Checking Untimed and Timed Linear Properties of the Interval Timed Colored Petri Net Model

Verificación de las propiedades lineales síncronas y asíncronas del Modelo de la Red de Petri Coloreado Intervalo Tiempo

Hanifa Boucheneb
Department of Computer Engineering,
École Polytechnique de Montréal,
P.O. Box 6079, Station Centre-ville, Montréal, Québec
hanifa.boucheneb@polymtl.ca

Article received on July 30, 2004; accepted on December 11, 2006

Abstract
This paper deals with verification of timed and untimed linear properties of the Interval Timed Colored Petri Net model. This model can simulate other timed colored Petri nets and allows describing large and complex real-time systems. We propose here to contract its generally infinite state space into a graph that captures all linear properties of the model. The resulting graph is finite iff, the model is bounded (the set of its reachable markings is finite). In this case, linear properties of the model can be verified on the graph using, for example, the classical linear model checking techniques.
Keywords: Formal methods, model checking, timed models, timed colored Petri net, state space contraction, linear properties.

Resumen
Este artículo se ocupa de la verificación de las propiedades lineales temporizadas y no temporizadas del modelo de redes de Petri coloreadas con intervalos temporizados. Este modelo puede simular otras redes de Petri coloreadas temporizadas y permite describir grandes y complejos sistemas en tiempo real. Nosotros proponemos contraer el espacio generalmente infinito, en un grafo que capture todas las propiedades lineales del modelo. El grafo resultante es finito, si y solamenti si, el modelo tiene límites (el conjunto de sus marcas accesibles es finito). En este caso, las propiedades lineales del modelo se pueden verificar en el grafo resultante, utilizando, por ejemplo, técnicas de comprobación del modelo lineal clásico.
Palabras clave: Métodos formales, comprobación modelo, modelos temporizados, red de Petri coloreada con intervalos temporizados, contracción del espacio del estado, propiedades lineares.

1 Introduction

Verification of concurrent systems is a complex task that requires powerful models and efficient analysis techniques. Model checking is one of the most popular verification techniques of concurrent systems [1, 2, 3, 11, 12, 15]. In this technique, the behavior of a system is represented by a finite transition system (state graph or state space), and the properties to be verified are expressed in either a standard temporal logic (LTL, CTL, CTL*) [1, 2, 12, 21], or in its time extension (MITL, TCTL) [1, 2, 18]. Properties are checked by exploring the state graph (an enumerative method). In theory, the applicability of this method is restricted to finite graphs, but in practice, it also runs up against the state explosion problem. To overcome these limitations, one solution consists in contracting the state graph into a finite and compact structure allowing to verify properties of interest [3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 18, 19, 20, 21].

This paper aims to apply the linear model checking techniques for timed colored Petri nets. Colored Petri nets (CPNs) are successfully used to specify and analyze complex systems. In this model, a color (a value) is associated with each token allowing to make much compact and manageable descriptions. To be able to analyze systems whose behaviors are time dependant, several extensions to time parameter have been proposed for the CPN model (Van der Aalst in [19], Christensen in [10], Pao-Ann Hsiung [16]). In Pao-Ann Hsiung's model, a time interval is associated with each transition specifying its minimal and maximal firing delays. Time intervals of this model have the same semantics as those of Merlin's model (Time Petri Nets) [5]. In Christensen's model, a date is associated with each created token, which indicates when the token will become available. An enabled transition will occur as soon as possible (when all its required tokens become available). The Van der Aalst's model called Interval timed Colored Petri Net (ITCPN) associates with each created token a time interval specifying when the token will become available (the earliest and latest times). As in Christensen's model, an enabled transition will occur as soon as possible. Among these extensions, the model proposed by Van der Aalst seems to be more appropriate since time intervals are associated with tokens instead of transitions or places. But, unlike Pao-Ann Hsiung's model, the Van der Aalst's model does not allow unbounded intervals and then expressing that a created token could be never available (be lost) is not possible. To overcome this limitation, we extend this model by allowing unbounded intervals. With this extension, the model can simulate other timed colored Petri nets and allows describing, in a concise way, large and complex real-time systems.

However, because of time density, the ITCPN state space is generally infinite. Therefore, its analysis by enumeration needs an extra step to contract its state space into a finite graph preserving properties of interest (linear properties: LTL, MITL). We say that a graph preserves linear properties of some model, if we can determine from it all evolutions of the model.

Van der Aalst proposed in [19], a contraction method for the ITCPN model which is "sound" (i.e.: any occurrence sequence in the state space of the model is also possible in the contracted state space) but not "complete" (i.e.: some firing sequence in the contracted state space does not reflect any firing sequence in the model state space). Therefore, the contracted space has not necessarily the same linear properties as the model. Moreover, for models allowing infinite firing sequences, theVan der Aalst's method produces infinite graphs.

We propose here another contraction approach that generates finite graphs for all bounded ITCPNs. Our approach is both "sound" and "complete" and the resulting graph preserves all linear properties of the model. It can be used to verify linear properties of the model. Untimed linear properties can be verified using the standard linear model checking techniques. To verify timed linear properties, we generally need to compute the minimal and maximal executing times of some firing sequences (paths of the graph). We develop here an algorithm that computes for a given path of the graph, its minimal and maximal path times.

Firstly, we give, in section 2, some definitions related to the ITCPN model and its behavior. We show afterwards, in sections 3, 4 and 5, how to contract the ITCPN state space into a graph preserving linear properties of the model. We distinguish two levels of contraction. In the first level (section 3), we agglomerate into one state class, all states reachable by firing the same sequence independently of their firing dates. Section 4 is devoted to the simplification of the firing rule given in section 3. We establish here an attractive characterization of state classes that simplify considerably their computation and their comparison. Afterwards, we show by means of an example that this contraction can produce infinite graphs for some bounded ITCPNs. For further contractions, we propose, in section 5, to relax state classes. In section 6, we show that with this relaxation, we obtain finite graphs for all bounded ITCPNs. Section 7 shows how to compute path times. Section 8 is devoted to an application example.

2 The Interval Timed Colored Petri Net model

The ITCPN model is a colored Petri net (CPN) augmented with time intervals associated with tokens. From the semantic point of view, each created token has a time stamp which can be any value inside its associated interval. The time stamp of a token indicates the delay required for the token to become available.

2.1 Formal definition of the ITCPN model

We introduce here only necessary definitions and notations. For further details, we refer to *[17]* for CPNs and to *[19]* for ITCPNs.

Definition 1: (Time domain, multisets)

- The time domain is the set of all non-negative real numbers plus ∞ , i.e.: $R^+ \cup \{\infty\}$. Note that ∞ is considered here as a particular value which satisfies the following relations: $r+\infty = \infty+r=\infty$, $r \le \infty$, for each r $\in (R^+ \cup \{\infty\})$, and $\infty-\infty = \infty$.
- Let X be a set. A multiset over X is a function N which associates with each element of X, an integer number. It is represented by the formal sum: $\sum_{x\in X} N(x)\bullet x$, where $N(x)$ is the occurrence number of x in N.
- Let X be a set, N_1 and N_2 two multi-sets over X. Operators +, - , $\le$, = on multi-sets are defined as usual:
  - $N_1 + N_2 = \sum_{x\in X} (N_1(x)+N_2(x))\bullet x$.
  - $N_1 \le N_2$ iff, $\forall x\in X,\ N_1(x) \le N_2(x)$.
  - $N_1 = N_2$ iff, $\forall x\in X,\ N_1(x) = N_2(x)$.
  - if $N_1 \le N_2$ then $N_1 - N_2 = \sum_{x\in X} (N_1(x)-N_2(x))\bullet x$.
  - The size of N_1 denoted by $|N_1|$ is: $\sum_{x\in X} N(x)$.

  We denote by X_{MS} the set of all multisets over X, and by 0 or $\varnothing$ the empty multiset.

Definition 2: (**ITCPN model**)

An ITCPN is a tuple $(\Delta, P, T, C, F, TM_0)$ where:

- Δ is a finite set of color sets. Each color set is finite.
- P is a finite and non empty set of places.
- T is a finite set of transitions such that: $(P\cap T = \varnothing)$.
- $Cd: P \rightarrow Powerset(\Delta)^1$. $Cd(p) \in Powerset(\Delta)$, is a finite set of all allowed colors in place p.
- Let CT be the set of all possible colored tokens, i.e.: $CT = \{(p, c) \mid p \in P \wedge c \in Cd(p)\}$ and INT the set of all intervals defined by: $\{[y,z] \in Q^+ \times (Q^+ \cup \infty) \mid y \le z\}$. Note that unlike *Van der Aalst*'s model, we allow here unbounded intervals.

  F is the transition function over T. $F(t): Dom(F(t)) \rightarrow (CT \times INT)_{MS}$, where $Dom(F(t)) \subseteq CT_{MS}$. $F(t)$ specifies which tokens are consumed and produced by firing transition t. It also specifies the time stamp intervals of the created tokens. Each transition is supposed to produce a finite set of tokens.
- TM_0 is the initial marking indicating which tokens are present initially, their colors and their time stamps, i.e.: $TM_0 \in (CT \times (Q^+ \cup \{\infty\})_{MS}$.

2.2 ITCPN behavior

We first explain the behavior of the model, using an example given in *[19]* and reported here in Figure *1*.

