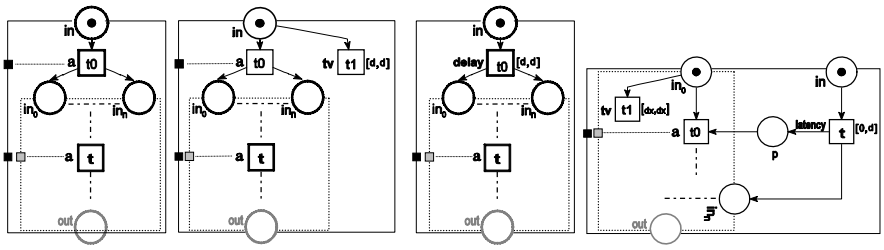
Similarly, $\setminus$ is used to remove places or transitions from a net (and all related arcs), and $\uplus$ denotes the free merging of two nets.

[5] $Lab(t) \in A_h \cup \{exit\}$

[6] where t_a is transition labelled by a. $h_a(\alpha) = \alpha$ if $\alpha \neq a$ and $h_a(a) = i_a$

Basic components The $C_{\texttt{stop}}$ component is simply the empty net (no place, no transition). $C_{\texttt{exit}}$ is a component which performs a successful termination. It has one input place, one output place, and a single transition labeled with `exit` and a static interval $[0, 0]$ (Fig.5).

Patterns applying to one component Let us consider the component $C_{\texttt{P}}$ of Fig. 4. Fig. 6 depicts different patterns applied to $C_{\texttt{P}}$.



(a) `a;P` (b) `a{d}P` (c) `delay(d)P` (d) `latency(d)P`

Fig. 6. Patterns applying to one component

– $C_{\texttt{a;P}}$ (Fig. 6(a)) is the component resulting from prefixing $C_{\texttt{P}}$ with action `a`. $C_{\texttt{a;P}}$ executes `a` then activates $C_{\texttt{P}}$.
$C_{\texttt{a;P}} = \langle \Sigma_{\texttt{a;P}}, Lab_{\texttt{a;P}}, \{in\}, O_{\texttt{P}} \rangle$ where the TPN $\Sigma_{\texttt{a;P}}$ is obtained by adding a place in and a transition t_0 to $\Sigma_{\texttt{P}}$, $Lab_{\texttt{a;P}}$ associates `a` to transition t_0.

$$\Sigma_{\texttt{a;P}} = (\Sigma_{\texttt{P}} \cup \langle \emptyset, (t_0, [0, \infty)), I_{\texttt{P}} \rangle) \cup \langle \emptyset, in, t_0 \rangle$$
$$Lab_{\texttt{a;P}} = Lab_{\texttt{P}} \cup (t_0, \texttt{a})$$

– $C_{\texttt{a\{d\};P}}$ (Fig. 6(b)) is the component resulting from prefixing $C_{\texttt{P}}$ with a limited offer of d units of time on action `a`. If for any reason, `a` cannot occur during this time interval, the `tv` transition will be fired (temporal violation situation) and $C_{\texttt{a\{d\};P}}$ will transform into an inactive component.

$$C_{\texttt{a\{d\};P}} = \langle \Sigma_{\texttt{a\{d\};P}}, Lab_{\texttt{a;P}} \cup \{(t_1, \text{tv})\}, \{in\}, O_{\texttt{P}} \rangle$$
$$\Sigma_{\texttt{a\{d\};P}} = \Sigma_{\texttt{a;P}} \cup \langle \{in\}, (t_1, [d, d]), \emptyset \rangle$$

– $C_{\texttt{delay(d)P}}$ (Fig 6(c)) is the component resulting from delaying the first action of `P` with a deterministic delay of `d` units of time. This is exactly the same pattern as $C_{\texttt{a;P}}$ except that the added transition has a delay label and a static interval equal to $[d, d]$.

$$C_{\texttt{delay(d)P}} = \langle \Sigma_{\texttt{delay(d)P}}, Lab_{\texttt{P}} \cup \{(t_0, \text{delay})\}, \{in\}, O_{\texttt{P}} \rangle$$
$$\Sigma_{\texttt{delay(d)P}} = (\Sigma_{\texttt{P}} \cup \langle \emptyset, (t_0, [d, d])), I_{\texttt{P}} \rangle) \cup \langle \emptyset, in, t_0 \rangle$$

– $C_{\texttt{latency(d)P}}$ (Fig 6(d)) is the component resulting from delaying the first actions of $C_{\texttt{P}}$ with a non deterministic delay of `d` units of time.

Like the delay operator, latency is defined by connecting a new transition to the input interface of $C_{\texttt{P}}$. But this time, we add a static interval equal to $[0, d]$. The definition of the latency translation pattern must handle the "subtle" case where one (or several) action(s) among $C_{\texttt{P}}$'s first actions is (are) constrained with a limited offer (this set is denoted by $\mathcal{FA}_{lo}$). For instance, in Fig 6(d), action `a` is offered to the environment during d_x units of time. The RT-LOTOS semantics states that the latency and the offering of `a` start simultaneously, which means that if the latency duration goes beyond d_x units of time, the offer on `a` will expire. To obtain the same behavior, we add the input place in_0 of `a` to the input interface of the resulting component $C_{\texttt{latency(d)P}}$. In the definition of the pattern, we denote I_{lo} the set of these input places ($I_{lo} \subset I_{\texttt{P}}$). Thus t_1 and t are enabled as soon as the component is activated (all its input places being marked). $C_{\texttt{latency(d)P}}$ is able to execute `a` (fire t_0) if t_0 is enabled (i.e if in_0 and p are marked) before t_1 is fired (at d_x). Therefore, action `a` is possibly offered to the environment for no more than d_x units of time, hence conforming to the RT-LOTOS semantics.

