Translating TPAL Specifications Into Timed-Arc Petri Nets

Valentín Valero, Juan José Pardo, and Fernando Cuartero *

Departamento de Informática
Escuela Politécnica Superior de Albacete
Universidad de Castilla-La Mancha
Campus Universitario s/n. 02071. Albacete, SPAIN
{Valentin.Valero, Juan.Pardo Fernando.Cuartero}@uclm.es

Abstract. It is well known that Petri nets are a very suitable model for the description and analysis of concurrent systems, and several timed extensions of PNs have been defined to specify the behaviour of systems including time restrictions. But some software designers still find it a little difficult to work directly with Petri nets, specially when dealing with large systems, and they prefer to work with a formalism closer to a programming language, such as algebraic specification languages. Our goal, then, in this paper is to present an automatic translation of specifications written in a (timed) algebraic language (TPAL) into a timed model of Petri nets (Timed-Arc Petri nets). In this task we try to exploit as far as possible the special characteristics of Timed-Arc Petri nets, more specifically the presence of *dead tokens*, which allows us to extend in some particular cases the classical translations.

Topics: Relationships between net theory and other approaches, Timed nets and timed process algebras.

1 Introduction

Process algebras and Petri nets have been widely used for the modelling and analysis of concurrent systems. Process algebras are much closer to a programming language, and thus, many software designers work gladly with them. In contrast to these, Petri nets have an easy interpretation, as they have a graphical nature. Furthermore, there is a solid mathematical foundation supporting them, and some techniques for the systematic analysis of properties are firmly established. Thus, some years ago it became evident that an integration of both formalisms would be very useful, because by doing so we could exploit the advantages of both description techniques. Goltz [10] presents a translation of CCS into Petri nets, while Taubner [18] defines such a translation for a more general

* This work has been supported by the CICYT project "Performance Evaluation of Distributed Systems", TIC2000-0701-C02-02.

process algebra, and more recently, a different approach has been introduced, the *Petri Box Calculus* [5], which takes the standard operators that we find in process algebras, applying them to a domain of Petri nets (*Boxes*). One of the main advantages of this latter approach is that we get a compositional behaviour of the resulting nets.

Both formalisms, process algebras and Petri nets, have been extended to expand their application areas to those systems which exhibit a time-dependent behaviour that should be considered both in the modelling and the analysis process, such as distributed systems, communication systems and real-time systems. Many timed extensions of the classical process algebras have been proposed: timed CCS [22], temporal CCS [13], timed CSP [16] and TPAL [14]. We may also find several proposals for timed extensions of Petri nets. A survey of the different approaches to introduce time in Petri nets is presented in [7]. We can identify a first group of models, which assign time delays to transitions, either using a fixed and deterministic value [15, 17, 20] or choosing it from a probability distribution [4]. Other models use time intervals to establish the enabling times of transitions [12]. Finally, we also have some models that introduce time on tokens [1, 2, 6]. In such a case tokens become classified into two different classes: available and unavailable ones. Available tokens are those that can be immediately used for firing a transition, while unavailable tokens cannot. We have to wait a certain period of time for these tokens to become available, although it is also possible for a token to remain unavailable forever (such tokens are said to be *dead*). More recently, Cerone and Maggiolo-Schettini [8] have defined a very general model (statically timed Petri nets), where timing constraints are intervals statically associated with places, transitions and arcs. Thus, models with timing constraints attached only to places, transitions or arcs can be obtained by considering particular subclasses of this general framework.

Timed-Arc Petri nets [6, 21, 11, 19, 3] are a timed extension of Petri nets in which tokens have associated a non-negative real value indicating the elapsed time from its creation (*its age*), and arcs from places to transitions are also labelled by time intervals, which establish restrictions on the age of the tokens that can be used to fire the adjacent transitions. As a consequence of these restrictions some tokens may become *dead*, because they will never be available, since they are too old to fire any transitions in the future. The interpretation and use of Timed-Arc Petri nets can be obtained from a collection of processes interacting with one another according to a rendez-vous mechanism. Each process may execute either local or synchronization actions. Local actions are those that the process may execute without cooperation from another process, and thus in the Petri net model of the whole system they would appear as transitions with a single precondition place, while synchronization actions would have several precondition places, which correspond to the states at which each one of the involved processes is ready to execute the action. Then, each time interval establishes some timing restrictions related to a particular process (for instance the time that a local processing may require). In consequence, the firing of a synchronization action can be done in a time window, which depends on the age of the tokens on its precondition places.

Timed-Arc Petri nets are, therefore, a very appropriate model for the description of concurrent systems with time restrictions, such as manufacturing systems, real-time systems, process control, workflow systems, etc. Moreover, one of the main advantages of timed-arc Petri nets is that it is quite easy to get a timed-arc Petri net modelling a system that has been previously described by a timed process algebra. Then, one of the main goals of this paper is to show how this translation can be made, and how we can benefit from the token aging in this task.

We start, therefore, from an algebraic specification, written in TPAL [14], and we examine how to translate it into a timed-arc Petri net. In order to achieve this goal we have structured the paper as follows: in Section 2 we present timed-arc Petri nets and their semantics; in Section 3 both the syntax and the operational semantics of the timed process algebra (TPAL) are defined; and the translation of TPAL specifications into TAPNs is defined in Section 4. A complete example of a timed algebraic specification, and its corresponding TAPN is presented in Section 5, and finally, in Section 6 the conclusions and some indications about our future work are formulated.

2 Timed-Arc Petri Nets

Timed-arc Petri nets are a timed extension of Petri nets, which have their tokens annotated with an age (a real value indicating the elapsed time from its creation) and arcs connecting places with transitions have associated a time interval, which limits the age of the tokens to be consumed to fire the adjacent transition. In the particular model that we consider in this paper some transitions can be urgent, in the sense that no time can elapse once they are enabled.

Definition 1. (Labelled Timed-arc Petri nets)
Given a finite set of visible actions Act, we define a labelled timed-arc Petri net (LTAPN) as a tuple [1] $N = (S, T, F, \mathit{times}, \lambda)$, where S is a finite set of *places*, T is a finite set of transitions ($S \cap T = \emptyset$), F is the *flow relation* ($F \subseteq (S \times T) \cup (T \times S)$), *times* is a function that associates a closed time interval to each arc (s,t) in F, i.e.: $\mathit{times} : F|_{S\times T} \longrightarrow \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$, and λ is the *labelling function*, $\lambda : T \longrightarrow Act \cup \{\tau\}$.

Thus, transitions can be labelled either with visible actions or with τ. These internal transitions (those labelled with τ) are considered urgent, i.e., no time can elapse when they are enabled, and they must be executed immediately, unless they are in conflict with some other transitions. Intuitively, these τ-transitions correspond to actions that the system can make internally, and therefore, once they are enabled it does not need to wait to execute these actions.

[1] We only consider arcs with weight 1 to simplify some definitions, but the extension to general arcs with greater weights is straightforward.

We will also use the standard notation to denote preconditions and postconditions of places and transitions:

$$^\bullet x = \{y \mid (y, x) \in F\} \qquad x^\bullet = \{y \mid (x, y) \in F\}$$

As previously mentioned, tokens are annotated with real values, so markings are defined by means of multisets on $\mathbb{R}_0^+$. More precisely, a marking M is a function:

$$M : S \longrightarrow \mathcal{B}(\mathbb{R}_0^+)$$

where $\mathcal{B}(\mathbb{R}_0^+)$ denotes the set of finite multisets of non-negative real numbers. Thus, as usual, each place is annotated with a certain number of tokens, but each one of them has associated a non-negative real number (its *age*). We will denote the set of markings of N by $\mathcal{M}(N)$, and using classical set notation, we will denote the number of tokens on a place s by $|M(s)|$.