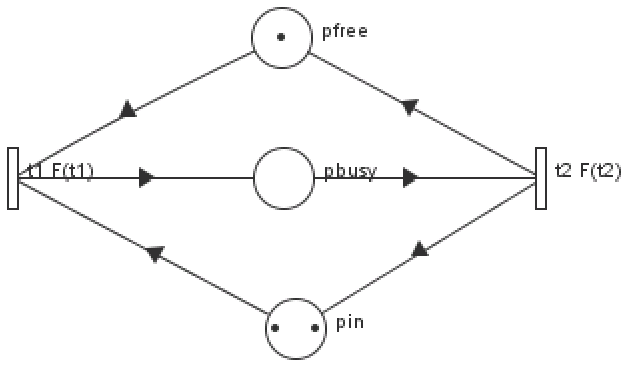

Fig. 1. An ITCPN model of a Jobshop

[1] *Powerset(Δ)* is a set such that $\Delta \subseteq Powerset(\Delta)$ and $\forall A,B \in Powerset(\Delta), A\times B \in Powerset(\Delta)$.

Figure *1* is the graphic representation of an ITCPN composed of three places $\{p_{in}, p_{busy}, p_{free}\}$, two transitions t_1, t_2 and three color sets:

- $M = \{M_1, M_2, \ldots, M_s\}$ associated with place p_{free},
- $J = \{J_1, J_2, \ldots, J_r\}$ associated with place p_{in}, and
- $M \times J$ associated with place p_{busy}.

It represents a jobshop, where jobs of place p_{in} are executed repeatedly. The jobshop is composed of one or several machines. Each machine is represented by a token, which is either in place p_{free} or in place p_{busy}.
Tokens consumed and produced by firing transitions t_1 and t_2 are specified by functions $F(t_1)$ and $F(t_2)$ defined by:

- $\forall j \in J,\ \forall m \in M,$
  $F(t_1)((p_{in}, j) + (p_{free}, m)) = (p_{busy}, (m, j), [1,3]).$

$F(t_1)$ means that transition t_1 consumes one token from each place p_{in} and p_{free}, and produces one token in place p_{busy}. When transition t_1 occurs, the time stamp of the created token can be any value inside interval $[1,3]$.

- $\forall j \in J,\ \forall m \in M,$
  $F(t_2)((p_{busy}, (m, j))) = (p_{free}, m, [2,2]) + (p_{in}, j, [1,1]).$

$F(t_2)$ means that transition t_2 consumes one token from p_{busy} and produces two tokens one in each place p_{in} and p_{free}. When transition t_2 occurs, the time stamps of the tokens created in places p_{free} and p_{in} are respectively *2* and *1*.

Initially, there are three tokens: $TM_0=(p_{free}, M_1,2)+(p_{in},J_1,1)+(p_{in},J_2,2).$

2.2.1 States of an ITCPN

To characterize the model state, we associate with each token a variable, called clock, which measures the time elapsed since the creation of the token. The clock is initialized to *0*, when the token is created. Afterwards, its value increases synchronously with time until its associated token is consumed.

Definition 3: (Timed tokens, states)

- A timed token is a tuple $(p,c,v,[a,b])$ where p is its place, c is its color, v is its clock value (or its clock name) and $[a,b]$ is its time stamp interval.
- A state σ of an ITCPN is a multi-set of timed tokens, i.e.: $\sigma \in (CT \times R^{+} \times INT)_{MS}$.
- The initial state of an ITCPN is obtained from its initial timed marking TM_0 by completing appropriately each token.

Note that there are other state definitions used in *[6, 7, 19]*. In *[6, 7]*, a variable, called delay, is associated with each token indicating the time to wait before the token becomes available. When a token is created, its delay is initialized to any value inside its time stamp interval. Afterwards, its value decreases with time until its associated token disappears. Delays are less appropriate than clocks to verify timed properties. The verification of these properties generally needs to compute some path times. The computation of path times is simpler using clocks, as they measure the time elapsed since they are initialized to zero.

In *[19]*, each token is completed with a date interval indicating the minimal and maximal dates at which the token will become available. If a token is created at date τ, its date interval is $[a+\tau, b+\tau]$, where $[a,b]$ is its time stamp interval. This state definition is not appropriate for enumerative methods, since for models allowing infinite firing sequences, the date will grow infinitely leading to infinite graphs.

2.2.2 State evolution

Initially, the model is in its initial state. Afterwards, its state evolves either by time progressions (clocks increase with time) or by firings.

Definition 4: (State events)

Let σ be a state and $M(\sigma)$ the marking obtained from σ by eliminating all time parameters (the underlying marking).

- Let t be a transition of T. Transition t is enabled for σ iff all tokens required for its firing are present in σ, i.e.: $\exists m \in Dom(F(t)), m \leq M(\sigma)$.
- An event of σ is a pair (t,in) where t is an enabled transition for σ and in is all tokens participating in its enabling.
- Let e be an event of σ. $Jin(e)$ denotes the multi-set of timed tokens required for firing event e. We have: $Jin(e) \leq \sigma$.
- We denote $E(\sigma)$ the set of all events of σ.
- Two events e_1 and e_2 are conflicting for σ iff, $Jin(e_1) + Jin(e_2) < \sigma$. When an event is fired all event conflicting with it are disabled.

In this model, an event shall occur as soon as possible (i.e.: when all required tokens become available). Its firing takes no time but leads to a new marking: consumed tokens disappear and possibly new tokens are created.

Let σ be a reachable state of an ITCPN, $e_f=(t_f,Jin(e_f))$ an event of σ and dv a non-negative real value.

Definition 5: (Time progression)

- Let $(p,c,v,[a,b])$ be a token of σ. The delay interval of this token is the domain of the time required to become available, i.e.: $[max(0,a-v),max(0,b-v)]$.
- Let e be an event of σ. The occurrence delay (i.e.: the firing delay) of e is the delay required for all tokens of Jin(e) to become available. The minimal and maximal firing delays of e denoted respectively by $FD_{min}(e)$ and $FD_{max}(e)$ are $max(0, max_{(p,c,v,[a,b]) \in Jin(e)} (a - v))$ and $max_{(p,c,v,[a,b]) \in Jin(e)} (b-v)$. From the semantic of the model (an event shall occur as soon as possible), there is at least one token $(p,c,v,[a,b])$ in $Jin(e)$ such that $v \leq b$ holds. By convention, if $Jin(e)$ is empty, $FD_{min}(e)$ and $FD_{max}(e)$ are equal to zero.
- A time progression of dv units can occur from the state σ (without any firing) iff, dv is smaller or equal to the maximal firing delays of all events, i.e.: $dv \leq min_{e \in E(\sigma)} FD_{max}(e)$. By convention, if $E(\sigma)$ is empty, $(min_{e \in E(\sigma)} FD_{max}(e))$ is equal to ∞. This condition of time progression is denoted $\sigma \rightarrow_{dv}$.
- After this time progression, the clock of each token $(p,c,v,[a,b])$ of σ increases by dv time units. We denote $(\sigma)_{+dv}$ the reached state: $\sum_{(p,c,v,[a,b]) \in \sigma} \sigma(p,c,v,[a,b]) \bullet (p,c,v+dv,[a,b])$.

We write $\sigma \rightarrow_{dv} \sigma'$ iff the state σ' is reachable from σ by time progression of dv units.

Definition 6: (Event firing)

- Event e_f can occur immediately from σ (without any progression of time) iff, its minimal firing delay is equal to 0, i.e.: $FD_{min}(e_f) = 0$. This immediate firing condition is denoted by $\sigma \rightarrow_{ef}$.
- If e_f can occur immediately from σ, its occurrence is instantaneous and leads to the state $\sigma' = \sigma - Jin(e_f) + Jout(e_f)$, where $Jout(e_f)$ is obtained from $F(t_f)(M(Jin(e_f)))$ by completing each token with the initial value of its clock (i.e.: 0). Recall that $F(t_f)$ is the transition function of t_f. This immediate firing is denoted by $\sigma \rightarrow_{ef} \sigma'$.
- Event e_f can occur from σ after possibly some progression of time iff, its minimal firing delay is not greater than the maximal firing delays of all other events, i.e.: $(FD_{min}(e_f) \leq min_{e \in E(\sigma)} (FD_{max}(e)))$.
- If e_f can occur from σ, it will occur after any delay dv inside interval $[FD_{min}(e_f), min_{e \in E(\sigma)} FD_{max}(e)]$. Its occurrence is instantaneous but leads to the state σ': $\sigma' = (\sigma - Jin(e_f))_{+dv} + Jout(e_f)$.
- An evolution of σ is a sequence of events and time progressions that can occur successively from σ. Evolutions of an ITCPN are those of its initial state.

Note that for the ITCPN model, the evolution of a state σ depends only on its underlying untimed tokens $M(\sigma)$ and the delay intervals of its tokens.

Example 1: (State evolution)

Consider the previous model (Figure *1*). Its initial state σ_0 consists of three tokens: $(p_{free},M_1,0,[2,2])+(p_{in},J_1,0,[1,1])+(p_{in},J_2,0,[2,2])$.

The first and the third one will become available after two time units while the second one will become available after one time unit. State σ_0 has two events:
$e_1= (t_1,(p_{free}, M_1, 0, [2,2]) + (p_{in}, J_1, 0,[1,1]))$ and $e_2= (t_1, (p_{free}, M_1,0, [2,2]) + (p_{in}, J_2,0,[2,2]))$.
Their minimal and maximal firing delays are:
$FD_{min}(e_1)= FD_{max}(e_1)= max(0,2-0,1-0)$ and $FD_{min}(e_2)= FD_{max}(e_2)=max(0,2-0,2-0)$.