Let $\mathcal{FA}(C_{\texttt{P}})$ be the set of transitions associated to the first actions of `P`[7], and $\mathcal{FA}_{lo}(C_{\texttt{P}})$ be the set of first actions constrained by a time limited offer:

$$
\begin{aligned}
\mathcal{FA}_{lo}(C_{\texttt{P}}) &= \left\{ t_a \in \mathcal{FA}(C_{\texttt{P}}) \;\middle|\; \text{tv} \in ({}^{\bullet}t_a)^{\bullet} \right\} \\
I_{lo} &= {}^{\bullet}\mathcal{FA}_{lo}(C_{\texttt{P}}) \\
C_{\texttt{latency(d)P}} &= \langle \Sigma_{\texttt{latency(d)P}}, Lab_{\texttt{P}} \cup \{(t, \text{latency})\}, I_{lo} \cup \{in\}, O_{\texttt{P}} \rangle \\
\Sigma_{\texttt{latency(d)P}} &= \Sigma_{\texttt{P}} \cup \bigcup_{t_a \in \mathcal{FA}_{lo}(C_{\texttt{P}})} \langle t, p_{t_a}, t_a \rangle \cup \langle \emptyset, in, \emptyset \rangle \\
&\quad \cup \left\langle \{in\}, (t, [0, d]), (I_{\texttt{P}} \backslash I_{lo}) \cup \bigcup_{t_a \in \mathcal{FA}_{lo}(C_{\texttt{P}})} \{p_{t_a}\} \right\rangle
\end{aligned}
$$

– $C_{\mu\texttt{X.(P;X)}}$ The recursion operator translation is mainly an untimed problem (relying on fixpoint theory). It is not presented in this paper, focused on timed aspects.

– $C_{\texttt{hide a in P}}$ is the component resulting from hiding action `a` in $C_{\texttt{P}}$. Hiding allows one to transform observable (external) actions into unobservable (internal) actions, then making the latter unavailable for synchronization with other components. In RT-LOTOS, hiding one or several actions induces a notion of urgency on action occurrence. Consequently, a TPN transition corresponding to a hidden action will be constrained by a time interval equal to $[0, 0]$. This implies that as soon as a transition is enabled, it is candidate for being fired.

[7] Its formal definition can be found in Def. 2, Sect. 4.2.

Patterns applying to a set of components Each of the following patterns transforms a set of components into one component.

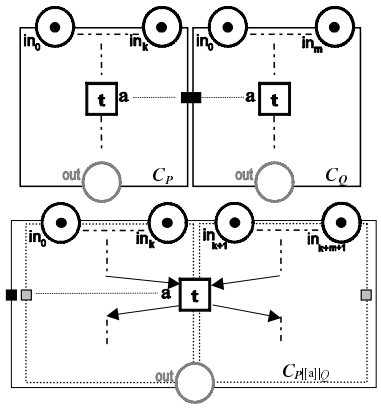


Fig. 7. Parallel synchronization pattern

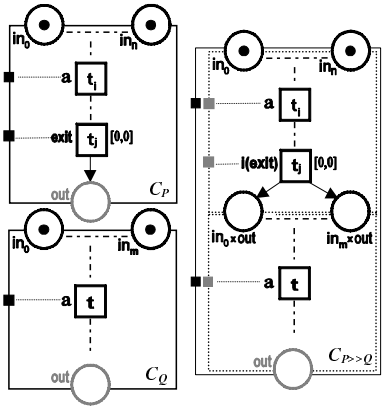


Fig. 8. Sequential composition pattern

– $C_{\mathtt{P[[a]]Q}}$ (Fig.7)
In Petri nets, a multi-way synchronization is represented by a transition with n input places. This transition can fire only if all its input places contain a token (cf. Fig. 2). At the PN level, the synchronization operation is achieved through transition merging. While in untimed Petri nets, the operation of transitions merging is straightforward, it turns to be a rather tricky issue in Time Petri nets. Indeed, it requires explicit handling of the time intervals assigned to the transitions to be merged. Possible incompatibility of these time intervals leads to express global timing constraints as a conjunction of intervals whose consistency is not guaranteed. This problem is not solved in [14] where each transition is assigned a time interval.
To solve this problem and make transitions merging operation always possible, we avoid assigning time intervals to actions transitions. Instead, the timing constraints are assigned to conflicting transitions (cf. time limited offer pattern). The advantage of this solution is twofold:
1) To allow an incremental combination of timing constraints. Figure 7 depicts synchronization between two components $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ on action `a`. Our goal is to define a compositional framework, where each component involved in this synchronization may add timing constraints with respect to the occurrence of action `a`, such that the global timing constraint on `a` in $C_{\mathtt{P[[a]]Q}}$ will be the conjunction of several simpler constraints. This implies that when component $C_{\mathtt{P}}$ is ready to execute `a`, it is forced in the absence of alternative actions, to wait for $C_{\mathtt{Q}}$ to offer `a`. This may lead for example to the expiration of a limited temporal offer on `a` in $C_{\mathtt{P}}$. This goal is achieved

without explicitly handling time intervals, and the synchronization is modeled as a straightforward transition merging as in untimed Petri nets.

2) Relaxing the TPN's strong semantics. In TPNs the firing of transitions is driven by necessity. Thus an action has to be fired as soon as its upper bound is reached (except for a transition in conflict with another one). Like process algebras in general, RT-LOTOS favors an interaction point of view, where the actual behavior of a system is influenced by its environment. Thus, an enabled transition may fire within its enabling time window but it cannot be forced to fire. A wide range of real-time systems work on that principle. In particular, soft real-time systems are typical examples of systems that cannot be forced to synchronize with their environment. This behavior would not be possible if the actions transitions were assigned time intervals, because their firing would be driven by necessity. To model necessity in RT-LOTOS we use the `hide` operator. Its combination with the 'temporal limited offering' and the 'latency' operators gives an interesting flexibility in terms of expressiveness.

The synchronization on `a` of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ is achieved by merging each `a` transition in $C_{\mathtt{P}}$ with each `a` transition in $C_{\mathtt{Q}}$, thus creating $n*m$ `a` transitions in $C_{\mathtt{P|[a]|Q}}$ (n and m being respectively the number of `a` transitions in $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$).