As initial markings we only allow markings M such that for all s in S, and any $x > 0$ we have $M(s)(x) = 0$ (i.e., the *initial age* of any token is 0). Then, we define *marked labelled timed-arc Petri nets* (MLTAPN) as pairs (N, M), where N is a labelled timed-arc Petri net, and M is an initial marking on it. Given a MLTAPN (N, M) we define $Init(N) = \{s \in S \mid |M(s)| > 0\}$. As usual, from the initial marking we will obtain new markings, as the net evolves, either by firing transitions, or by time elapsing. A labelled timed-arc Petri net with an arbitrary marking can be graphically represented by extending the usual representation of P/T nets with the corresponding time information. In particular we will use the age of each token to represent it. Therefore, MLTAPNs have initially a finite collection of zero values labelling each place. $\square$

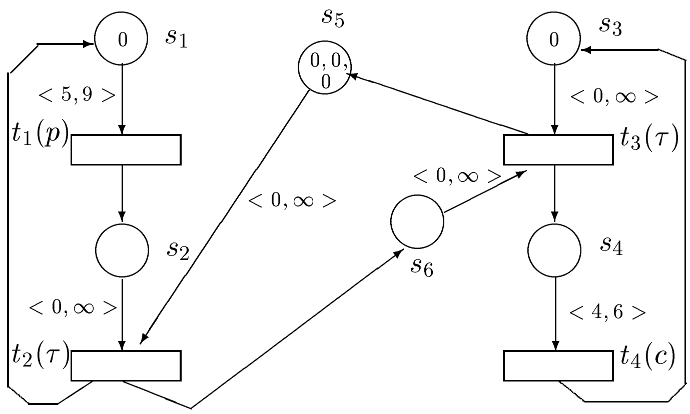

Fig. 1. Timed-arc Petri net modelling the PC-problem

In Fig. 1 we show a MLTAPN modelling a producer/consumer system, where we have represented by transition t_1 the action corresponding to the manufacturing process of the producer (p), which takes between 5 and 9 units of time, and by t_2 the action of including the generated object in the buffer (which is urgent). Notice that the initial tokens on s_5 represent the capacity of the buffer (3), and the arc connecting this place with t_2 is labelled by the interval $< 0, \infty >$, because these tokens can be consumed at any instant in the future. Tokens on s_6 represent the objects on the buffer which have not been yet consumed. Transition t_3 models the (urgent) action of taking out an object from the buffer, which can occur at any instant. Finally, transition t_4 models the processing that makes the consumer for the objects extracted from the buffer (c), and this action takes between 4 and 6 units of time.

Let us observe that if the enabling time for the firing of one of these transitions (t_1 or t_4) expires, the system eventually becomes deadlocked, because we obtain a *dead* token either on s_1 or s_4.

Let us now see how we can fire transitions, and how we model the passage of time.

Definition 2. (Firing rule)
Let $N = (S, T, F, \mathit{times}, \lambda)$ be a LTAPN, M a marking on it, and $t \in T$.

(i) We say that t is *enabled* at the marking M if and only if:

$$\forall s \in {}^\bullet t \ \exists x_s \in \mathbb{R}_0^+ \text{ such that}$$

$$M(s)(x_s) > 0 \wedge x_s \in \mathit{times}(s,t)$$

i.e., on each precondition of t we have some token whose age belongs to $\mathit{times}(s,t)$. As usual, this will be denoted by $M[t\rangle$.

(ii) If t is enabled at M, it can be fired, and by its firing we reach a marking M', as follows:

$$M'(s) = M(s) - C^-(s,t) + C^+(t,s), \ \forall s \in S$$

where both the subtraction and the addition operators work on multisets, and:

- $C^-(s,t) = \begin{cases} \{x_s\} & \text{if } s \in {}^\bullet t, x_s \in \mathit{times}(s,t), \text{ and } M(s)(x_s) > 0 \\ \emptyset & \text{otherwise} \end{cases}$
- $C^+(t,s) = \begin{cases} \emptyset & \text{if } s \notin t^\bullet \\ \{0\} & \text{otherwise} \end{cases}$

Thus, from each precondition place of t we remove a token fulfilling (i), and we add a new token (with age 0) on each postcondition place of t. We denote these evolutions by $M[t\rangle M'$, but it is noteworthy that these evolutions are in general non-deterministic, because when we fire a transition t, some of its precondition places could hold several tokens with different ages that could be used to fire it. Besides, we see that the firing of transitions does not take up any time. Therefore to model the passage of time we need the function *age*, defined below. By applying it we age all the tokens of the net by the same time:

(iii) The function $age : \mathcal{M}(N) \times \mathbb{R}_0^+ \longrightarrow \mathcal{M}(N)$ is defined by:

$$age(M,x)(s)(y) = \begin{cases} M(s)(y-x) & if\ \ y \geq x \\ 0 & otherwise \end{cases}$$

We say that x units of time can elapse in (N, M) if and only if there is no τ-labelled transition enabled at $age(M, y)$, $\forall y \in [0, x]$. In this case, the marking obtained from M after x units of time without firing any transitions will be that given by $age(M, x)$.

□

Although we have defined the evolution by firing single transitions, this can be easily extended to the firing of *steps* or *bags* of transitions; those transitions that could be fired together in a single step could be also fired in sequence in any order, since no *aging* is produced by the firing of transitions. In this way we obtain step transitions that we denote by $M[R\rangle M'$. Finally, by alternating step transitions and time elapsing we can define a timed step semantics, where timed step sequences are those sequences $\sigma = M_0[R_1\rangle_{x_1} M_1 \dots M_{n-1}[R_n\rangle_{x_n} M_n$, where M_i are markings, R_i multisets of transitions and $x_i \in \mathbb{R}_0^+$, in such a way that $M_i[R_{i+1}\rangle M'_{i+1}$ and $M_{i+1} = age(M'_{i+1}, x_{i+1})$. Note that we allow $x_i = 0$ in order to capture the execution in time zero of two causally related steps.

Then, given a MLTAPN (N, M_0), we define $[M_0\rangle$ as the set of reachable markings on N starting from M_0, and we say that (N, M_0) is bounded if for every $s \in S$ there exists $n \in \mathbb{N}$ such that for all $M \in [M_0\rangle$ we have $|M(s)| \leq n$. A token on a place s at a marking M is said to be *dead* if it can never be used to fire any transitions, i.e., it will remain on its place forever, just growing up. Thus, we say that a marking is *dead* when all its tokens are *dead*.

We say that (N, M_0) is safe if $|M(s)| \leq 1$, $\forall s \in P$, for every $M \in [M_0\rangle$, and we will say that it is *soft-safe* if for every reachable marking we have on each place at most one non-dead token.

In a previous paper [19] we have shown that TAPNs without internal transitions (without urgency) have a greater expressiveness than PNs, even though TAPNs are not Turing complete, because they cannot correctly simulate a 2-counter machine. In that paper we proved that reachability is undecidable for this kind of TAPN. Other properties that we have studied in a more recent paper [9] are coverability, boundedness and detection of dead tokens, which are all decidable for non-urgent TAPNs. Decidability of coverability has been also proved in [3] for an extended version of non-urgent TAPNs, in which all arcs can be annotated with bags of intervals in $\mathbb{N} \times \mathbb{N} \cup \{\infty\}$.

However, it is very simple to show that LTAPNs with urgent transitions can simulate two-counter machines, and hence most of the properties become undecidable for them.

3 Syntax and Semantics of TPAL

In this section we briefly present both the syntax and semantics of TPAL, a timed process algebra which is very similar to some other proposals that we may find in the literature [13, 16, 22].