Both events can occur from the initial state after *2* time units. After *2* time units, the model will reach the state: $(p_{free},M_1,2,[2,2]) + (p_{in},J_1,2,[1,1]) + (p_{in},J_2,2,[2,2])$.

The occurrence of event e_1 leads to the state $\sigma_1= (p_{in}, J_2,2,[2,2]) + (p_{busy}, (M_1,J_1),0,[1,3])$.

The occurrence of event e_2 leads to the state σ_2 = (pin, J1,2, [1,1]) + (pbusy, (M1,J2),0,[1,3]).

Definition 7: (State space)
The state space of an ITCPN is the graph of its evolutions. It is defined as a tuple $(SS,\rightarrow, \sigma_0)$ where:

- $\sigma_0 \in SS$ is the initial state;
- $\rightarrow \subseteq (SS \times (EE \cup R^+) \times SS)$ is the transition relation defined in Definitions *5* and *6*, EE is the set of all events;
- $SS = \{ \sigma \mid \sigma_0 \rightarrow^* \sigma\}$, where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$.

Because of time density, the state space of the ITCPN model is generally infinite and then not useful for enumerative analysis methods. *Van der Aalst* proposed in *[19]* a contraction method for the ITCPN state space which is "sound" but not "complete". This is due to the fact that this method “forgets” the occurrence time to memorize only intervals, i.e.: a state class (a set of states) is defined as a multiset of triplets of the form (place, color, interval). Consequently, for event producing several tokens, the dependencies (relations binding intervals) are lost and resulting classes may contain unreachable states (leading sometimes to unreachable markings). For instance, consider the model shown in Figure 2. We suppose that the color domain of each place is $\{e\}$, the initial state is $(p_0,e,0)$ and transition functions are defined as follows:

- $F(t_0)((p_0, e)) = (p_1, e,[0,2])$.
- $F(t_1)((p_1, e)) = (p_2, e,[1,2]) + (p_3,e, [1,3])$.
- $F(t_2)((p_2, e)) = F(t_3)((p_3, e)) = 0$.

Using the *Van der Aalst*'s method, the firing of transition t_0 leads to the state class $(p_1,e,[0,2])$. The firing of t_1 from this class produces the class $(p_2,e,[1+0,2+2]) + (p_3,e,[3+0,4+2])$ where states $(p_2,e,1) +(p_3,e,6)$ and $(p_2,e,4) +(p_3,e,3)$ are represented but not reachable. From the former state, transition t_2 is fired before t_3 ($1 < 6$) while from the second, transition t_3 is fired before t_2 ($3 < 4$).
Van der Aalst's method states that both markings (p_2,e) and (p_3,e) are reachable which is in fact wrong. The reason is that after firing transition t_1, the created token (p_2,e) becomes available before token (p_3,e) $([1,2]<[3,4])$.
Therefore, transition t_3 cannot be fired before t_2. The firing of t_2 leads to the marking (p_3,e). Marking (p_2,e) is then not reachable. Due to these represented but unreachable states (markings), the resulting graph has not necessarily the same properties as the initial model. Moreover, for some bounded ITCPNs allowing infinite firing sequences, *Van der Aalst*'s method produces infinite graphs.

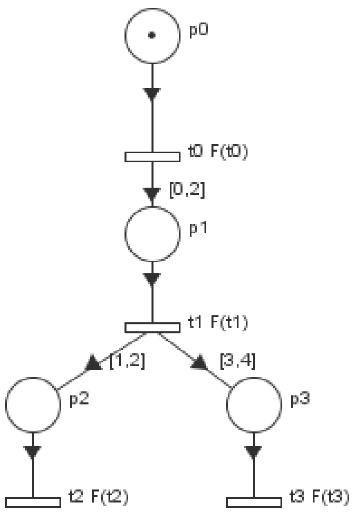

Fig. 2. ITCPN used to explain the Van *der Aalst*'s method

We propose, in the following, another approach which is both "sound" and "complete". In addition, as we will show, our approach generates finite graphs for all bounded ITCPNs.

3 Agglomerating states reachable by the same firing sequence

Separating time progressions and event firings is not appropriate for enumerative analysis. To contract the ITCPN state space, we first combine time progressions with event firings and then agglomerate, into one state class, all states reachable by firing the same sequence of events. The resulting graph called state class graph can be defined as follows.

Definition 8: (State class graph)

Formally, the state class graph of an ITCPN is a structure (CC, α, α_0) where:

- $\alpha_0 \in CC$; α_0 contains only the initial state of the model;
- $\alpha \subseteq (CC \times EE \times CC)$ is the transition relation between state classes defined by:
  $\forall(\alpha, e, \alpha') \in (CC \times EE \times CC)$, $(\alpha \alpha_e \alpha')$ iff
  $(\exists\, \sigma \in \alpha, \exists\, dv \geq 0, \exists\, \sigma' \in \alpha', \sigma \rightarrow_{dv} (\sigma)_{+\delta\varpi} \rightarrow_e \sigma') \wedge \alpha' = \{\sigma' \mid \exists\, \sigma \in \alpha, \exists\, dv \geq 0, \exists\, \sigma' \in \alpha', \sigma \rightarrow_{dv} (\sigma)_{+\delta\varpi} \rightarrow_e \sigma'\}$.
- $CC = \{\alpha \mid \alpha_0\, \alpha^*\, \alpha\}$, where α^* is the reflexive and transitive closure of α.

3.1 Characterization and computation of state classes

A state class agglomerates all states reachable by the same firing sequence. The initial state class consists of the initial state of the model. We can characterize a state class by a marking and a logical formula as follows.

Definition 9: (State class)

A state class α can be defined as a pair (SM, FT) where:

- *SM* is a multi-set of timed tokens where clock values are replaced by clock names (one distinct clock name per token).
- *FT* is a logical formula which characterizes the clock valuations of all states agglomerated in the class α. Each clock valuation corresponds to a state of α.

Starting from the initial class, the successor state classes can be computed using the following firing rule. Events and functions FD_{min}, FD_{max}, *Jin* and *Jout* are defined as in section *2.2.2* except that clock values are replaced by clock names.

Proposition 1: (Firing rule)

Let $\alpha=(SM,FT)$ be a state class and e_f one of its events.

- Event e_f can occur from α (i.e. $\alpha\, \alpha_{e_f}$) iff:
  $FT \wedge (FD_{min}(e_f) \leq min_{e \in E\{SM\}} FD_{max}(e))$.
- Suppose that event e_f can occur from the state class α. Its occurrence leads to the state class α' = (SM',FT'), where: $SM' = SM - Jin(e_f) + Jout(e_f)$ and FT' is computed in four steps:
  - Initialize FT' with
    $FT \wedge FD_{min}(e_f) \leq dh \leq \quad min_{e\, \in E\,(SM)}\, (FD_{max}(e))$;
  - Add for each clock h a new variable $\underline{h}$ and the constraint $\underline{h}=h+dh$;
  - Add for each token $(p,c,h,[a,b])$ created by e_f, i.e.: $((p,c,h,[a,b]) \in Jout(e_f))$, the constraint $\underline{h} = 0$;
  - Eliminate by substitution all variables h, dh and clocks of all tokens consumed by event e_f (i.e.: tokens of $Jin(e_f)$). Afterwards, rename each variable $\underline{h}$ in h.

Proof: The firing condition is immediate from the Definition *4*. The successor class can be computed in four steps. Starting from the firing condition, we first introduce the variable *dh* representing the time progression before firing e_f. As clocks increase with time progression, we add for each clock *h* a new variable $\underline{h}$ and the constraint $\underline{h} =h+dh$. Afterwards, we add the time constraints of all newly created tokens. Their clocks are initially equal to *0*. At the last step, we eliminate by substitution old variables h, dh and clocks of all tokens consumed by e_f. The old name of each remaining variable $\underline{h}$ is then restored (i.e.: $\underline{h}$ is renamed in h).

Example 2: (Applying the firing rule)
The initial state class of the model in Figure *1* is $\alpha_0 = (SM_0, FT_0)$, where:
$SM_0 = (p_{free},M_1,h_1,[2,2]) + (p_{in},J_1, h_2, [1,1]) + (p_{in},J_2, h_3, [2,2])$ and $FT_0 = (h_1 = 0 \wedge h_2 = 0 \wedge h_3 = 0)$.

The state class α_0 has two events:
$e_1= (t_1, (p_{free}, M_1, h_1,[2,2]) + (p_{in}, J_1, h_2,[1,1]))$ and $e_2= (t_1, (p_{free}, M_1, h_1, [2,2]) + (p_{in}, J_2, h_3, [2,2]))$.

Event e_1 can occur from the state class α_0 because the following formula is consistent:
$h_1=0 \wedge h_2=0 \wedge h_3=0 \wedge\ max(0,2-h_1,1-h_2)\leq min(max(2-h_1,1-h_2),max(2-h_1,2-h_3))$.