Let $T_{\mathtt{X}}^{\mathtt{a}}$ be the set of transitions labelled with `a` in $C_{\mathtt{X}}$.

$$T_{\mathtt{X}}^{\mathtt{a}} = \{t \in T_{\mathtt{X}} \mid Lab_{\mathtt{X}}(t) = \mathtt{a}\} \qquad\qquad T_{\mathtt{X}}^{\mathtt{A}} = \bigcup_{\mathtt{a}\in\mathtt{A}} T_{\mathtt{X}}^{\mathtt{a}}$$

The net $\Sigma_{\mathtt{P|[A]|Q}}$ is obtained by replacing each transition t_p in $C_{\mathtt{P}}$ with label $\mathtt{a} \in \mathtt{A}$ by a set of transitions (t_p, t_q) such that t_q is also labelled by `a`, with ${}^\bullet(t_p, t_q) = {}^\bullet t_p \cup {}^\bullet t_q$ and $(t_p, t_q)^\bullet = t_p{}^\bullet \cup t_q{}^\bullet$.

A $[0, \infty)$ temporal interval is associated with the newly created transition (cf. H5).

Note that the two components have to synchronize on `exit` transition to conform to RT-LOTOS semantics. The two output interfaces are merged: $Out = \{out\}$ if $O_{\mathtt{P}} \neq \emptyset \wedge O_{\mathtt{Q}} \neq \emptyset$, $Out = \emptyset$ otherwise.

Let us denote $merge(t_p, t_q) = \langle {}^\bullet t_p \cup {}^\bullet t_q, ((t_p, t_q), IS(t_p)), t_p{}^\bullet \cup t_q{}^\bullet \rangle$; $A' = A \cup \{\mathtt{exit}\}$; $PreOut = T_{\mathtt{P}}^{\mathtt{exit}} \times T_{\mathtt{Q}}^{\mathtt{exit}}$ if $O_{\mathtt{P}} \neq \emptyset \wedge O_{\mathtt{Q}} \neq \emptyset$, $PreOut = \emptyset$ otherwise.

$$C_{\mathtt{P|[A]|Q}} = \langle \Sigma_{\mathtt{P|[A]|Q}}, Lab_{\mathtt{P|[A]|Q}}, I_{\mathtt{P}} \cup I_{\mathtt{Q}}, Out \rangle$$

$$\Sigma_{\mathtt{P|[A]|Q}} = \left(\Sigma_{\mathtt{P}} \backslash T_{\mathtt{P}}^{\mathtt{A'}} \backslash O_{\mathtt{P}}\right) \uplus \left(\Sigma_{\mathtt{Q}} \backslash T_{\mathtt{Q}}^{\mathtt{A'}} \backslash O_{\mathtt{Q}}\right) \cup \bigcup_{t_p \in T_{\mathtt{P}}^{\mathtt{A'}}, t_q \in T_{\mathtt{Q}}^{\mathtt{A'}}} merge(t_p, t_q) \cup \langle PreOut, Out, \emptyset \rangle$$

$$Lab_{\mathtt{P|[A]|Q}}(t) = \begin{cases} Lab_{\mathtt{X}}(t) & \mathtt{if}\ \ t \in T_{\mathtt{X}}, \mathtt{X} \in \{\mathtt{P}, \mathtt{Q}\} \\ \mathtt{a} & \mathtt{if}\ \ t = (t_p, t_q) \wedge t_p \in T_{\mathtt{P}}^{\mathtt{a}} \end{cases}$$

– $C_{\mathtt{P>>Q}}$ (Fig. 8) depicts a sequential composition of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ which means that if $C_{\mathtt{P}}$ successfully completes its execution then it activates $C_{\mathtt{Q}}$. This kind

of composition is possible only if $C_{\mathtt{P}}$ has an output interface. The resulting component $C_{\mathtt{P>>Q}}$ is obtained by merging the output interface of $C_{\mathtt{P}}$ and the input interface of $C_{\mathtt{Q}}$, and by hiding the `exit` interaction point of $C_{\mathtt{P}}$.

$$C_{\mathtt{P>>Q}} = \langle \Sigma_{\mathtt{P>>Q}}, Lab_{\mathtt{hide\ exit\ in\ P}} \cup Lab_{\mathtt{Q}}, I_{\mathtt{P}}, 0_{\mathtt{Q}} \rangle$$
$$\Sigma_{\mathtt{P>>Q}} = \langle P_{\mathtt{P}} \backslash O_{\mathtt{P}} \cup P_{\mathtt{Q}}, T_{\mathtt{hide\ exit\ in\ P}} \cup T_{\mathtt{Q}}, Pre_{\mathtt{P}} \cup Pre_{\mathtt{Q}}, Post_{\mathtt{P>>Q}}, IS_{\mathtt{P}} \cup IS_{\mathtt{Q}} \rangle$$
$$Post_{\mathtt{P>>Q}} = (Post_{\mathtt{P}} \backslash \{(t, O_{\mathtt{P}}) \mid t \in {}^{\bullet}O_{\mathtt{P}}\}) \cup \{(t, in_{\mathtt{Q}}) \mid in_{\mathtt{Q}} \in I_{\mathtt{Q}} \wedge t \in {}^{\bullet}O_{\mathtt{P}}\} \cup Post_{\mathtt{Q}}$$