TPAL terms are those defined by the following BNF notation:

$$P ::= stop \mid a <x_1, x_2>; P \mid \tau; P \mid wait(x); P \mid P \,\Box\, P \mid P \parallel_A P \mid P \backslash\, a \mid X \mid \mu X.P$$

where $A \subseteq Act$, $X \in Idf$ (set of identifiers), $a \in Act$, $x, x_1 \in \mathbb{R}_0^+$ and $x_2 \in \mathbb{R}_0^+ \cup \{\infty\}$.

The informal interpretation of the operators follows:

$stop$.- This represents a deadlock, i.e. no action can be executed (but time can elapse).

$a <x_1, x_2>; P$.- The action a can be executed after a delay in the time interval $[x_1, x_2]$ (with respect to the previous action). Once the action a has been executed, the process behaves as P. However, if we do not execute the action and the time expires, we will not be able to execute the action, and the process will have become deadlocked.

$\tau; P$.- The internal action τ is immediately executed, and the process behaves as P.

$wait(x); P$.- The process waits for x units of time, and then it behaves as $\tau; P$. A τ marks the end of the wait, and it guarantees that it finishes in exactly x units of time.

$P_1 \,\Box\, P_2$.- This is essentially the classical external choice operator, but time can only elapse when both components are able to do that.

$P_1 \parallel_A P_2$.- This represents the parallel composition of P_1 and P_2 synchronizing on the actions in A. Again, time can elapse in a parallel composition only when both components can do that.

$P \backslash a$.- All instances of the action a in P are hidden, thus becoming urgent (τ).

$\mu X.P$.- This is the classical recursion operator, for the description of infinite behaviours.

Terms obtained from this syntax will be denoted by TPAL. We will usually restrict it to regular terms, for which terms $\mu X.P$ must be guarded (all appearances of X in P must be prefixed), and there is neither a parallel operator, nor a hiding operator in P. These regular terms will be denoted by RTPAL.

3.1 Operational Semantics

We define an operational semantics for the closed terms of TPAL by a labelled transition system in Plotkin and Milner's style (see Table 1). We have two kinds of transition rules:

$P \underset{x}{\longmapsto} P'$ The process P behaves as P' after x units of time, $x \in \mathbb{R}_0^+$.

$P \xrightarrow{e} P'$ The process P executes the action $e \in Act \cup \{\tau\}$ and then behaves as P' (no time elapses).

With rule R1 we capture that for the process stop the only possible evolution is the passage of time. Rules R2a-c capture the semantics of the timed prefix. Concretely, rule R2c states that once the time to execute an action has expired,

R1) $\mathrm{stop} \underset{x}{\longmapsto} \mathrm{stop} \quad \forall x \in \mathbb{R}_0^+$

R2a) $a<0,x>;P \xrightarrow{a} P \quad \forall x \in \mathbb{R}_0^+$

R2b) $a<x_1,x_2>;P \underset{x'}{\longmapsto} a<x_1 \dot{-} x', x_2 - x'>;P$
$0 \leq x' \leq x_2, \;\; x \dot{-} y = \max\{0, x-y\}$

R2c) $a<0,0>;P \underset{x'}{\longmapsto} \mathrm{stop} \quad x' > 0$

R3) $\tau;P \xrightarrow{\tau} P$

R4a) $\mathrm{wait}(x);P \underset{x'}{\longmapsto} \mathrm{wait}(x-x');P\,,\; 0 \leq x' \leq x$

R4b) $\mathrm{wait}(0);P \xrightarrow{\tau} P$

R5a) $\dfrac{P_1 \xrightarrow{e} P_1'}{P_1 \Box P_2 \xrightarrow{e} P_1'}$ **R5b)** $\dfrac{P_2 \xrightarrow{e} P_2'}{P_1 \Box P_2 \xrightarrow{e} P_2'}$

R6) $\dfrac{P_1 \underset{x_1}{\longmapsto} P_2 \quad P_2 \underset{x_2}{\longmapsto} P_3}{P_1 \underset{x_1+x_2}{\longmapsto} P_3}$ **R7)** $\dfrac{P_1 \underset{x'}{\longmapsto} P_1' \quad P_2 \underset{x'}{\longmapsto} P_2'}{P_1 \Box P_2 \underset{x'}{\longmapsto} P_1' \Box P_2'}$

R8a) $\dfrac{P_1 \xrightarrow{e} P_1' \quad e \notin A}{P_1 \parallel_A P_2 \xrightarrow{e} P_1' \parallel_A P_2}$ **R8b)** $\dfrac{P_2 \xrightarrow{e} P_2' \quad e \notin A}{P_1 \parallel_A P_2 \xrightarrow{e} P_1 \parallel_A P_2'}$

R8c) $\dfrac{P_1 \xrightarrow{a} P_1', \quad P_2 \xrightarrow{a} P_2' \;\; a \in A}{P_1 \parallel_A P_2 \xrightarrow{a} P_1' \parallel_A P_2'}$ **R8d)** $\dfrac{P_1 \underset{x'}{\longmapsto} P_1', \quad P_2 \underset{x'}{\longmapsto} P_2'}{P_1 \parallel_A P_2 \underset{x'}{\longmapsto} P_1' \parallel_A P_2'}$

R9a) $\dfrac{P \xrightarrow{e} P' \quad e \in Act \;\; e \neq a}{P \backslash a \xrightarrow{e} P' \backslash a}$ **R9b)** $\dfrac{P \xrightarrow{a} P'}{P \backslash a \xrightarrow{\tau} P' \backslash a}$

R9c) $\dfrac{P \underset{x}{\longmapsto} P',\; \neg P \xrightarrow{a} \;\wedge \neg[\exists\, x' < x,\; P \underset{x'}{\longmapsto} P''\,,\; P'' \xrightarrow{a}]}{P \backslash a \underset{x}{\longmapsto} P' \backslash a}$

R10a) $\dfrac{P\{\mu x.P/x\} \xrightarrow{e} P'}{\mu x.P \xrightarrow{e} P'}$ **R10b)** $\dfrac{P\{\mu x.P/x\} \underset{x}{\longmapsto} P'}{\mu x.P \underset{x}{\longmapsto} P'}$

Table 1. Operational Semantics of TPAL

we lose the ability to execute that action. Rules R3 and R4a-b capture the urgent character of internal actions, and the semantics of the wait operator. Rules R5a-b give the classical semantics for the external choice, while R7 states that the time can pass in an external choice only when both components are able to do that. Rules R8a-c are the classical ones for the parallel operator, while R8d is similar to R7. Semantics of the hiding operator is captured by rules R9a-c, the latter capturing the fact that once an action has been hidden, it becomes urgent. More precisely, the passage of x units of time for the process $P\backslash a$ is only possible if there is no enabled a-action for all processes P' obtained from P by the passage of x' units of time, with $x' < x$. Finally, rules R10a-b capture the semantics for the recursion operator (the notation $P\{\mu X.P/X\}$ stands for the replacement of each occurrence of X in P by $\mu X.P$).

We will denote the labelled transition system that we may obtain by applying the rules in Table 1 for a process P by $lts(P)$.

4 Net Semantics for TPAL

In this section we define the translation of a TPAL process into a labelled timed-arc Petri net. We first consider the finite case, whithout recursion; the definition is based on the syntactical structure of the given term. For the particular case of the external choice we provide two constructions, essentially depending on the initial parallelism of the components involved. Specifically, for the particular case of both components not having initial parallelism[2], the structure of the obtained net is very simple, and the procedure for doing this is well known. However, we also provide a definition for the external choice when one or even both components have initially a parallel behaviour. For the infinite case we have to deal with recursive terms, so we first present the translation for regular processes, showing later how we can go beyond these terms.