Its occurrence leads to the class $\alpha_1 = (SM_1,FT_1)$ where:
$SM_1 = (p_{in}, J_2, h_3,[2,2]) + (p_{busy}, (M_1, J_1), h_4,[1,3])$ and
FT_1 is computed as follows:

- Initialize FT_1 with: $h_1=0 \wedge h_2=0 \wedge h_3=0 \wedge\ max(0,2-h_1,1-h_2) \leq dh \leq\ min(max(2-h_1,1-h_2),max(2-h_1,2-h_3))$.
- Add for each clock h a new variable $\underline{h}$ and the constraint $\underline{h}=h+dh$:
  $h_1=0 \wedge h_2=0 \wedge h_3=0 \wedge\ max(0,2-h_1,1-h_2) \leq dh \leq min(max(2-h_1,1-h_2),max(2-h_1,2-h_3)) \wedge$
  $\underline{h_1}= h_1+dh \wedge \underline{h_2}=h_2+dh \wedge \underline{h_3}=h_3+dh$.
- Add for each token $(p,c,h,[a,b])$ of $Jout(e_1)$, the constraint $\underline{h} = 0$:
  $h_1=0 \wedge h_2=0 \wedge h_3=0 \wedge max(0,2-h_1,1-h_2) \leq dh \leq\ min(max(2-h_1,1-h_2),max(2-h_1,2-h_3)) \wedge$
  $\underline{h_1}= h_1+dh \wedge \underline{h_2}=h_2+dh \wedge \underline{h_3}=h_3+dh \wedge \underline{h_4}=0$.
- Eliminate by substitution old variables h , dh and clocks of all tokens consumed by event e_1 (i.e.: tokens of $Jin(e_1)$); Afterwards, rename each variable $\underline{h}$ in h: $h_3=2 \wedge h_4=0$.

The reachable class by firing event e_1 from the class α_0 is then $\alpha_1 = (SM_1, FT_1)$ where:
$SM_1 = (p_{in}, J_2, h_3,[2,2]) + (p_{busy}, (M_1, J_1), h_4,[1,3])$ and
$FT_1 = (h_3 = 2 \wedge\ h_4 = 0)$.

In the state class graph, only states reachable by firing sequences are explicitly represented. States reachable by time progression are not represented but, they can be computed from those represented in the state class graph.

3.2 State class evolutions

Let $\alpha=(SM,FT)$ be a state class. The set of its evolutions is the union of evolutions of its states. We give in the following the basis operations useful to determine its evolutions (i.e.: time progression, immediate firing).

Proposition 2: (Time progression)

Let $\alpha=(SM,FT)$ be a reachable state class and dv a nonnegative real value.

- A time progression of dv units is possible from α iff the following formula is consistent: $FT \wedge (\wedge_{e \in E(SM)} dv \leq FD_{max}(e))$.
- In this case, the set of all reachable states from α by time progression of dv units is (SM,FT'), where FT' is computed as follows:
  - Initialize FT' to: $FT \wedge (\wedge_{e \in E\{SM\}} dv \leq FD_{max}(e))$;
  - Replace each clock h in FT' by $h\text{-}dv$.

Proof: A time progression of dv units is possible for all states of α such that the maximal firing delays of their events are all greater or equal to dv. The time progression condition isolates from α all states allowing a time progression of dv units. If it is consistent, the formula characterizing the set of reachable states by this time progression is obtained, from this condition, by replacing each clock h by $h\text{-}dv$. This substitution increments appropriately the values of h: $old(h) = new(h) - dv$.

Example 3: (Time progression)

Consider the state class $\alpha_1=(SM_1,FT_1)$ computed in Example 2. α_1 has only one event: $e_3=(t_2, (p_{busy}, (M_1,J_1),h_4,[1,3]))$.

A time progression of 2 units is possible from the class α_1 because the following formula is consistent:
$h_3=2 \wedge h_4=0 \wedge 2 \leq 3 - h_4$.

The set of all reachable states from states of α_1 after 2 time units is $\alpha_1'=(SM_1,FT_1')$ where:
$SM_1 = (p_{in}, J_2, h_3,[2,2]) + (p_{busy}, (M_1, J_1), h_4,[1,3])$ and
FT_1' is computed as follows:

- Initialize FT_1' with: $h_3=2 \wedge h_4=0 \wedge 2 \leq 3 - h_4$.
- Replace each clock h by $h\text{-}2$: $h_3\text{-}2=2 \wedge h_4\text{-}2=0 \wedge 2 \leq 3 - h_4+2$.

Hence, $FT_1' = (h_3 = 4 \wedge h_4 = 2)$.

Proposition 3: (Immediate firing)

Let $\alpha=(SM,FT)$ be a reachable state class and e_f an event of α.

- Event e_f can fire immediately (without any time progression) from the state class α iff the following formula is consistent: $FT \wedge (FD_{min}(e_f)=0)$.
- In this case, the set of all reachable states (SM,FT') from α by firing immediately event e_f, can be computed as follows:
  - Initialize FT' to: $FT \wedge (FD_{min}(e_f)=0)$;
  - Eliminate by substitution all clocks associated with tokens consumed by e_f and add for each token $(p,c,h,[a,b])$ of $Jout(e_f)$, the constraint $h=0$.

Proof: Event e_f can fire immediately (without any time progression) from all states of α such that their minimal firing delays are equal to 0. The firing condition given in the proposition isolates from α all states allowing e_f to fire immediately. If it is consistent, the formula characterizing the set of reachable states is obtained, from this condition, by eliminating by substitution all clocks associated with tokens consumed by e_f, and adding for each token $(p,c,h,[a,b])$ newly created, the constraint $h=0$. The initial value of clocks is 0.

Example 4: (Immediate firing)

Consider the state class α_1' computed in Example *3*. Its event $e_3= (t_2, (p_{busy}, (M_1,J_1), h_4,[1,3]))$ is immediately firable from α_1' since the following formula is consistent: $h_3=4 \wedge h_4=2 \wedge max(0,1 - h_4) = 0$.

The set of all reachable states from states of α_1' by firing immediately event e_3 is (SM_3,FT_3) where: $SM_3 = (p_{in}, J_1, h_1,[2,2]) + (p_{free}, M_1, h_2,[1,1]) + (p_{in}, J_2, h_3,[2,2])$ and FT_3 is computed as follows:

- Initialize FT_3 with: $h_3=4 \wedge h_4=2 \wedge max(0,1 - h_4) = 0$.
- Eliminate by substitution all clocks associated with tokens consumed by e_3 and add for each token $(p,c,h,[a,b])$ of $Jout(e_3)$, the constraint $h=0$: $h_3=4 \wedge h_1=0 \wedge h_2=0$.

By construction, all firing sequences as well as their reachable states are explicitly represented in the state class graph. Moreover, we can deduce from the states represented in the graph, all evolutions of the model. Consequently, the state class graph preserves all linear properties. However, the firing rule given in Proposition *1* is not simple to use since it requires resolution of systems of inequations. Furthermore, as we will show, in the next section, this graph can be infinite even if, the model is bounded.

4 Simplification of the firing rule

We show, in the following, how to simplify the firing rule, given in Proposition *1*, into an attractive form more simple to implement. This simplification leads to an attractive characterization of state classes which simplifies both the computation and the comparison of state classes.

4.1 Clock bound function of a state class

The simplified form of the firing rule, established in Propositions *4* and *5*, is based on the clock bound function of a state class. This function indicates for each clock its bounds and for each pair of clocks, the bounds of their difference.

Definition 10: (Clock bound function)

Let $\alpha=(SM,FT)$ be a state class, V_{SM} the set of all variables (clocks) in SM and o the symbol representing the value zero.

We define the clock bound function H as follows:

$H: (V_{SM} \cup \{o\})^2 \rightarrow Q \cup \{ \infty\} , H(x,y) = max (x\text{-}y \mid FT)$.

In other terms, $H(x,y)$ is the biggest value of $x\text{-}y$ in the domain of FT.

The clock bound function H_0 of the initial state class $\alpha_0=(SM_0,FT_0)$ is:

$\forall(x,y) \in (V_{SM0} \cup \{o\})^2, H_0(x,y) = 0$.

The next theorem states that a state class can be characterized by its symbolic marking and its clock bound function. This characterization is more useful for comparison of state classes. Two state classes $\alpha=(SM,FT)$ and $\alpha'=(SM',FT')$ are equal iff, it is possible to rename clocks in (SM',FT') so as to obtain: $SM = SM'$ and $H = H'$.

In this way, we avoid the equivalency test of formulas.

Theorem 1: (Canonical form)

Let $\alpha=(SM,FT)$ be a state class. The domain of FT is equivalent to: $\bigwedge_{(x,y) \in (VSM \cup \{o\})^2} (x\text{-}y \le H(x,y))$

This form is called the canonical form of FT.

Proof: (By induction)

The domain represented by FT_0 is equivalent to:

$\bigwedge_{(x,y) \in (VSM0 \cup \{o\})^2} x\text{-}y \le 0$.

Suppose that the domain of a class $\alpha=(SM,FT)$ is equivalent to:

$\wedge_{(x,y)\in(VSM\cup\{o\})^2}(x-y\leq H(x,y))$.

Let $\alpha'=(SM',FT')$ be the successor of $\alpha=(SM,FT)$ by an event e_f. From Proposition *1*, domains of its clocks ($\underline{h}=h+dh$) and the differences between its clocks ($\underline{h}-\underline{h}'$) are continuous intervals. Moreover, the bounds of these intervals belong to the domains. Therefore, the domain of FT' is equivalent to:

$\wedge_{(x,y)\in(VSM'\cup\{o\})^2}(x-y\leq H'(x,y))$.

4.2 Firing rule using clock bounds

In the previous subsection, we have shown that state classes can be characterized by their markings and their clock bound functions. This characterization simplifies the comparison of state classes. We establish in the following another firing rule more appropriate and more simple (avoiding the resolution of systems of inequations).