– $C_{\mathtt{P[]Q}}$ (Fig. 9) is the component which behaves either as $C_{\mathtt{P}}$ or $C_{\mathtt{Q}}$.
We do not specify whether the choice between the alternatives is made by the component $C_{\mathtt{P[]Q}}$ itself, or by the environment. Anyway, it should be made at the level of the first actions in the component. In other words, the occurrence of one of the first actions in either component determines which component will continue its execution and which one must be deactivated. The problem can be viewed as a competition between $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$. These two components compete to execute their first action. As long as the that action has not yet occurred, $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ age similarly, which means that $Time$ transitions (labeled by tv, delay or latency) may occur in both components without any consequence on the choice of the wining component. Once one first action has occurred, the control is irreversibly transferred to the winning component, the other one being deactivated, in the sense that it no longer contains enabled transitions. The choice operator is known to cause trouble in presence of initial parallelism. [15] defines a choice operator where each alternative has just one initial place. Therefore, none of the alternative allows any initial parallelism. We think that it is a strong restriction. We do not impose any constraint on the choice alternatives.
The solution we propose to define a choice between two components is as follows: to obtain the intended behavior, we introduce a set of special places, called *lock* places. Those places belong to the input interface of component $C_{\mathtt{P[]Q}}$. Their function is to undertake control transfer between the two components. For each first action of $C_{\mathtt{P}}$ we introduce one lock place per concurrent first action in $C_{\mathtt{Q}}$ (for instance `a` has one concurrent action in $C_{\mathtt{Q}}$: `c`, while `c` has two concurrent actions in $C_{\mathtt{P}}$: `a` and `b`) and vice versa. A lock place interacts only with those transitions representing the set of initial actions and the time labeled transitions they are related with (delay for `a` and tv for `b`). $Time$ transitions restore the token in the lock place, since they do not represent an action occurrence, but a time progression which has not to interfere with the execution of the other component (as long as the first action has not occurred, the two components age similarly). The occurrence of an initial action of $C_{\mathtt{P}}$ (respectively $C_{\mathtt{Q}}$) locks the execution of $C_{\mathtt{Q}}$ (respectively $C_{\mathtt{P}}$) by stealing the token from the lock places related to all $C_{\mathtt{Q}}$'s (respectively $C_{\mathtt{P}}$'s) first actions.
A unique *out* place is created by merging the *out* places of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$.
– $C_{\mathtt{P[>Q}}$ (Fig. 10) is the component representing the behavior where component $C_{\mathtt{P}}$ can be interrupted by $C_{\mathtt{Q}}$ at any time during its execution. It means that at any point during the execution of $C_{\mathtt{P}}$, there is a choice between executing

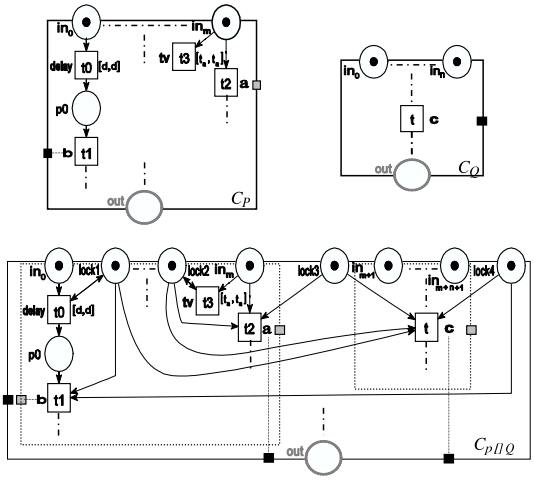


Fig. 9. Choice between $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$

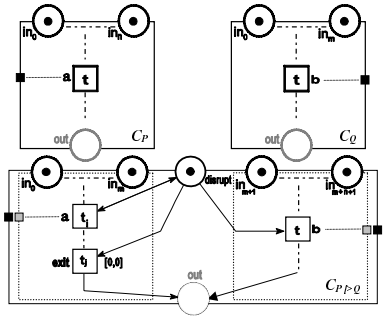


Fig. 10. The disrupt pattern

one of the next actions from $C_{\mathtt{P}}$ or one of the first actions from $C_{\mathtt{Q}}$. For this purpose, $C_{\mathtt{Q}}$ *steals* the token from the shared place named `disrupt` (which belongs to the input interface of $C_{\mathtt{P[>Q}}$), thus the control is irreversibly transferred from $C_{\mathtt{P}}$ to $C_{\mathtt{Q}}$ (`disrupt` is an *input* place for $C_{\mathtt{Q}}$ first action and `exit` transition of $C_{\mathtt{P}}$, it is also an *input/output* place for all the others transitions of $C_{\mathtt{P}}$). Once an action from $C_{\mathtt{Q}}$ is chosen, $C_{\mathtt{Q}}$ continues executing, and transitions of $C_{\mathtt{P}}$ are no longer enabled.

4.3 Formal proof of the translation consistency

We prove that the translation preserves the RT-LOTOS semantics and that the defined compositional framework preserves the good properties (H1–H5) of the components. This is done by induction: assuming that some components $C_1, ..., C_n$ are respectively *equivalent* to some RT-LOTOS terms $T_1, ..., T_n$, and given a RT-LOTOS operator Op, we prove that the pattern applied to $C_1, ..., C_n$ gives a component which is equivalent to the term $Op(T_1, ..., T_n)$ (the behavior over time must be accounted for).

The proof is carried out in two steps:

– we first define a more informative RT-LOTOS semantics, which does not introduce any new operation, but explicitly acquaints the occurrence of *time-events*. A *time-event* represents the expiration of an RT-LOTOS temporal operator. As an illustration, let us consider the rule of the limited offering as it appears in the original semantics of RT-LOTOS. In the following rule, any delay $d' > d$ will *silently* transform a process `a{d};P` into `stop`.

$$\mathtt{a\{d\};P} \xrightarrow{d'} \mathtt{stop} \tag{1}$$

In the augmented RT-LOTOS semantics, a "tv" transition is introduced to denote the limited offer expiration (cf 2).

$$\texttt{a\{d\};P} \xrightarrow{d} \texttt{a\{0\};P} \xrightarrow{tv} \texttt{stop} \xrightarrow{d'-d} \texttt{stop} \tag{2}$$

A delay $d' > d$ is of course still allowed from `a{d};P`, but it is splited into three steps: a delay d, a "temporal violation" (tv), and a delay $d' - d$.
it is easy to define a branching bisimulation[8] which abstracts from the occurence of the newly added *time-event* transitions and show that the new semantics of RT-LOTOS is branching bisimilar to the original semantics of RT-LOTOS.