4.1 Finite case

For the finite case we only consider processes without any recursion operators, these terms will be denoted by *fTPAL*.

Stop The *LTAPN* of *stop* only consists of a single place with a single token, whose initial age is 0:

$$N[\![stop]\!] = (\{s_0\}, \emptyset, \emptyset, \emptyset, \emptyset)$$
$$M_0(s_0) = \{0\}$$

Timed and urgent prefix Let $P = a < x_1, x_2 >; P_1 \in fTPAL$, and $N_1 = ((S_1, T_1, F_1, times_1, \lambda_1), M_{1,0})$ the *MLTAPN* of P_1. The net of P is defined by including a new place, and a new transition, which is connected with every initial place of N_1:

$$N = ((S, T, F, times, \lambda), M_0)$$

where:

$$\begin{array}{ll}
S = & \{s_0\} \cup S_1\,, \;\; s_0 \notin S_1 \\
T = & \{t_0\} \cup T_1\,, \;\; t_0 \notin T_1 \\
F = & F_1 \cup \{(s_0, t_0)\} \cup \{(t_0, s_{1,0}) \mid s_{1,0} \in Init(N_1)\} \\
times(s,t) = & \begin{cases} times_1(s,t) \text{ if } (s,t) \in F_1 \\ (x_1, x_2) \qquad \text{if } (s,t) = (s_0, t_0) \end{cases} \\
\lambda(t) = & \begin{cases} \lambda_1(t) \text{ if } t \in T_1 \\ a \qquad \text{if } t = t_0 \end{cases} \\
M_0(s) = & \begin{cases} \{0\} \text{ if } s = s_0 \\ \;\emptyset \;\; \text{otherwise} \end{cases}
\end{array}$$

[2] In the sense that neither of them has more than one initial place.

For the process $\tau; P$ the net has the same structure, the only changes being that now t_0 is labelled with τ, and the time interval of the arc (s_0, t_0) is $[0, 0]$. The net for the process $wait(x); P$ has got this structure too, but the time interval is now $[x, x]$.

External choice Let $P_1, P_2 \in fTPAL$ be two finite processes, and N_1, N_2 the nets associated with them, with $N_i = ((S_i, T_i, F_i, times_i, \lambda_i), M_{i,0})$, $i = 1, 2$, and $S_1 \cap S_2 = \emptyset$, $T_1 \cap T_2 = \emptyset$. For the definition of $N[\![P_1 \Box P_2]\!]$ we distinguish two cases, depending on the number of places initially marked in N_1 and N_2. If both components have only one initial place we may construct the net just by adding a new place, and removing the initial places of both nets; this new initial place is then connected with the postcondition transitions of the removed places. However, if at least one of the components involved has several initial places we need a construction which is a little more sophisticated, by adding some new places, each one corresponding to a pair of initial transitions of N_1 and N_2 respectively.

$$N[\![P_1 \Box P_2]\!] = \begin{cases} N_1 \Box_1 N_2 \text{ if } |Init(N_1)| = 1 \wedge |Init(N_2)| = 1 \\ N_1 \Box_2 N_2 \text{ otherwise} \end{cases}$$

where $\Box_1, \Box_2$ are both net operators, which are defined as follows:

- $N_1 \Box_1 N_2 = ((S, T, F, times, \lambda), M_0)$, where:

$$\begin{array}{l} S = \{s_0\} \cup S_1 \cup S_2 \setminus (Init(N_1) \cup Init(N_2)) \\ T = T_1 \cup T_2 \\ F = \{(s_0, t) \,|\, (s_i, t) \in F_i, s_i \in Init(N_i), i = 1, 2\} \cup \\ \qquad (F_1 \cup F_2) \setminus \{(s_i, t_i) \,|\, (s_i, t_i) \in F_i, s_i \in Init(N_i), i = 1, 2\} \\ times(s, t) = \begin{cases} times_i(s, t) \text{ if } (s, t) \in F_i,\ i = 1, 2 \\ times_i(s_i, t) \text{ if } s = s_0,\ s_i \in Init(N_i),\ (s_i, t) \in F_i, i = 1, 2 \end{cases} \\ \lambda(t) = \lambda_i(t)\,, \text{ where } t \in T_i,\ i = 1, 2 \\ M(s) = \begin{cases} \emptyset \text{ if } s \neq s_0 \\ \{0\} \text{ if } s = s_0 \end{cases} \end{array}$$

- $N_1 \Box_2 N_2 = ((S, T, F, times, \lambda), M_0)$, where:

$$\begin{array}{l} S = S_{12} \cup S_1 \cup S_2,\ with\ S_{12} = \{(t_{i_1}, t_{i_2}) \,|\, t_{i_1} \in Init(N_1)^\bullet,\ t_{i_2} \in Init(N_2)^\bullet\} \\ T = T_1 \cup T_2 \\ F = F_1 \cup F_2 \cup \{((t_{i_1}, t_{i_2}), t_{i_1}),\ ((t_{i_1}, t_{i_2}), t_{i_2}) \,|\, (t_{i_1}, t_{i_2}) \in S_{12}\} \\ times(s, t) = \begin{cases} times_i(s, t) & \text{if } (s, t) \in F_i,\ i = 1, 2 \\ (0, MW(P_1 \Box P_2)) & \text{if } s = (t_{i_1}, t_{i_2}) \in S_{12} \end{cases} \\ \lambda(t) = \lambda_i(t)\,, \text{ where } t \in T_i,\ i = 1, 2 \\ M(s) = \begin{cases} \{0\} \text{ if } s = (t_{i_1}, t_{i_2}) \in S_{12} \vee (s \in Init(N_i), i = 1, 2) \\ \emptyset \text{ otherwise} \end{cases} \end{array}$$

where $MW(P)$ is defined as follows:

$$\begin{array}{ll} MW(stop) = 0 & MW(a < x_1, x_2 >; P) = x_2 \\ MW(\tau; P) = 0 & MW(wait(x); P) = x \\ MW(P_1 \Box P_2) = Max(MW(P_1), MW(P_2)) & MW(P \backslash a) = MW(P) \\ MW(P_1 \|_A P_2) = Max(MW(P_1), MW(P_2)) & \end{array}$$

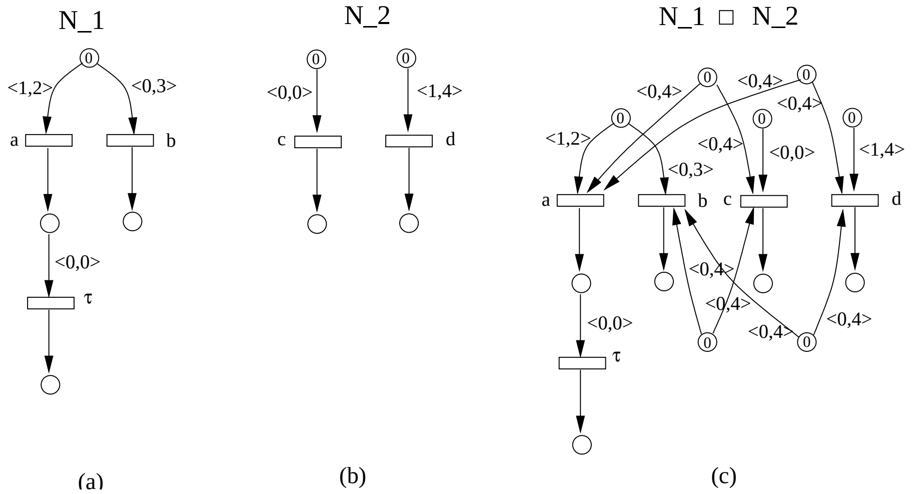

Fig. 2. Timed-Arc Petri nets of P_1, P_2 and $P_1 \Box P_2$.