Let $\alpha=(SM,FT)$ be a state class and H its clock bound function.

Let j_i be a token of SM. We denote respectively h_i, a_i and b_i the clock and bounds of the static delay interval of j_i.

For any event e and any clock h of α, $mina(h,e)$ and $maxb(h,e)$ denote respectively $min_{ji\in Jin(e)} H(h_i,h)-a_i$ and $max_{ji\in Jin(e)} H(h,h_i)+b_i$.

Proposition 4: (Computing clock bounds)

Let $\alpha=(SM,FT)$ be a state class, e_f an event firable from α and $\alpha'=(SM',FT')$ the successor of α by e_f.

The function H' can be computed using H as follows:

- $\forall x\in V_{SM'}\cup\{o\}, H'(x,x)=0.$
- $\forall h\in V_{SM'}$, if its token is created by e_f: $H'(o,h)=0$ and $H'(h,o)=0$
  else $H'(o,h)=min\,(H(o,h), mina(h,e_f))$ and
  $H'(h,o)=min_{e\in E(\Sigma M)}maxb(h,e)$
- $\forall(h,h')\in V_{SM'}{}^2, h\neq h'$,
  if their tokens are not created by e_f: $H'(h,h')=min(H(h,h'), H'(h,o)+H'(o,h'))$
  else: $H'(h,h')=H'(h,o)+H'(o,h')$

Proof: Recall the definition of H': $\forall(x,y)\in(V_{SM'}\cup\{o\})^2$, $H'(x,y)=max\,(x-y\mid FT')$

1) Then: $\forall x\in V_{SM'}\cup\{o\}$, $H'(x,x)=0$;
2) Computing $H'(h,o)$ and $H'(o,h)$, for $h\in V_{SM'}$:
   - In case h is associated with some token created by event e_f, the value of h is 0. Hence, $H'(h,o)=0$ and $H'(o,h)=0$.
   - In case h is associated with some token not created by e_f, $H'(h,o)$ and $H'(o,h)$ are respectively the biggest values of $h+dh$ and $-(h+dh)$ satisfying the firing condition of e_f, i.e.: $FT\wedge FD_{min}(e_f)\leq dh\leq min_{e\in E(SM)}\ FD_{max}(e)$. After developing FD, we can rewrite this firing condition to obtain: $(FT\wedge\ max\,(0, max_{jf\in Jin(ef)}\ (a_f-h_f))\ \leq dh\leq min_{e\in E(SM)}\ max_{ji\in Jin(e)}\ (b_i-h_i))$.

   The biggest values of $h+dh$ and $-(h+dh)$ can be obtained by adding h to each member of the second term of the formula, replacing in the upper bound of $h+dh$, each $h-h_i$ by the biggest value of $h-h_i$ satisfying FT (i.e.: $H(h,h_i)$) and finally replacing in the lower bound of $h+dh$, each $h-h_f$ by the smallest value of $h-h_f$ satisfying FT (i.e.: $-H(h_f,h)$).
3) Computing $H'(h,h')$, for $(h,h')\in V_{SM'}{}^2$:
   - In case h or h' is associated with some token created by e_f, we have $H'(h,o)=0$ or $H'(o,h')=0$. Therefore: $H'(h,h')=H'(h,o)+H'(o,h')$;
   - In case h and h' are associated with tokens not created by e_f, $H'(h,h')$ is the biggest value of $(h-h')$ satisfying the firing condition of e_f.

   Consequently, $H'(h,h')=min\,(H(h,h'),H'(h,o)+H'(o,h'))$.

Proposition 5: (Firing condition)

Let e_f be an event of $\alpha=(SM, FT)$. e_f can occur from the state class α iff: $min_{jf\in Jin(ef)\ e\in E(SM)}(max_{ji\in Jin(e)}\ b+H(h_f,h_i))-a_f \geq 0$.

Proof: The firing condition given in Proposition 1, can be rewritten to obtain: $FT \wedge min_{jf\in Jin(ef),e\in E(SM)}(max_{ji\in Jin(e)}b+h_f-h_i)-a_f\geq 0$.

Since $H(h_f,h_i) = max(h_f-h_i \mid FT)$, the previous formula is consistent iff:
$min_{jf\in Jin(ef),e\in E(SM)}(max_{ji\in Jin(e)}\ b+H(h_f,h_i))-a_f \geq 0$.

Example 5: (Applying the firing rule)
Consider the initial state class $\alpha_0 = (SM_0,H_0)$ of the model in Figure *1*.

$H_0 =$

	o,h_1,h_2,h_3
o,h_1,h_2,h_3	0

Using the clock bound function, the successor by event e_1 of this class is computed as follows:

Event e_1 can occur from the state class α_0 because the following relation holds:
$min(max(2+0,1+0)-2,max(2+0,2+0)-2\ ,\ max(2+0,1+0)-1,\ max(2+0,2+0)-1) \geq 0$.

Its occurrence leads to the class $\alpha_1 = (SM_1,H_1)$ where: $SM_1 = (p_{in},\ J_2,\ h_3,[2,2]) + (p_{busy},\ (M_1,\ J_1),\ h_4,[1,3])$ and non null elements of H_1 are computed as follows:
$H_1(h_3,o)=min\ (\ max(2+0,2+0),\ max(2+0,1+0))=2$
$H_1(o,h_3)= min(0,\ 0\text{-}2,\ 0\text{-}1) = \text{-}2$
$H_1(h_3,h_4)= H_1(h_3,o)=2$
$H_1(h_4,h_3)= H_1(o,h_3)= \text{-}2$

The reachable class by firing event e_1 from the class α_0 is then $\alpha_1 = (SM_1,\ H_1)$ where:
$SM_1 = (p_{in},\ J_2,\ h_3,[2,2]) + (p_{busy},\ (M_1,\ J_1),\ h_4,[1,3])$ and

$H_1\ =$

	o,h_4	h_3
o,h_4	0	-2
h_3	2	0

4.3 State class evolutions

With the clock bound functions, we have been able to easy up the construction of the state class graph (computation and comparison of state classes). In the following, we propose a simple way to determine using the clock bound functions the state class evolutions. We focus on the computation of clock bound functions. The computation of markings is done in the same way as shown in section *3*.

Proposition 6: (Time progression)
Let $\alpha=(SM,FT)$ be a reachable state class and dv a nonnegative real value.

- A time progression of dv units is possible from the state class α iff: $dv \leq min_{e\ \in E(SM)}\ max_{ji\in Jin(e)}\ (b+H(o,h_i))$.
- In this case, the clock bound function H' of the set of states reachable from α by time progression of dv is computed as follows:
    - $\forall x\ \in V_{SM} \cup \{o\},\ H'(x,x)\ =\ 0$

- $\forall h \in V_{SM}$, $H'(o,h) = H(o,h)-dv$ and $H'(h,o) = min(dv+H(h,o), min_{e\in E(SM)} maxb(h,e))$
- $\forall (h,h') \in V_{SM}^2$, $h \neq h'$, $H'(h,h') = min(H(h,h'), H'(h,o)+H'(o,h'))$

Proof: The proof is similar to those of Propositions *4* and *5*.

- Recall the time progression condition given in Proposition 2: $FT \wedge dv \leq min_{e\in E(SM)} FD_{max}(e)$
  After developing FD_{max}, we obtain: $FT \wedge (\wedge_{e\in E(SM)} dv \leq max_{ji\in Jin(e)} b_i-h_i$.
  This formula is consistent iff: $dv \leq min_{e\in E(SM)} max_{ji\in Jin(ei)} b_i+H(o,h_i)$.
- (1) By definition of the clock bound function, we have: $\forall x \in V_{SM'} \cup \{o\}$, $H'(x,x)=0$;
  (2) For each $h \in V_{SM'}$, $H'(h,o)$ and $H'(o,h)$ are respectively the biggest values of $h+dv$ and $-(h+dv)$ satisfying the time progression condition. After developing *FD* and add *h* to each member of the second term of the formula, we obtain:
  $FT \wedge (h+dv) \leq min_{e\in E(SM)} max_{ji\in Jin(e)} (b_i - h_i+h)$ Then:
  $H'(o,h) = H(o,h)-dv$ and $H'(h,o) = min(dv+H(h,o), min_{e\in E(SM)} maxb(h,e))$
  (3) For $(h,h') \in V_{SM'}^2$, $H'(h,h')$ is the biggest value of $(h-h')$ satisfying the time progression condition. Consequently,
  $H'(h,h') = min (H(h,h'), H'(h,o)+H'(o,h'))$.

Ω

Example 6: (Time progression)

Consider the state class $\alpha_1 = (SM_1,H_1)$ computed in Example *2*. A time progression of *2* units is possible from the class $\alpha_1=(SM_1,H_1)$ because the following relation holds: $2 \leq 3-0$.

The set of all reachable states from states of α_1 after *2* time units is $\alpha_1'=(SM_1,H_1')$ where:

$H_{1'}$ =

	O	h_4	h_3
o	0	-2	-4
h_4	2	0	-2
h_3	4	2	0

Proposition 7: (Immediate firing)

Let $\alpha=(SM,FT)$ be a reachable state class and e_f an event of α.