– We then prove that the semantic model of the components is strongly timed bisimilar to this more informative RT-LOTOS semantics. Intuitively an RT-LOTOS term and a component are timed bisimilar [16] iff they perform the same action at the same time and reach bisimilar states. For each operator, we prove that, from each reachable state, if the occurrence of a time progression (respectively an action) is possible in an RT-LOTOS term, it is also possible in its associated component, and conversely. Therefore, we ensure that the translation preserves the sequences of possible actions but also the occurrence dates of these actions. It is worth to mention that during the execution of a component the structure of the encapsulated TPN remains the same; only the markings are different, while the structure of an RT-LOTOS term may change through its execution. As a result, the TPN encapsulated in a component may not directly correspond to an RT-LOTOS behavior translation but is strongly bisimilar with a TPN which does correspond to an RT-LOTOS expression translation (The same TPN and current state without the unrechable structure).

Let us illustrate the template of the proof on the parallel synchronization.

Notation A paragraph starting with $\xrightarrow{\mathbb{R}}$ proves that each time progression which applies to the RT-LOTOS term is acceptable in its associated component. Conversely, $\xleftarrow{\mathbb{R}}$ denotes the opposite proof. Similarly $\xrightarrow{a}$ and $\xleftarrow{a}$ are used for the proof on actions occurrences.

Proof of the synchronization pattern (Fig. 7)

$\xrightarrow{\mathbb{R}}$**:** A time progression is acceptable in $C_{\texttt{P|[A]|Q}}$ iff it is acceptable for each enabled transition.
By construction, $T_{\texttt{P|[A]|Q}} = (T_{\texttt{P}} \backslash T_{\texttt{P}}^{\texttt{A'}}) \cup (T_{\texttt{Q}} \backslash T_{\texttt{Q}}^{\texttt{A'}}) \cup E$, with E the set of newly created transitions: $E = \bigcup_{t_p \in T_{\texttt{P}}^{\texttt{A'}}, t_q \in T_{\texttt{Q}}^{\texttt{A'}}} merge(t_p, t_q)$.

[8] Our temporal branching bisimulation looks like the weak bisimulation of CCS which abstracts the internal actions. The difference with ours is that the *time-event* transitions do not resolve the choice and the disabling contrary to the internal actions of CCS.

Let t be an enabled transition of $C_{\texttt{P|[A]|Q}}$.
If t is in $(T_{\texttt{P}} \backslash T_{\texttt{P}}^{\texttt{A'}})$, by construction this time progression is acceptable for t since its initial timing constraint in $C_{\texttt{P}}$ has not been changed, and it is not involved in the synchronization. The same applies if t is in $(T_{\texttt{Q}} \backslash T_{\texttt{Q}}^{\texttt{A'}})$.
If t is in E, we show that $Lab(t) \neq \texttt{exit}$. Let us assume that $Lab(t) = \texttt{exit}$. By construction, it exists t_p and t_q such that $t = (t_p, t_q)$ and $Lab(t_p) = Lab(t_q) = \texttt{exit}$. `exit` is a special urgent action (cf. H5), its execution is enforced as soon as it is enabled. By construction ${}^\bullet t = {}^\bullet t_p \cup {}^\bullet t_q$: if t is enabled, then t_p is enabled in $C_{\texttt{P}}$ and t_q is enabled in $C_{\texttt{Q}}$. The enabling of t_p and t_q precludes any time progression. By induction, it precludes time progression in `P` and `Q` and then in `P|[A]|Q` ($\overset{\mathbb{R}}{\not\rightarrow}$).
From H5, we have that all non `exit` transitions in E are associated with a time interval equal to $[0, \infty)$. Therefore, time progression is also acceptable in $C_{\texttt{P|[A]|Q}}$.

$\overset{\mathbb{R}}{\leftarrow}$**:** The proof is similar to $\overset{\mathbb{R}}{\rightarrow}$.

$\overset{a}{\rightarrow}$**:** Assuming `P|[A]|Q` $\overset{a}{\rightarrow}$, is this action acceptable for $C_{\texttt{P|[A]|Q}}$?
Two cases must be discussed: either action `a` is a synchronization action between `P` and `Q` (`a` $\in$ `A'`), or it is not.

Case `a`$\in$`A'`**:** A synchronization action `a` is possible in `P|[A]|Q` iff it is possible in `P` and in `Q`. By induction, it exists transitions t_p and t_q labelled by a firable in $C_{\texttt{P}}$ and in $C_{\texttt{Q}}$.
That is to say, marking ${}^\bullet p_t \subseteq M_{\texttt{P}}$, and the same for `Q`. By construction, we have ${}^\bullet(t_p, t_q) = {}^\bullet t_p \cup {}^\bullet t_q$, then the marking $M_{\texttt{P|[A]|Q}}$ enables (t_p, t_q). After the firing, the marking of $C_{\texttt{P|[A]|Q}}$ can be seen as the union of the one of $C_{\texttt{P}}$, and $C_{\texttt{Q}}$, because $(t_p, t_q)^\bullet = t_p{}^\bullet \cup t_q{}^\bullet$.

Case `a`$\notin$`A'`**:** If action `a` occurs in `P|[A]|Q` it either is an action of `P` or an action of `Q`. By induction hypothesis, it is either a firable transition in $C_{\texttt{P}}$ or in $C_{\texttt{Q}}$. Since this transition is not modified in $C_{\texttt{P|[A]|Q}}$, it is firable in $C_{\texttt{P|[A]|Q}}$.

$\overset{a}{\leftarrow}$**:** Similarly to $\overset{a}{\rightarrow}$.

5 Tools and experiments

5.1 Tools

RTL. The Real-Time LOTOS Laboratory developed by LAAS-CNRS [17], takes as input an RT-LOTOS specification and generates either a simulation trace or a reachability graph. RTL generates a minimal reachability graph preserving the CTL[9] properties.

TINA. (TIme petri Net Analyzer [3]) is a software environment to edit and analyze Petri nets and Time Petri nets.