This function gives us an upper bound for the maximum age that a non-dead token may have on any initial place (of course, it could also be infinite).

Notice that $\Box_1$ generates a net whose size is essentially the addition of the sizes of the argument nets (we just remove one place), while $\Box_2$ increases the size, by including some new places and arcs. Specifically, there are $|Init(N_1)^\bullet| \times |Init(N_2)^\bullet|$ new initial places, and $2 \times |Init(N_1)^\bullet| \times |Init(N_2)^\bullet|$ new arcs.

Example 1. Let $P_1 = a <1,2>; \tau; stop \,\Box\, b <0,3>; stop$, and $P_2 = c <0,0>; stop \|_{\emptyset} d <1,4>; stop$. The corresponding MLTAPNs for both processes are shown in Fig.2.a-b, and the net for $P_1 \Box P_2$ is shown in Fig.2.c. □

Hiding Let P be a fTPAL process and N its corresponding MLTAPN. In order to construct the net associated with $P\backslash a$ we just need to change all the labels a in N by τ.

Parallel operator Let $P_1, P_2 \in fTPAL$ be two finite processes, and N_1, N_2 the nets associated with them, with $N_i = ((S_i, T_i, F_i, times_i, \lambda_i), M_{i,0})$, $i = 1, 2$, and $S_1 \cap S_2 = \emptyset$, $T_1 \cap T_2 = \emptyset$. The net associated with $P_1 \|_A P_2$ is defined as follows:

$$N = ((S, T, F, times, \lambda), M_0)$$

where:

$$\begin{aligned} S &= S_1 \cup S_2 \\ T &= ((T_1 \cup T_2) \setminus \{t_i \in T_i \mid \lambda_i(t_i) \in A,\ i = 1, 2\}) \cup \\ &\quad\ \{(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, \lambda_1(t_1) = \lambda(t_2) \in A\} \\ F &= (F_1 \cup F_2)|_{(S \times T) \cup (T \times S)} \cup \{(s_1, (t_1, t_2)) \mid (s_1, t_1) \in F_1\} \cup \\ &\quad\ \{(s_2, (t_1, t_2)) \mid (s_2, t_2) \in F_2\} \cup \{((t_1, t_2), s_1) \mid (t_1, s_1) \in F_1\} \cup \\ &\quad\ \{((t_1, t_2), s_2) \mid (t_2, s_2) \in F_2\} \end{aligned}$$

$$
\begin{aligned}
times(s,t) &= \begin{cases} times_i(s,t) & \text{if } (s,t) \in F_i,\ i=1,2 \\ times_1(s_1,t_1) & \text{if } t=(t_1,t_2) \wedge (s_1,t_1) \in F_1 \\ times_2(s_2,t_2) & \text{if } t=(t_1,t_2) \wedge (s_2,t_2) \in F_2 \end{cases} \\
\lambda(t) &= \begin{cases} \lambda_i(t) & \text{if } t \in T_i \\ \lambda_1(t_1) & \text{if } t=(t_1,t_2) \end{cases} \\
M_0(s) &= \quad M_{i,0}(s) \text{ where } s \in S_i,\ i=1,2
\end{aligned}
$$

Thus, we join in a single transition all the pairs of transitions (t_1, t_2) which carry the same label in A, taking as preconditions (postconditions) of these new transitions the union of the preconditions (postconditions) of the two transitions involved. Consequently, with this definition the size of the generated nets will be in general a little smaller than the addition of the sizes of both argument nets.

Lemma 1. For any finite TPAL process P, $N[\![P]\!]$ is safe.

Proof: Immediate. $\square$

We now state the equivalence between the operational semantics and the net semantics for finite processes. More specifically, the transition system associated with a finite TPAL process is *bisimilar* to the reachability graph that we may construct for the associated MLTAPN.

Theorem 1. Let $P \in fTPAL$ be a finite process and (N, M_0) its corresponding MLTAPN, according to the previous construction. Then, the labelled transition system associated with P, $lts(P)$, and the reachability graph of (N, M_0) are bisimilar, i.e., there is a relation $\rho \subseteq fTPAL \times \mathcal{M}(N)$ such that:

1. $(P, M_0) \in \rho$.
2. $\forall (Q, M) \in \rho$:
   (a) If $Q \underset{x}{\longmapsto} Q'$, then (N, M) can age by x units of time, and $(Q', age(M, x)) \in \rho$.
   (b) If $Q \stackrel{a}{\longrightarrow} Q'$, then $\exists t \in T, \lambda(t) = a$, such that $M[t\rangle M'$, with $(Q', M') \in \rho$.
   (c) If (N, M) can age by x units of time, then there is a transition $Q \underset{x}{\longmapsto} Q'$, with $(Q', age(M, x)) \in \rho$.
   (d) If $M[t\rangle M'$, then there is a transition $Q \stackrel{\lambda(t)}{\longrightarrow} Q'$, with $(Q', M') \in \rho$.

Proof: By structural induction, the base case (*stop*) is immediate. For the timed prefix, the urgent action prefix and the *wait* operator the reasoning is very similar, just adding some new elements (corresponding to the initial situation) to the relation ρ_1 that the induction hypothesis provides us. For instance, for $P = \tau; P_1$ we take $\rho = \{(\tau; P, M_0)\} \cup \{(Q, M) \mid M|_{Init(N[\![P]\!])} = \emptyset,\ (Q, M|_{S_1}) \in \rho_1\}$, where S_1 is the set of places of $N[\![P_1]\!]$.

For the external choice we have two constructions, so ρ must be defined in both cases. The definition for $\square_1$ is quite easy, so we only present here the definition corresponding to $\Box_2$, for which we take:

$$\begin{array}{l}\rho = \{(P_1 \Box P_2, M_0)\} \cup \{(P_1' \Box P_2', age(M_0,x)) \mid P_1 \Box P_2 \xrightarrow[x]{} P_1' \Box P_2',\ x > 0\} \cup \\ \quad \{(Q,M) \mid M|_{S_2 \setminus Init(N_2)} = \emptyset,\ (Q, M|_{S_1}) \in \rho_1,\ (\exists R_1 \subseteq Init(N_1)^\bullet, R_1 \neq \emptyset, \\ \quad\ \forall t_1 \in R_1,\ \forall t_2 \in Init(N_2)^\bullet,\ M(t_1,t_2) = \emptyset) \wedge \exists x \geq 0,\ ((\forall s_2 \in Init(N_2), \\ \quad\ M(s_2) = \{x\}) \wedge (\forall s \in Init(N[\![P]\!]) : (M(s) = \{x\} \vee M(s) = \emptyset)) \wedge \\ \quad\ (\forall s_1 \in Init(N_1),\ M|_{S_1}(s_1) \neq \emptyset \Rightarrow \forall t_1 \in s_1^\bullet, \forall t_2 \in Init(N_2)^\bullet, \\ \quad\ M(t_1,t_2) = \{x\}))\} \ \cup \\ \quad \{(Q,M) \mid M|_{S_1 \setminus Init(N_1)} = \emptyset,\ (Q, M|_{S_2}) \in \rho_2,\ (\exists R_2 \subseteq Init(N_2)^\bullet, R_2 \neq \emptyset, \\ \quad\ \forall t_2 \in R_2, \forall t_1 \in Init(N_1)^\bullet,\ M(t_1,t_2) = \emptyset) \wedge \exists x \geq 0,\ ((\forall s_1 \in Init(N_1), \\ \quad\ M(s_1) = \{x\}) \wedge (\forall s \in Init(N[\![P]\!]) : (M(s) = \{x\} \vee M(s) = \emptyset)) \wedge \\ \quad\ (\forall s_2 \in Init(N_2),\ M|_{S_2}(s_2) \neq \emptyset \Rightarrow \forall t_2 \in s_2^\bullet, \forall t_1 \in Init(N_1)^\bullet, \\ \quad\ M(t_1,t_2) = \{x\}))\}\end{array}$$

where $N_i = N[\![P_i]\!]$, and S_i is the set of places of $N[\![P_i]\!]$. Notice that once we have fired a first transition, let us say t_1 of N_1, all places (t_1, t_{2_j}) become unmarked, which disables all the initial transitions of N_2, but we may still fire the same initial transitions of N_1 that could be fired taking this net isolated.