- Event e_f can fire immediately (without any time progression) from the class α iff: $max_{jf\in Jin(ef)} a_f-H(h_f,o) \leq 0$.
- In this case, the clock bound function H' of the state class reachable from α by firing immediately e_f, can be computed as follows:
  - $\forall x \in V_{SM} \cup \{o\}$, $H'(x,x) = 0$
  - $\forall h \in V_{SM}$,
    $H'(o,h) = min(H(o,h), mina(h,e_f))$
    $H'(h,o) = H(h,o)$
  - $\forall (h,h') \in V_{SM}^2$, $h \neq h'$,
    $H'(h,h') = min(H(h,h'), H'(h,o) + H'(o,h'))$

Proof: The proof is similar to the proofs of Propositions *4* and *5*.

- Recall the immediate firing condition given in Proposition 3: $FT \wedge (FD_{min}(e_f)=0)$. After developing FD_{min}, we can rewrite it to obtain: $FT \wedge max_{jf\in Jin(ef)} a_f - h_f \leq 0$
  This formula is consistent iff: $max_{jf\in Jin(ef)} a_f-H(h_f,o) \leq 0$.
- (1) From the definition of the clock bound function, we have: $\forall x \in V_{SM'} \cup \{o\}$, $H'(x,x)=0$;

(2) For each $h \in V_{SM'}$, $H'(h,o)$ and $H'(o,h)$ are respectively the biggest values of h and $-h$ satisfying the immediate firing condition. After developing FD_{min} and adding h to each member of the second term of the formula, we can rewrite it to obtain: $FT \wedge max_{jf \in Jin(ef)}\ a_f - h_f + h \leq h$.

Then: $H'(o,h) = min\ (H(o,h),\ min_{jf \in Jin(ef)}\ (H(h_f,h) - a_f))$

and $H'(h,o) = H(h,o)$

(3) For each $(h,h') \in V_{SM'}^{\ 2}$, $H'(h,h')$ is the biggest value of $(h-h')$ satisfying the immediate firing condition. Consequently, $H'(h,h') = min\ (H(h,h'), H'(h,o) + H'(o,h'))$.

Example 7: (Immediate firing)

Consider the state class α_1' computed in Example *6*. Its event $e_3 = (t_2, (p_{busy}, (M_1,J_1), h_4,[1,3]))$ is immediately firable from α_1' since the following relation holds: $1 - 2 \leq 0$.

The set of all reachable states α_3 by firing immediately event e_3 is (SM_3,H_3) where:

$SM_3 = (p_{in},J_1, h_1,[2,2]) + (p_{free},M_1, h_2,[1,1]) + (p_{in},J_2, h_3,[2,2])$

H_3 =

	o, h_1, h_2	h_3
o,h_1,h_2	0	-4
h_3	4	0

We have proposed another characterization of state classes which is more manageable and induces both more simple computation and comparison of state classes. It also allows computing more easily the state class evolutions. However, the state class graph can be infinite even if, the model has a finite number of reachable markings (bounded). This may occur, for example, if there is some token which will never be consumed. Its clock will increase infinitely.

For example, consider the model in Figure *1*. The model has three reachable markings but an infinite number of reachable state classes. From the initial state class, the token $(p_{in}, J_2, h_3,[2,2])$ will never be consumed, if the model executes repeatedly the sequence (e_1,e_3): $e_1 = (t_1, (p_{free}, M_1, h_1,[2,2])) + (p_{in}, J_1, h_2,[1,1])$ *and* $e_3 = (t_2, (p_{busy}, (M_1,J_1), h_4,[1,3]))$.

The occurrence of this sequence leads to the state class $\alpha_4 = (SM_4, H_4)$:

SM$_4$ = $(p_{free}, M_1, h_1,[2,2]) + (p_{in}, J_1, h_2,[1,1]) + (p_{in}, J_2, h_3,[2,2])$

and

H_4 =

	o, h_1, h_2	h_3
o,h_1,h_2	0	-3
h_3	5	0

Each execution of this sequence generates a new state class. All state classes obtained share the same marking but the domains of h_3 are different. These domains are all beyond the time stamp interval *[2,2]* of the associated token. As firing delays of events do not depend on values beyond the time stamp intervals of their tokens, we can add to the domain of h_3 all values greater or equal to 2. This operation, called relaxation, produces the same state class (SM_{13}, H_{13}), when we apply it to all state classes reachable by firing repeatedly the sequence e_1e_3:

$SM_{13} = (p_{in},J_1, h_1,[2,2]) + (p_{free},M_1, h_2,[1,1]) + (p_{in}, J_2, h_3,[2,2])$ and

H_{13} =

	o, h_1, h_2	h_3
o,h_1,h_2	0	-2
h_3	∞	0

For further contractions, we propose to relax each computed state class before comparing it with previously computed state classes. As we will show, with this operation, we obtain finite graphs for all bounded ITCPNs.

5 Relaxation of state classes

The relaxation of a state class consists in adding, to the clock domains, some values which do not affect the evolutions of the state class. This operation, known under the name *k-approximation*, is similar to the one used for timed automata *[9]* and time Petri nets *[13]*.

Let $\alpha=(SM,H)$ be a state class and $(p,c,h,[a,b])$ a token of α. Let k_h be the constant defined by:
if $(b \neq \infty)$ then $k_h = b$ else $k_h = a$.

In case the domain of h contains some values greater or equal to h, we can add to the domain of h all values greater or equal to k_h, without affecting the behaviour of the class. The reason is that when h reaches the value k_h, the token becomes available and stays available until its consummation.

Let $\alpha = (SM,H)$ be a state class. The relaxation of α is done as follows:

```
        For each   h   in  V_SM
     { // let  [a,b]  be the time stamp interval of  h.
•             if (b ≠ ∞)    k_h = b     else   k_h = a;
•             if ( H(h,o) • k_h ) {  //the biggest value of h is  • k_h
                                 H(h,o)=••∞;
              •    if (-H(o,h) • k_h) {  //the smallest value of h is  • k_h
                                 H(o,h)= - k_h ;
                      For  each  h'   in  (V_SM - {h})
           {       • // let  [a',b']  be the time stamp interval of  h'.
                        if (H(h,h') • k_h ) //the biggest value of h-h' is  • k_h
                                       H(h,h') =••∞;
                                 if ( b' ≠ ∞ )    k_h' = b'    else    k_h' = a';
                                 if (-H(h,h') •   k_h' ) //the smallest value of h'-h is   •
k_h'
                                  H(h,h')= - k_h' ;
•             }
                   //compute tightest clock bounds ;
            For each x in V_SM ∪{o}
                        For each y in V_SM∪{o}
                              For each z  in V_SM∪{o}
                                    if (H(x,y) > H(x,z)+ H(z,y))
                                                H(x,y) =  H(x,z)+ H(z,y)•;
•     }
```

Example 8: (Relaxation of a state class)

Consider the state class $\alpha_1 = (SM_1,H_1)$ computed in Example *5*. This state class has to be relaxed because $-H_1(o,h_3)\geq 2$ and $H_1(h_3,o)\geq 2$. The relaxation of h_3 in α_1 consists in replacing its domain $[2,2]$ by $[2,\infty]$. After relaxation, we obtain:

H_1 =

	o,h_4	h_3
o,h_4	0	-2
h_3	∞	0

The following theorem establishes that the relaxation of a state class has the same set of evolutions as itself.

Theorem 2: Let α= (SM,H) be a state group. Then α and its relaxation have the same evolutions.

Proof: We relax a domain of some clock h of V_{SM}, iff some values of h in α are greater or equal to its k_h. In this case, the relaxation consists in replacing the domain of h by $[min(-H(o,h), k_h),\infty]$. This operation does not affect the evolutions of the state class, because the delay interval $[max(0,a-h),b-h]$ of the token $(p,c,h,[a,b])$ associated with h is the same for each value of h greater or equal to k_h. If $b = \infty$, the delay interval is $[0,\infty]$. Otherwise, the delay interval is $[0,0]$.

We relax the domain of h-h', iff some values of h-h' in α are greater or equal to k_h (i.e.: $H(h,h') \geq k_h$) or smaller or equal to $-k_{h'}$ (i.e.: $H(h,h') \leq - k_{h'}$). Indeed, if $H(h,h') \geq k_h$ then $H(h,o) \geq k_h$. In this case, relaxation extends the domain of h to infinite and then the domain of h-h' has to be extended to infinite too. If $H(h,h') \leq - k_{h'}$ then $H(o,h') \leq - k_h$. In this case, relaxation replaces $H(o,h')$ by $-k_h$. Since $h\text{-}h' \geq -h'$ and (after relaxation) $-h' \geq - k_{h'}$, we have to replace $H(h,h')$ by $-k_{h'}$.

It comes that each added state has the same evolutions as some state of the class. Then, the added states when we relax a class do not affect the evolutions of the class.

The state class graph of an ITCPN model is built by applying the firing rule given in Propositions *4* and *5* to the initial state class and to each new state class. Each computed state class is relaxed before comparing it with the previously computed state classes. As the relaxation of state classes does not affect their evolutions, the relaxed state class graph preserves all linear properties of the model. If it is finite, it can be used to determine linear properties of the model by applying the appropriate classical model checking techniques.

Theorem *3* below establishes one necessary and sufficient condition to obtain finite relaxed state class graphs. Its proof is based on the following proposition proven in *[5]*.

Proposition 9: Let Y be a finite linear combination of rational constants with integer coefficients. If Y is bounded by finite rational constants (i.e.: $inf \leq Y \leq sup$ and $inf, sup \neq \infty$) then the value domain of Y is finite.