TINA offers various abstract state space constructions that preserve specific classes of properties of the concrete state space of the nets. Those classes of

[9] Computational Tree Logic

properties include general properties (reachability properties, deadlock freeness, liveness), and specific properties.

RTL2TPN. is a translator prototype, which implements the translation pattern of Sect. 4. It takes as an input an RT-LOTOS specification and outputs a TPN in the format accepted by TINA. RTL2TPN reuses RTL's parser and type-checker.

5.2 Case studies

```
specification scenario: exit
behavior hide sA, sBC, eAB, eC in
( (
        (sA; delay(3,6)eAB{3}; exit)
   ||| (sBC; delay(3,8)eC{5}; exit)
  )
  |[sBC, eAB]|
  (sBC; delay(3,5)eAB{2}; exit)
)
|[sA,sBC]|
(sA; latency(inf)sBC; exit)
endspec
```

Fig. 11. RT-LOTOS specification of the Multimedia scenario

Multimedia scenario. The author of this multimedia scenario wants to present 3 medias named A, B and C, respectively. The duration of these medias are respectively defined by the following time intervals [3,6], [3,5] and [3,8]. This means that the presentation of the media A will last at least 3 sec and at most 6 sec. From the author's point of view, any duration of the media presentation is acceptable, as long as it belongs to the specified time interval. Besides, the author expresses the following global synchronization constraints:

1) The presentation of medias A and B must end simultaneously.

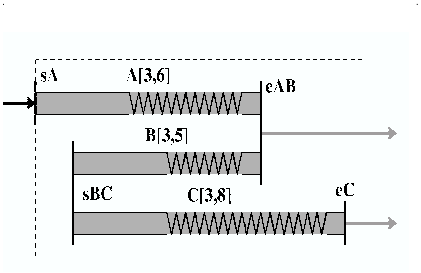
2) The presentation of medias B and C must start simultaneously .

3) The beginning of the multimedia scenario is determined by the beginning of A and its termination is determined by the end of A and B, or by the end of C (cf Fig. 12(a)).

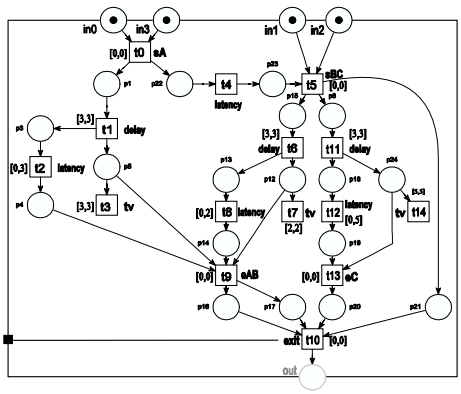
Fig. 12(b) depicts the associated TPN generated by RTL2TPN. For this example TINA builds a graph of 230 classes and 598 transitions.

The scenario is potentially consistent because it exists a path starting by action sA (sA characterizes the beginning of the scenario) and leading to the end of scenario presentation (occurrence of either eAB or eC).

Dinning philosophers. We use one well known multi-process synchronization problem: the Dinning philosophers to check the robustness of the solutions proposed in the paper while facing a state explosion situation. We propose a timed extension of the problem which goes like this: A certain number of philosophers,



(a) The MM constraints .44



(b) The component

sitting around a round table spend their lives thinking and eating. Each of them has in front of him a bowl of rice. Between each philosopher and his neighbor is a chopstick. Each philosopher thinks for a while, then gets hungry and wants to eat. In order to eat, he has to get two chopsticks. Once a philosopher picks his left chopstick, it will take him between 0 and 10 seconds to take the right one. Once he possesses two chopsticks, he can begin eating and it will take him 10 to 1000 seconds[10] to finish eating, then he puts back down his left chopstick and the right one in a delay between 0 to 10 seconds.

```
process Philosopher[think,hungry,b_eat,e_eat,
                    pick_left,put_left,pick_right,put_right]:noexit :=
think; hungry; pick_left;latency(10)pick_right; b_eat;
delay(10,1000)e_eat;put_left;latency(10)put_right;
Philosopher[think,hungry,b_eat,e_eat,pick_left,put_left,pick_right,put_right]
endproc

process Chopstick[pick,put] :noexit :=
pick ; put ; Chopstick[pick,put]
endproc
```

Fig. 12. RT-LOTOS specification of the dinning philosophers

The RT-LOTOS specification of the problem consists in the parallel synchronization of different instances of processes Philosopher and Chopstick[11] (cf Fig 12).

[10] delay and latency may be expressed together by a single syntactic construct delay(dmin, dmax)meaning delay(dmin)latency(dmax-dmin)

[11] All the experiments described in this paper have been performed on a PC with 512 Mo memory and a processor at 3.2 GHz.

Timed Milner scheduler. This is a temporal extension of Milner's scheduler problem [18]. The interesting point about this example is that the timing constraints as introduced in the system, do not increase the state space (compared to the untimed version of the problem). However they complicate the computation of the system's state space.

```
process cycler[g_i,a_i,b_i,g_i+1] : noexit:=

 g_i; latency(10)a_i; delay(10,100)
 (
 (b_i;latency(10)g_i+1;cycler[g_i,a_i,b_i,g_i+1])
  []
 (g_i+1;latency(10)b_i;cycler[g_i,a_i,b_i,g_i+1])
 )

endproc
```

```
Specification SCHEDULER: noexit
Behaviour
Hide g_1..g_n,a_1..a_n,b_1..b_n in
(cycler[g_1,a_1,b_1,g_2]
 |[g_1,g_2]|
 (...
  cycler[g_i,a_i,b_i,g_i+1]
  |[g_i+1]|
  ...
(cycler[g_n,a_n,b_n,g_1] ||| g_1;stop)...)
```

Fig. 13. RT-LOTOS specification of the timed Milner scheduler

Let us consider a ring of n process called Cyclers. A Cycler should cycle endlessly as follows: (i) Be enabled by predecessor at g_i, (ii) after a non deterministic delay between 0 and 10 units of time, receive initiation request at a_i, (iii) after a certain amount of time between 10 and 100 units of time, it receives a termination signal at b_i and enables its successor at g_{i+1}(in either order).