For the parallel operator, ρ can be defined as follows:

$$\rho = \{(Q_1 \|_A Q_2, M) \mid (Q_1, M|_{S_1}) \in \rho_1,\ (Q_2, M|_{S_2}) \in \rho_2\}$$

where S_i is the set of places of $N[\![P_i]\!]$, $i = 1, 2$.

And finally, for $P = P_1 \backslash a$ we take $\rho = \{(Q \backslash a, M) \mid (Q, M) \in \rho_1\}$ $\Box$

4.2 Infinite case

We now consider recursive processes of *RTPAL*. Some changes must be made in the definition of the external choice, since for this construction to work properly we require initial places (for both components) not to have any incoming arcs, which can be achieved by *unfolding* the outer recursion that could appear on the external choice components.

Then, $N[\![P_1 \Box P_2]\!] =_{def} N[\![Unfold(P_1)]\!] \Box_i N[\![Unfold(P_2)]\!]$, where either $\Box_1$ or $\Box_2$ are applied according to the same criteria as in the finite case. *Unfold* is defined as follows:

$$\begin{array}{ll}Unfold(stop) = stop & Unfold(a \langle x_1, x_2\rangle; P) = a \langle x_1, x_2\rangle; P \\ Unfold(\tau; P) = \tau; P & Unfold(wait(x); P) = wait(x); P \\ Unfold(P \backslash a) = Unfold(P) \backslash a & Unfold(\mu X.P) = Unfold(P\{\mu X.P/X\}) \\ Unfold(P_1 \Box P_2) = Unfold(P_1) \Box Unfold(P_2) & \\ Unfold(P_1 \|_A P_2) = Unfold(P_1) \|_A Unfold(P_2) & \end{array}$$

Notice that this definition is correct, since we are only dealing with regular processes. When the unfolding is required, the size of the net obtained will be significantly larger than the addition of the sizes of the argument nets, due to the replication of places and transitions.

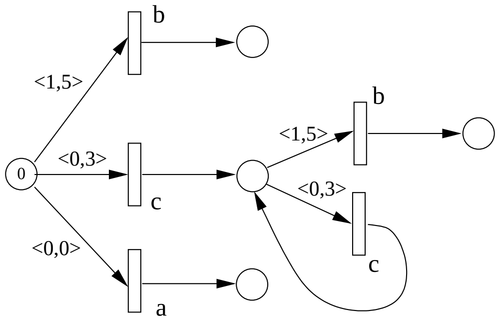

Fig. 3. MLTAPN for the process of example 2

Recursion In order to construct the MLTAPN associated with a regular process $\mu X.P$ we follow two steps:

1. We first construct the net corresponding to $P(X)$, taking for the identifier X a single place as we did for the process *stop*. Notice that as we are only dealing with regular terms, we will not find any terms such as $X \backslash a$, $X \Box Q$ or $X \|_A Q$. Besides, these X-places will have only a single precondition transition, since every appearance of X must be prefixed. We will denote the set of X-places by S^X.
2. Then, we remove these X-places, and we connect their precondition transitions to every initial place of the net (we take $\mathit{Init}(N[\![\mu X.P]\!]) = \mathit{Init}(N[\![P(X)]\!])$).

Example 2. Let $P = a <0,0>; \mathit{stop} \Box \mu X.(b <1,5>; \mathit{stop} \Box c <0,3>; X)$. After the *unfolding* the second component of the outer external choice becomes:

$$b < 1,5 >; \mathit{stop} \Box (c < 0,3 >; \mu X.(b <1,5>; \mathit{stop} \Box c <0,3>; X))$$

Then, by applying the previous definitions we obtain the MLTAPN shown in Fig.3. $\Box$

Lemma 2. For any RTPAL process P, $N[\![P]\!]$ is safe.

Proof: This is a consequence of P being regular. Notice that if an external choice appears within a recursion, both components must be sequential, i.e., only $\Box_1$ will be applicable in this case. This observation is important, because the application of $\Box_2$ within a recursion would be problematic, since some new places are created with $\Box_2$, and some of these new places retain some tokens even after the resolution of the conflict. $\Box$

Theorem 2. Let $P \in \mathit{RTPAL}$ be a regular process and (N, M_0) its corresponding MLTAPN, according to the previous construction. Then, the labelled transition system associated with P, $\mathit{lts}(P)$, and the reachability graph of (N, M_0) are bisimilar.

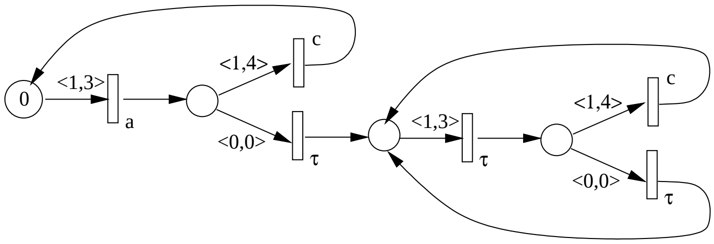

Fig. 4. Translation of a hiding within a recursion

Proof: By structural induction. We only need to explain how to construct ρ for the recursion, since the reasoning for the other operators coincides with that made for the finite case.

Let $P = \mu X.Q_1$, and ρ_1 the bisimulation that we may obtain for $Q_1(X)$ by applying the induction hypothesis (taking the net associated with it, according to the first step of the construction). Then, we can take:

$$\rho = \{(\mu X.Q_1, M_0)\} \cup \{(Q\{\mu X.Q_1/X\}, M|_{S_1 \setminus S_1^X}) \mid (Q, M) \in \rho_1, Q \neq X\}$$

where S_1 is the set of places of $N[\![Q_1(X)]\!]$ and S_1^X is the set of X-places in S_1. Notice that only one X-place can become marked in $N[\![Q_1(X)]\!]$ at the same time, because P is regular; besides, no other place in $N[\![Q_1(X)]\!]$ will be marked in that case. $\square$

4.3 Beyond regular terms

The previous construction for recursive terms works fine for regular terms, but we would like to extend the translation a bit further. A first restriction that can be lifted with some minor changes is the banning of a hiding operator within a recursion. In this case, when we generate the net for $P(X)$ we obtain some (X, A)-places, which are essentially X-places, but all the actions in the set A are hidden for X. Therefore, we can *replicate* $N[\![P(X)]\!]$ for every (X, A)-place, hiding in these nets the actions in A, and then linking the preconditions of each (X, A)-place with the initial places of the corresponding net. Figure 4 shows the MLTAPN corresponding to the process $\mu X.(a < 1,3 >; ((c < 1,4 >; X)\square((a < 0,0 >; X)\backslash a)))$.