Theorem 3: An ITCPN has a finite relaxed state class graph iff, it is bounded (it has a finite number of reachable markings).

Proof:

$\rightarrow$) is obvious.

$\leftarrow$) If the model is bounded, it has a finite set of reachable markings. Since the number of different markings is finite, it suffices to prove that for each marking, we have a finite number of different classes that share the same marking.

Consider a marking. We have to show that there is a finite number of different functions H for the considered marking. For each pair of clocks (h,h') of V_{SM}^2, terms $H(h,h')$, $H(o,h')$ and $H(h,o)$ are finite combinations of rational constants with integer coefficients (finite combinations because the number of different time intervals in the model is finite).

These terms are bounded or equal to ∞. Let $[a,b]$ be the time stamp interval associated with h and k_h the value defined by:

$k_h = b$ if $b \neq \infty$, $k_h = a$ otherwise.

- $H(h,o)$ is initialized to 0. The value of $H(h,o)$ increases with time but the relaxation changes to ∞ all its values greater or equal to k_h. Then: $0 \leq H(h,o) \leq k_h$ or $H(h,o) = \infty$.

- $H(o,h)$ is initialized to 0. Its value decreases with time but the relaxation changes to $- k_h$ all its values smaller than $- k_h$. Then: $- k_h \leq H(o,h) \leq 0$.
- $- k_{h'} \leq H(h,h') \leq k_h$ or $H(h,h') = \infty$.

From Proposition 9, the value domains of $H(h,o)$, $H(o,h')$ and $H(h,h')$ are finite. Consequently, the number of different H is finite.

This necessary and sufficient condition that ensures a finite relaxed state class graph may be difficult to use since we have not a general procedure to decide whether or not an ITCPN has a finite number of reachable markings. However, we have a straightforward sufficient condition using the underlying colored Petri net (CPN), and we know several methods to decide this property on CPN, namely the invariant method: An ITCPN has a finite number of markings (i.e.: bounded), if its underlying CPN has a finite number of reachable markings. The reverse is not true. Indeed, an ITCPN can have a finite set of reachable markings but its underlying CPN has an infinite number of reachable markings. As an example, consider the model shown in Figure *2* where the initial marking is $M_0 = (p_1,prod)$, $F(t_1)(p_1,prod) = (p_1,prod,[1,2]) + (p_2,mess,[0,1])$, and $F(t_2)(p_2,mess) = 0$.

This model has three reachable markings: M_0, $M_1 = (p_1,prod)+ (p_2,mess)$ and $M_2 = (p_1,prod) + 2\ (p_2,mess)$. But its underlying coloured Petri net is unbounded (place p_2 is unbounded).

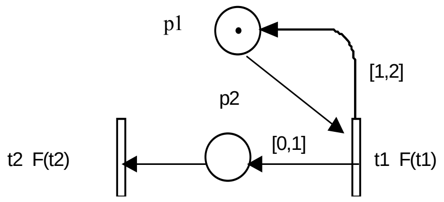

Fig. 2. Bounded ITCPN but unbounded CPN

Example 9: (Relaxed state class graph)

Applying our approach to the model in Figure *1* produces the state class graph shown in Figure *3*. It consists of *5* state classes, *8* arcs and *4* events:

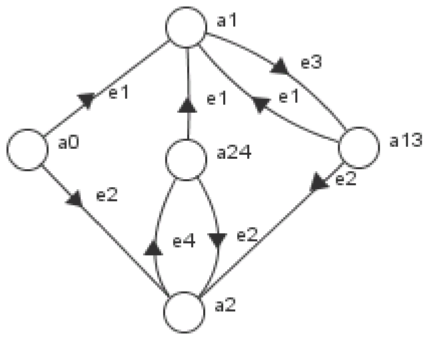

Fig. 3. The relaxed state class graph of the model in Figure *1*

α_0: $SM_0= (p_{free},M_1, h_1,[2,2]) + (p_{in},J_1, h_2,[1,1]) + (p_{in},J_2, h_3,[2,2])$,

H_0 =

	o,h_1,h_2,h_3
o,h_1,h_2,h_3	0

α_1: $SM_1 = (p_{in}, J_2, h_3,[2,2]) + (p_{busy}, (M_1,J_1), h_4,[1,3])$,

H_1 =

	o,h_4	h_3
o,h_4	0	-2
H_3	∞	0

α_{13}: $SM_{13}= (p_{free},M_1,h_1,[2,2])+(p_{in},J_1,h_2,[1,1]) + (p_{in}, J_2, h_3,[2,2])$,

H_{13} =

	o, h_1, h_2	h_3
o,h_1,h_2	0	-2
H_3	∞	0

α_2: $SM_2=(p_{in},J_1,h_2,[1,1])+(p_{busy},(M_1,J_2),h_5,[1,3])$,

H_2 =

	o, h_5	h_2
o,h_5	0	-1
H_2	∞	0

α_{24}: $SM_{24} =(p_{free},M_1,h_1,[2,2]) + (p_{in},J_1,h_2,[1,1]) + (p_{in},J_2, h_3,[2,2])$,

H_{24} =

	o, h_1, h_3	H_2
o,h_1,h_3	0	-1
h_2	∞	0

$e_1= (t_1,(p_{free}, M_1, h_1,[2,2]) + (p_{in}, J_1, h_2,[1,1]))$
$e_2= (t_1,(p_{free}, M_1, h_1,[2,2]) + (p_{in}, J_2, h_3,[2,2]))$
$e_3= (t_2, (p_{busy}, (M_1,J_1), h_4,[1,3]))$
$e_4= (t_2,(p_{busy}, (M_1,J_2), h_5,[1,3]))$.

The Figure *4* shows the graph obtained by applying our approach to the model shown in Figure *2*:

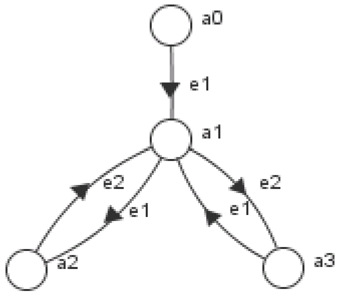

Fig. 4. The relaxed state class graph of the model in Figure *2*

α_0: SM_0= $(p_1,prod, h_1,[1,2])$

H_0 =

	o,h_1
o,h_1	0

α_1: $SM_1 = (p_1, prod, h_1,[1,2]) + (p_2, mess, h_2,[0,1])$,

H_1 =

	o,h_1,h_2
o,h_1,h_2	0

α_3: SM_3= $(p_1,Prod,h_1,[1,2])$,

H_3 =

	o	h_1
o	0	0
h_1	1	0

α_2: $SM_2=(p_1,prod,h_1,[1,2])+(p_2,mess,h_2,[0,1])+(p_2,mess,h_{2'},[0,1])$

H_2 =

	$o, h_1, h_{2'}$	h_2
$o,h_1,h_{2'}$	0	-1
H_2	1	0

$e_1= (t_1, (p_1, prod, h_1,[1,2]))$
$e_2= (t_2, (p_2, mess, h_2,[0,1]))$

Note that the state class graph obtained using the *Van der Alst*'s approach is infinite and contains unreachable markings. Indeed, starting from the initial state class $(p_1,prod,[1,2])$, event $(t_1, (p_1, prod,[1,2]))$ may occur at any date inside *[1,2]*. Its occurrence leads to the state class $\beta = (p_1,prod, [1+1,2+2]) + (p_2,mess,[0+1,1+2])$.

Both events $(t_1, (p_1, prod,[2,4]))$ and $(t_2, (p_2, mess,[1,3]))$ may occur respectively for the date intervals *[2,min(3,4)]* and *[1,min(3,4)]*.

The occurrence of event $(t_2, (p_2, mess,[1,3]))$ from β leads to the class $(p_1,prod,[1+1,2+2])$ which has the same marking as the initial state class but the interval of its token is increased. The repetitive firing of the sequence t_1 and t_2

will generate an infinite number of state classes. Hence, the graph of state classes obtained using the *Van der Aalst*'s approach is infinite.

The occurrence of event $(t_1, (p_1, prod,[2,4]))$ from β leads to $(p_1,prod,[1+2,2+3])+(p_2,mess,[1,3])+(p_2,mess,[0+2,1+3])$. From this state class, event $(t_1, (p_1, prod,[3,5]))$ may occur at date *3*. Its firing leads to the state class $(p_1,prod, [1+3,2+3]) + (p_2,mess,[1,3]) + (p_2,mess,[2,4])+ (p_2,mess,[0+3,1+3])$.

Note that the marking of this class is, in fact, not reachable, and the firing sequence t_1, t_1, t_1 cannot occur from the initial state class (see Figure *4*).