Table 1. Performance comparison of RTL2TPN+TINA vs RTL

	RTL2TPN+TINA		**RTL**	
	Classes/Transitions	CPU(sec)	Classes/Transitions	CPU(sec)
2 **Philosophers**	89/ 150	<1	64/ 120	<1
3	653/ 1536	<1	530/ 1395	3
4	66 251/ 220 488	4	?	>24h
5	876 090/ 3 668 235	103	-	-
2 **Cyclers**	35/ 50	<1	31/ 57	<1
3	81/ 141	<1	58/ 115	<1
4	127/ 232	<1	77/ 153	<1
5	161/ 316	<1	96/ 191	3
6	219/ 414	<1	115/ 229	18
7	232/ 432	<1	134/ 267	97
8	311/ 596	<1	153/ 305	490
9	357/ 687	<1	172/ 343	2363
10	461/ 911	<1	?	>24

The reachability algorithm implemented in RTL is exponential in number of clocks. As the number of philosophers (respectively Cyclers) grows RTL does

not challenge TINA's runtime performances. However the size of state space generated by RTL for the RT-LOTOS specifications is more compact than the one generated by TINA for the associated TPNs issued by RTL2TPN. This is due to a useful but however expensive minimization procedure carried out in RTL. This minimization adapted from [19] permits to consider regions larger than the ones required from a strict reachability point of view, thereby minimizing the number of regions.

6 Related work

Much work has been done on translating process algebras into Petri Nets, by giving a Petri net semantics to process terms [20, 15, 21]. [21] suggests that a good net semantics should satisfy the retrievability principle, meaning that no new "auxiliary" transitions should be introduced in the reachability graph of the Petri net. [20, 15] do not satisfy this criterion. In this paper, we define a one-to-one mapping which is compliant with this strong recommendation.

Untimed models A survey of the literature indicates that proposals for LOTOS to Petri net translations essentially deal with the untimed version of LOTOS [22–27]. The opposite translation has been discussed by [26] where only a subset of LOTOS is considered, and by [28] where the authors addressed the translation of Petri nets with inhibitor arcs into basic LOTOS by mapping places and transitions into LOTOS expressions. [25] demonstrated the possibility to verify LOTOS specifications using verification techniques developed for Petri nets by implementing a Karp and Miller procedure in the LOTOS world.

Among all these approaches, [22, 27] is the only one operating a complete translation of LOTOS (it handles both the control and data parts of LOTOS). Moreover, it just considers regular LOTOS terms, and so do we. The LOTOS to PN translation algorithms of [22, 27] were implemented in the CAESAR tool. Besides the temporal aspects addressed in this paper, a technical difference with [22, 27] lies in the way we structure TPNs. Our solution is based on TPNs components. In our approach, a component may contain several tokens. Conversely, [22, 27] structures Petri nets into units, each of them containing one token at most. This invariant limits the size of markings, and permits optimizations on memory consumption. The counterpart is that [22, 27] use ϵ-transitions. The latter introduces non determinism. They are eliminated when the underlying automaton is generated (by transitive closure). The use of ϵ-transitions may be inefficient in some particular cases, such as the example provided in [29].

The major theoretical study on taking advantage of both Petri nets and process algebras is presented in [12]. The proposed solution is Petri Box Calculus (PBC), a generic model that embodies both process algebra and Petri nets. The authors start from Petri nets to come up with a CCS-like process algebra whose operators may straightforwardly be expressed by means of Petri nets.

Timed models [30] pioneered work on timed enhancements of the control part of LOTOS inspired by timed Petri nets models. [31] defined a mapping from TPNs to TE-LOTOS which makes it possible to incorporate basic blocks specified as 1-bounded TPNs into TE-LOTOS specifications. However, because of the strong time semantics of TPNs (a transition is fired as soon as the upper bound of its time interval is reached unless it conflicts with another one) a direct mapping was not always possible.

A Timed extension of PBC has been proposed in [14]. Although the component model proposed in this paper is not a specification model but an intermediate model used as gateway between RT-LOTOS and TPNs, we find it important to compare our work with [14].

Of prime interest to us is the way [14] introduces temporal constraints in his framework by providing each action with two time bounds representing the earliest firing time and latest firing time. This approach is directly inspired by TPNs, where the firing of actions is driven by necessity. However, a well known issue with this strategy is that it is badly compatible with a compositional and incremental building of specifications. The main difficulty is to compose time intervals when dealing with actions synchronization. The operational semantics of [14] relies on intervals intersection to calculate a unique time interval for a synchronized transition. However, this approach is not always satisfactory (see [13]).

7 Conclusion

Search for efficiency in RT-LOTOS specification verification is the main motivation behind the work presented in this paper. We propose a transformational approach between RT-LOTOS, which is a compositional FDT, and Time Petri Nets, which are not. The semantics of the two FDTS are compared. In order to bridge the gap between RT-LOTOS and TPNs, the latter are embedded into components that may be composed. RT-LOTOS-to-TPN translation patterns are defined in order to match the RT-LOTOS composition operators. The translation has been formally proved to be semantics preserving. The patterns have been implemented in a prototype tool which takes as input an RT-LOTOS specification and outputs a TPN in a format that may be processed by TINA [3]. The benchmarks provided in Section 6 demonstrate the interest of the proposed approach.

One major contribution of the paper is to give RT-LOTOS an underlying semantics expressed in terms of TPNS and to clarify the use of RT-LOTOS operators, in particular the *latency* operator. Discussion in this paper is nevertheless limited to the control part of the RT-LOTOS FDT defined in [2]. We have recently extended our work to the data part of RT-LOTOS. RT-LOTOS specifications will be translated into the new format supported by TINA: Predicates/Actions Time Petri nets. The latter enhance the modelling capabilities of TPNs with global variables associated with the nets together with predicates and actions associated with transitions.