Lifting the restriction concerning parallelism within recursion is not possible in general, because we would generate infinite nets. However, in some particular cases we may also apply our construction for recursive processes with parallel subterms; consider for instance the process $P = \mu X.(a < 5,8 >; X \parallel_\emptyset b < 1,2 >; c < 0,1 >; stop)$. By the time that the left-hand side component cycles, we must have the righ-thand side either finished or *dead*. In consequence, the net that we associate to this process (see Fig.5) will not be safe, but *soft-safe*, because some *dead tokens* will remain on the subnet corresponding to the righ-thand side component when we put new tokens on the initial places.

Therefore, we can extend our construction for those recursive terms $\mu X.P$ fulfilling the following conditions:

1. There is no recursive subterm in P.
2. Every occurrence of X in P is prefixed.
3. No hiding [3] is allowed in P.
4. For each subterm $P_1 \|_A P_2$ in P, X only appears in one of these components, either P_1 or P_2. Besides, the *maximum time* for the other component to finish its execution must be finite and smaller than the addition of the *minimum time* for this component (where X appears) to cycle and the *minimum time* to reach that subterm $P_1 \|_A P_2$ from P, i.e., assuming that X appears in P_1:

$$Max\_time(P_2) < Min\_time(P_1, X) + Min\_time(P, P_1 \|_A P_2)$$

   where *Max_time* and *Min_time* can be defined as follows:
   - $Max\_time(stop) = 0$
     $Max\_time(a < x_1, x_2 >; P_1) = x_2 + Max\_time(P_1)$
     $Max\_time(\tau; P_1) = Max\_time(P_1)$
     $Max\_time(wait(x); P_1) = x + Max\_time(P_1)$
     $Max\_time(P_1 \Box P_2) = Max(Max\_time(P_1), Max\_time(P_2))$
     $Max\_time(P_1 \|_A P_2) = Max(Max\_time(P_1), Max\_time(P_2))$
     $Max\_time(P \backslash a) = Max\_time(P)$
   - $Min\_time(Q, Q) = 0$
     In all following cases: $Q \neq$ first argument (on left-hand side of definition)
     $Min\_time(stop, Q) = \infty$
     $Min\_time(X, Q) = \infty$
     $Min\_time(a < x_1, x_2 >; P_1, Q) = x_1 + Min\_time(P_1, Q)$
     $Min\_time(\tau; P_1, Q) = Min\_time(P_1, Q)$
     $Min\_time(wait(x); P_1, Q) = x + Min\_time(P_1, Q)$
     $Min\_time(P_1 \Box P_2, Q) = Min(Min\_time(P_1, Q), Min\_time(P_2, Q))$
     $Min\_time(P_1 \|_A P_2, Q) = Min(Min\_time(P_1, Q), Min\_time(P_2, Q))$
     $Min\_time(P \backslash a, Q) = Min\_time(P, Q)$
5. Every external choice $P_1 \Box P_2$ of type 2 in P satisfies:

$$MW(P_1 \Box P_2) < Min\_time(P_1 \Box P_2, X) + Min\_time(P, P_1 \Box P_2)$$

In our previous example $Max\_time(P_2) = 3$, $Min\_time(P_1, X) = 5$, and $Min\_time(P, P_1 \|_\emptyset P_2) = 0$.

Lemma 3. Let $P \in fTPAL$ be with $Max\_time(P) < \infty$. Then, any reachable marking of $N[\![P]\!]$ in a time greater than $Max\_time(P)$ will be dead.

Proof: It can be easily checked by structural induction that $Max\_time(P)$ is an upper bound for P to stop. Thus, this is an immediate consequence of Theorem 1. □

[3] This restriction is considered again for simplicity, but it could be lifted by finite replication.

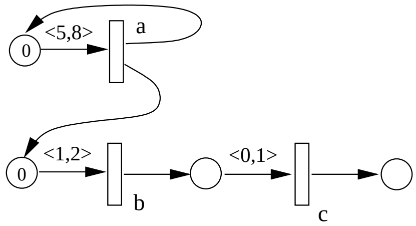

Fig. 5. $N[\![P = \mu X.(a < 5,8 >; X\|_\emptyset b < 1,2 >; c < 0,1 >; stop)]\!]$

Lemma 4. For any TPAL process P fulfilling the conditions 1-5, $N[\![P]\!]$ is soft-safe.

Proof: We can reduce the proof to recursive processes fulfilling the conditions 1-5. It can be checked by structural induction that $Min\_time(P, Q)$ is really a lower bound for the time that P requires to reach the subterm Q. By construction, the only net operators for which some unused tokens can remain on the corresponding subnet when we reach a X-place are $\Box_2$ and $\|_A$:

- With $\Box_2$ we have created some new places (t_1, t_2), and some of them remain marked once the choice has been resolved. Nevertheless, the arcs leaving these places are labelled by $< 0, MW(P_1 \Box P_2) >$, and thus, these tokens will be dead by the time that a new collection of tokens can arrive at these places, because of condition 5. The remaining tokens on these initial places of the subnet corresponding to $P_1 \Box P_2$ will be dead too, because their arcs must be labelled with values $< x_1, x_2 >$, with $x_2 \leq MW(P_1 \Box P_2)$.
- Let $P_1 \|_A P_2$ be a subterm of P, with X in P_1. Then, no token remaining on the places of the subnet $N[\![P_2]\!]$ can be alive when a collection of new tokens arrives at the initial places of the subnet $N[\![P_1 \|_A P_2]\!]$, because of condition 4 and Lemma 3. A finite number of parallel operators can appear in P_1, so we proceed by induction on that number:
  - No parallel operator appears in P_1: in this case the only remaining tokens in $N[\![P_1]\!]$ after a cycle can be those corresponding to an external choice of type 2, for which we have already proved the *soft-safeness*.
  - We now assume that n parallel operators appear in P_1, i.e., we have a subterm $P_{11} \|_B P_{12}$ in P_1, where n_1 parallel operators appear in P_{11} and n_2 in P_{12}, with $n - 1 = n_1 + n_2$. It follows by condition 4 that X can only appear in either P_{11} or P_{12}, so let us assume that it appears in P_{11}. Then, by condition 4 and Lemma 3 we may conclude again the *soft-safeness* for the places of the subnet corresponding to P_{12}, and by the induction hypothesis we conclude the *soft-safeness* for the places of $N[\![P_{11}]\!]$.

$\Box$

Corollary 1. Let $P \in \mathit{TPAL}$ be a process for which every subterm $\mu X.Q$ fulfills the conditions 1-5. Then, the reachability graph of $N[\![P]\!]$ and the labelled transition system associated with P are bisimilar. $\square$

5 Example: Train-Gate Controller

As an illustration, we consider an example of an automatic controller that opens and closes a gate at a railroad crossing. The specification consists of three processes: *TRAIN*, *GATE* and *CONTROL*, which have the following behaviour:

$$\begin{array}{lll} \mathit{TRAIN} & = & \mu X.\mathit{circulating} <10,20>; \mathit{approach} <0,0>; \mathit{in} <5,5>; \mathit{crossing} <4,4>; \\ & & \mathit{exit} <0,0>; X \\ \mathit{GATE} & = & \mu Y.\mathit{lower} <0,\infty>; \mathit{down} <1,2>; \mathit{raise} <0,\infty>; \mathit{up} <1,3>; Y \\ \mathit{CONTROL} & = & \mu Z.\mathit{approach} <0,\infty>; \mathit{reaction} <1,1>; \mathit{lower} <0,0>; \mathit{exit} <0,\infty>; \\ & & \mathit{reaction} <1,1>; \mathit{raise} <0,0>; Z \\ \mathit{TGC} & = & ((\mathit{TRAIN} \|_{\{\mathit{approach},\mathit{exit}\}} \mathit{CONTROL}) \|_{\{\mathit{raise},\mathit{lower}\}} \mathit{GATE}) \backslash \mathit{approach} \backslash \mathit{exit} \\ & & \backslash \mathit{reaction} \backslash \mathit{lower} \backslash \mathit{raise} \end{array}$$

The *train* sends an *approach* signal to the controller 5 minutes before it enters the crossing (*in*). It takes 4 minutes to cross, and then it sends the *exit* signal to the controller. The controller needs 1 minute to respond to the *approach* signal, and then it orders the closing of the gate (*lower*). It also needs 1 minute to respond to the *exit* signal, raising the gate in this case. The *gate* takes between 1 and 2 minutes to close, and from 1 to 3 minutes to open.