The verification approach developed for RT-LOTOS is being adapted to TURTLE, a real-time UML profile based on RT-LOTOS. We thus expect to develop an interface between the TURTLE toolkit [32] and TINA.

References

1. ISO - Information processing systems - Open Systems Interconnection: LOTOS - a formal description technique based on the temporal ordering of observational behaviour. ISO International Standard 8807:1989, ISO (1989)
2. Courtiat, J.P., Santos, C., Lohr, C., Outtaj, B.: Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. Computer Communications **23**(12) (2000)
3. Berthomieu, B., Ribet, P., Vernadat, F.: The TINA tool: Construction of abstract state space for Petri nets and time Petri nets. Int. Journal of Production Research **42**(14) (2004)
4. Milner, R.: Communications and Concurrency. Prentice Hall (1989)
5. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
6. Courtiat, J.P.: Formal design of interactive multimedia documents. In H.Konig, M.Heiner, A., ed.: Proc. of 23rd IFIP WG 6.1 Int Conf on Formal Techniques for Networked and distributed systems (FORTE'2003). Volume 2767 of LNCS. (2003)
7. Courtiat, J.P., de Oliveira, R.: On RT-LOTOS and its application to the formal design of multimedia protocols. Annals of Telecommunications **50**(11–12) (1995) 888–906
8. Merlin, P.: A study of the recoverability of computer system. PhD thesis, Dep. Comput. Sci., Univ. California, Irvine (1974)
9. Merlin, P., Faber, D.J.: Recoverability of communication protocols. IEEE Transactions on Communications **COM-24**(9) (1976)
10. Berthomieu, B., Menasche, M.: Une approche par énumération pour l'analyse des réseaux de Petri temporels. In: Actes de la conférence IFIP'83. (1983) 71–77
11. Berthomieu, B., Diaz, M.: Modeling and verification of time dependant systems using Time Petri Nets. IEEE Transactions on Software Engineering **17**(3) (1991)
12. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. Monographs in Theoretical Computer Science: An EATCS Series. Springer-Verlag (2001) ISBN: 3-540-67398-9.
13. Sadani, T., Boyer, M., de Saqui-Sannes, P., Courtiat, J.P.: Effective representation of regular RT-LOTOS terms by finite time petri nets. Technical Report 05605, LAAS/CNRS (2006)
14. Koutny, M.: A compositional model of time Petri nets. In: Proc. of the 21st Int. Conf. on Application and Theory of Petri Nets (ICATPN 2000). Number 1825 in LNCS, Aarhus, Denmark, Springer-Verlag (2000) 303–322
15. Taubner, D.: Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. Number 369 in LNCS. Springer-Verlag (1989)
16. Yi, W.: Real-time behaviour of asynchronous agents. In: Proc. of Int. Conf on Theories of Concurrency: Unification and Extension (CONCUR). Volume 458 of LNCS. (1990)
17. RT-LOTOS: Real-time LOTOS home page. (http://www.laas.fr/RT-LOTOS/)
18. Milner, R.: A calculus of communication systems. Volume 92 of LNCS. (1980)
19. Yannakakis, M., Lee, D.: An efficient algorithm for minimizing real-time transition system. In: Proc. of f the Conf. on Computer-Aided Verification (CAV). Volume 697 of LNCS., Berlin (1993)

20. Goltz, U.: On representing CCS programs by finite Petri nets. In: Proc. of Int. Conf. on Math. Foundations of Computer Science. Volume 324 of LNCS. (1988)
21. Olderog, E.R.: Nets, Terms, and formulas. Cambridge University Press (1991)
22. Garavel, H., Sifakis, J.: Compilation and verification of LOTOS specifications. In Logrippo, L., et al., eds.: Protocol Specification, Testing and Verification, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium, 1990, Ottawa, Ont., Canada, Amsterdam, Netherlands, North-Holland (1990) 379–394
23. Barbeau, M., von Bochmann, G.: Verification of LOTOS specifications: A Petri net based approach. In: Proc. of Canadian Conf. on Electrical and Computer Engineering. (1990)
24. Larrabeiti, D., Quelmada, J., Pavón, S.: From LOTOS to Petri nets through expansion. In Gotzhein, R., Bredereke, J., eds.: Proc. of Int. Conf. on Formal Description Techniques and Theory, application and tools (FORTE/PSV'96). (1996)
25. Barbeau, M., von Bochmann, G.: Extension of the Karp and Miller procedure to LOTOS specifications. Discrete Mathematics and Theoretical Computer Science **3** (1991) 103–119
26. Barbeau, M., von Bochmann, G.: A subset of LOTOS with the computational power of place/transition-nets. In: Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets (ICATPN). Volume 691 of LNCS. (1993)
27. Garavel, H., Lang, F., Mateescu, R.: An overview of cadp 2001. European Association for software science and technology (EASST) Newsletter **4** (2002)
28. Sisto, R., Valenzano, A.: Mapping Petri nets with inhibitor arcs onto basic LOTOS behavior expressions. IEEE Transactions on computers **44**(12) (1995) 1361–1370
29. Sadani, T., Courtiat, J., de Saqui-Sannes, P.: From RT-LOTOS to time Petri nets. new foundations for a verification platform. In: Proc. of 3rd IEEE Int Conf on Software Engineering and Formal Methods (SEFM). (2005)
30. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: Protocol Specification, Testing and Verification X (PSTV), Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol. (1990) 395–408
31. Durante, L., Sisto, R., Valenzano, A.: Integration of time Petri net and TE-LOTOS in the design and evaluation of factory communication systems. In: Proc. of the 2nd IEEE Workshop on Factory Communications Systems (WFCS'97). (1997)
32. Apvrille, L., Courtiat, J.P., Lohr, C., de Saqui-Sannes, P.: TURTLE : A real-time UML profile supported by a formal validation toolkit. IEEE Transactions on Software Engineering **30**(4) (2004)