The behaviour of the complete system is described by the process *TGC*, in which some actions are hidden, because we want to enforce their execution when they are enabled.

The corresponding MLTAPN for this system, obtained by applying the translation, is depicted in Fig. 6.

6 Conclusions

We have defined a translation of a timed process algebra (TPAL) into timed-arc Petri nets with urgent transitions. This translation has been firstly defined for regular terms of TPAL, but we have also seen how to extend this translation for non-regular terms that fulfill certain conditions. In this latter case we have benefited from the token aging, obtaining *soft-safe* nets, for which every reachable marking has on each place at most one non-dead token.

Our future work on this subject will focus on the analysis of the generated nets, firstly studying techniques of reduction, and then going onto the verification of properties on these nets. We are also developing a tool supporting TPAL, which currently includes a translation of TPAL specifications into a kind of dynamic state graph, so part of our future work will also be to implement the translation into TAPNs in that tool, as well as the simulation and verification of the generated timed-arc Petri nets.

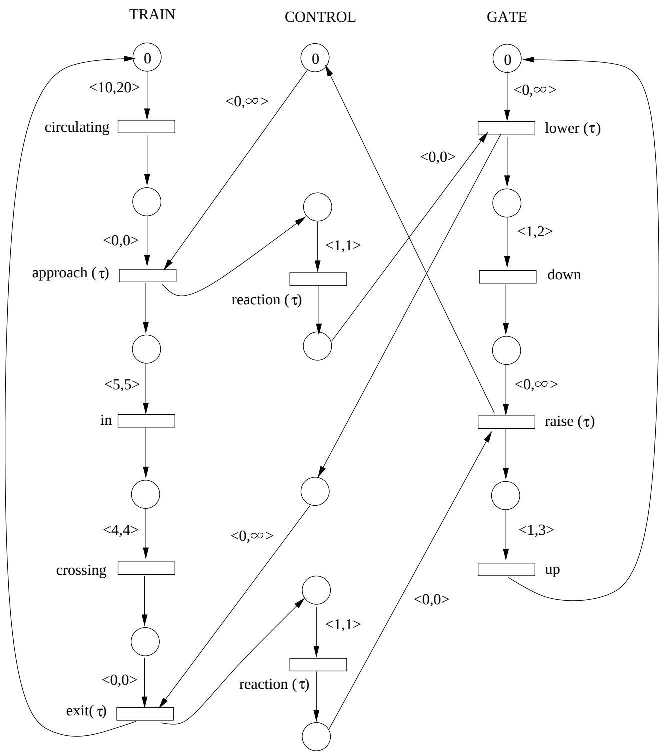

Fig. 6. MLTAPN modelling the Train-Gate Controller

Acknowledgement

The authors would like to thank the reviewers for their comments and suggestions, which have helped to improve this paper significantly.

References

1. W.M.P. van der Aalst. *Interval Timed Coloured Petri Nets and their Analysis.* Lecture Notes in Computer Science, vol. 691, pp. 451-472. 1993.
2. W.M.P. van der Aalst and M.A. Odijk. *Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets.* Real-Time Systems, vol. 9, pp. 241-263. 1995.
3. Parosh A. Abdulla and Aletta Nylén. *Timed Petri Nets and BQOs.* Proc. 22nd International Conference on Theory and Application of Petri Nets. Lecture Notes in Computer Science, vol. 2075, pp. 53-70. 2001.
4. M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte and A. Cumani. *On Petri Nets with Stochastic Timing.* Proc. of the International Workshop on Timed Petri Nets, IEEE Computer Society Press, pp. 80-87. 1985.
5. E. Best, R. Devillers, J. Hall. *The Petri Box Calculus: A New Causal Algebra with Multi-label Communication.* Advances in Petri Nets, 1992. Lecture Notes in Computer Science, vol. 609, pp. 21-69. 1992.

6. T. Bolognesi, F. Lucidi and S. Trigila. *From Timed Petri Nets to Timed LOTOS*. Proceedings of the Tenth International IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification. North-Holland, 1990.
7. Fred D.J. Bowden. *Modelling time in Petri nets*. Proc. Second Australia-Japan Workshop on Stochastic Models. 1996.
8. Antonio Cerone and Andrea Maggiolo-Schettini. *Time-based expressivity of time Petri nets for system specification*. Theoretical Computer Science (216)1-2, pp. 1-53. 1999.
9. D. de Frutos, V. Valero and O. Marroquín. *Decidability of Properties of Timed-Arc Petri Nets*. Proc. ICATPN 2000, Lecture Notes in Computer Science, vol. 1825, pp. 187-206. 2000.
10. U. Goltz. *On representing CCS programs by finite Petri nets*. MFCS, Lecture Notes in Computer Science, vol. 324 (1988), 339-350.
11. Hans-Michael Hanisch. *Analysis of Place/Transition Nets with Timed-Arcs and its Application to Batch Process Control*. Application and Theory of Petri Nets, LNCS vol. 691, pp:282-299. 1993.
12. P. Merlin. *A Study of the Recoverability of Communication Protocols*. PhD. Thesis, Univ. of California. 1974.
13. F. Moller and C. Tofts. *A Temporal Calculus of Communicating Systems*. CONCUR'90. Lecture Notes in Computer Science, vol. 458, pp. 401-415. 1990.
14. J. Pardo, V. Valero, F. Cuartero and D. Cazorla. *Automatic Translation of TPAL specifications into Dynamic State Graphs*. To appear in the proc. of APSEC'01, IEEE Computer Society Press. 2001.
15. C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD. Thesis, Massachusetts Institute of Technology, Cambridge. 1973.
16. G.M. Reed and A.W. Roscoe. *Metric Spaces as Models for Real-Time Concurrency*. Mathematical Foundations of Programming. Lecture Notes in Computer Science, vol. 298, pp. 331-343. 1987.
17. J. Sifakis. *Use of Petri Nets for Performance Evaluation*. Proc. of the Third International Symposium IFIP W.G.7.3., Measuring, Modelling and Evaluating Computer Systems. Elsevier Science Publishers, pp. 75-93. 1977.
18. Dirk Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*. Lecture Notes in Computer Science, vol. 369. 1989.
19. V. Valero, D. de Frutos and F. Cuartero. *On Non-decidability of Reachability for Timed-Arc Petri Nets*. Proc. 8th Int. Workshop on Petri Nets and Performance Models, PNPM'99, pp. 188-196. 1999.
20. V. Valero, D. de Frutos, and F. Cuartero. *Decidability of the Strict Reachability Problem for TPN's with Rational and Real Durations*. Proc. 5th. International Workshop on Petri Nets and Performance Models, pp. 56-65. 1993.
21. B. Walter. *Timed Petri-Nets for Modelling and Analysing Protocols with Real-Time Characteristics*. Proc. 3rd IFIP Workshop on Protocol Specification, Testing and Verification, North-Holland. 1983.
22. Wang Yi. *A Calculus of Real Time Systems*. PhD. Thesis, Chalmers University of Technology. 1991.