6 Analysis of the relaxed state class graph

The relaxed state class graph of an ITCPN, obtained using our approach, indicates the firing sequences and allows to compute all evolutions of the model. Consequently, it preserves linear properties of the model. So, if it is finite, it can be used to verify linear properties of the model. Untimed linear properties can be checked on the graph using the classical linear model checking techniques *[10, 14]*. Concerning timed properties, a variety of real time extensions of *Linear Time Logic* (*LTL*) have been proposed for expressing requirements of real time systems. Among these extensions, we consider the *Metric Interval Temporal Logic (MITL)* which extendes *LTL* by associating a time interval with temporal operators (always G, eventually F, and until U). The verification of these properties can be performed using a technique similar the one developed in *[1]*. This technique consists in constructing some timed automaton for the negation of the property to be The relaxed state class graph of an ITCPN, obtained using our approach, indicates the firing sequences and allows to compute all evolutions of the model. Consequently, it preserves linear properties of the model. So, if it is finite, it can be used to verify linear properties of the model. Untimed linear properties can be checked on the graph using the classical linear model checking techniques *[10, 14]*. Concerning timed properties, a variety of real time extensions of *Linear Time Logic* (*LTL*) have been proposed for expressing requirements of real time systems. Among these extensions, we consider the *Metric Interval Temporal Logic (MITL)* which extendes *LTL* by associating a time interval with temporal operators (always G, eventually F, and until U). The verification of these properties can be performed using a technique similar the one developed in *[1]*. This technique consists in constructing some timed automaton for the negation of the property to be verified and then constructing the synchronous product of the state transition system (graph of evolutions) of the model with the timed automaton. Clock constraints and guards of the automaton are used to express time requirements and atomic propositions of the formula to be checked. The property is satisfied iff the synchronous product is empty.

For example, the important bounded response requirement of real time systems is expressed by the MITL formula as follows: $G(p \Rightarrow F_{[0,c]}\ q)$ which means that request p must be followed by a response q within c time units. The timed Buchi automaton of the negation of this property is shown in Figure 5, where *init* is the initial summit and *Err* is an acceptation node. The clock *hp* is used to measure the time elapsed since *p* is satisfied.

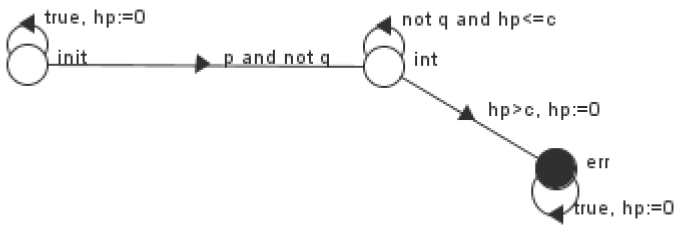

Fig. 5 Timed Buchi automaton of *not* $G(p \Rightarrow F_{[0,c]}\ q)$

As clocks of the timed Buchi automaton are used to measure time elapsing between events, their values correspond to some time paths. Therefore, for the synchronous product, we need to compute some time paths. We show, in the following, how to compute the bounds of the sojourn time in a state class as well as the time required to execute the firing sequence of a path or cycle of the graph.

6.1 Sojourn time in each state class

Let $\alpha=(SM,H)$ be a state class and e_f a firable event from α. The firing interval of event e_f in class α is the domain of the time needed for all its tokens to become available, i.e.: $[\, max(0,\ max_{jf \in Jin(ef)}\ a_f\text{-}H(h_f,o)),\ \ max_{jf \in Jin(e)}\ b_f+H(o,h_f)]$.

The lower and upper bounds of this interval are called the minimal and maximal firing delays of event e_f in class α. From α, event e_f can occur (before other events of α) after any delay between the minimal firing delay of e_f and the smallest maximal delay of all events of α. Therefore, the firing interval of event e_f before other events of α is:
$[\, max(0,\ max_{jf \in Jin(ef)}\ a_f\text{-}H(h_f,o)),\ \ min_{e \in E(SM)}\ max_{ji \in Jin(e)}\ b_i+H(o,h_i)]$.

The model may sojourn in class α until firing one of its events. The domain of the sojourn time in α is then the union of the firing intervals of all its events, i.e.: $[max(0,min_{e \in E(SM)}\ max_{ji \in Jin(e)}\ a_i\text{-}H(h_i,o)),\ \ min_{e \in E(SM)}\ max_{ji \in Jin(e)}\ b_i+H(o,h_i)]$.

Example 10: (Sojourn times)
For the state class graph given in Figure *3*, the following table reports the domain of the sojourn time in each state class graph.

Table 1 Sojourn times in each state class

α_0	*[2,2]*
α_1	*[1,3]*
α_{13}	*[2,2]*
α_2	*[1,3]*
α_{24}	*[2,2]*

We have shown how to compute using the clock bound function of a class, the domain of the sojourn time in the class. In the following subsection, we propose an algorithm which computes for a given path of the graph, the interval of its running time (path time).

6.2 Path times

Let *p* be a finite path of the relaxed state class graph of an ITCPN model. To compute its path time, we add a new variable named *hp* to all clock bound functions of the path classes. *hp* is used to measure the path time. Its value is initialized to *0* in the first class of the path. Afterwards, its domain is iteratively computed for each arc of the path as shown in Proposition *4*.

```
Bounds RunningTime ( Path p ) {
    for the first class  (SM,H)  of the path  p  do {
            add a new variable named hp to H;
                  // hp is null in the first class of the path
              H(hp,o):= 0;
              H(o,hp):= 0;
             for each  h  of  V_SM  {
                                  H(h,hp):= H(h,o);
                  H(hp,h):= H(o,h);
            }
        }

      for each arc ((SM,H),e_f,(SM',H')) of the path p (from the    first to the
last one) {
              add the variable  hp  to  H' ;
```

```
          //Compute H'(o,hp) = min(H(o,hp),  min_{jf•• Jin(ef)} (H(h_f,hp)-a_f))
             H'(o,hp):= H(o,hp) ;
            for each  (p_f,c_f,h_f,[a_f,b_f])  of  Jin(e_f) {
                 H'(o,hp):= min(H'(o,hp),H(h_f,hp)-a_f);
            }

           // Compute  H'(hp,o) =  min_{e•E(SM)} max_{j•Jin(e)}b+H(hp,h)
            H'(hp,o):= •∞;
            for each event  e  of  E(SM) {
                x:= 0;
               for each  (p,c,h,[a,b])  of  Jin(e)
                         x:= max (x, b+H(hp,h));
               H'(hp,o):= min(H(hp,o), x);
            }
            for each  h  of  V_SM' {
                                 // Compute  H'(hp,h)
                         H'(h,hp):= H'(h,o)+H'(o,hp);
                         H'(hp,h):= H'(hp,o)+H'(o,h);
                       if  h  is associated with a token not in  Jin(e_f)  {
                              H'(h,hp):= min (H(h,hp), H'(h,hp));
                              H'(hp,h):= min (H(hp,h), H'(hp,h));
                       }
            }
        }
        return   (-H'(o,hp),H'(hp,o)) ;
    }
```

Example 11: (Path time)

Consider the relaxed state class graph of the model in Figure *1* and the MITL property G$((p_{in},J_1) => F_{[0,4]}\ (p_{busy}, (M_1,J_1))$. This property means whenever the job J_1 is waiting for execution, it must be in execution within *4* time units. To verify this property, we have to construct the synchronous product of the timed buchi automaton of the property and the state class graph shown in Figure *3*. The part of the resulting graph, which exhibits that the synchronous poduct is not empty, is shown in Figure *6*.

The property is not satisfied since there exists a cycle which passes over the acceptation node (*Err*).

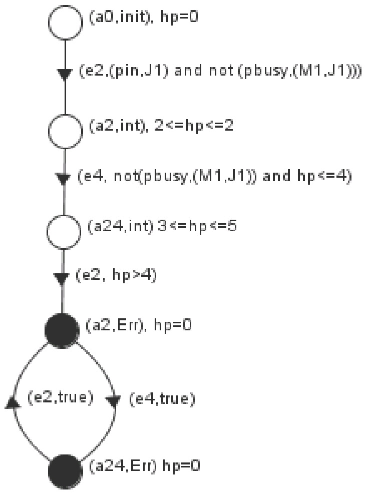

Fig. 6. A part of the synchronous product of the class graph and timed automaton shown resp. in Figures *3* and *5*.

Table *2* shows how to compute time bounds of the path *(α_0 init) e_2 (α_2 int) e_4 (α_{24} int)* which corresponds to the bounds of clock *hp* when the final node of the path is reached.

Table2 Computing bounds of some path time

(α_0 init)	$H_0(hp,o) = H_0(hp,h_1)= 0$ $H_0(hp,h_2)= H_0(hp,h_3) = 0$ $H_0(o,hp)= H_0(h_1,hp) = 0$ $H_0(h_2,hp) = H_0(h_3,hp) = 0$
(α_2,int)	$H_2(hp,o) = H_2(hp,h_5)= 2;$ $H_2(hp,h_2)= 0\ ;$ $H_2(o,hp) = H_2(h_5,hp) =-2;$ $H_2(h_2,hp)= \infty.$
(α_{24} int)	$H_{24}(hp,o)= H_{24}(hp,h_1)= 5;$ $H_{24}(hp,h_2)= 0;$ $H_{24}(hp,h_3) = 5\ ;$ $H_{24}(o,hp)= H_{24}(h_1,hp)= -3;$ $H_{24}(h_3,hp)= -3;$ $H_{24}(h_2,hp) = \infty$

7 Application

To illustrate our approach, we consider a more realistic example. In a hospital, each patient has a medical record composed of various informations (administrative, medical, surgical... etc). These informations can be stored within a single computer or distributed over a large number of interconnected systems. However, each user of the system should only access needed information for which he has the right clearance. For instance, a doctor needs to access information about his patient medical records. Whereas a secretary is only allowed to access administrative information such as the name, the age, costs of the treatments...etc. We then consider a decision-making system to decide whether or not a surgical operation is necessary according to patient state. Figure *7* shows the corresponding ITCPN model composed of seven places p_1 , p_2 , ..., and p_7, representing message channels and five transitions t_1, t_2, ..., and t_5 representing processes where:

- p_1 is a database containing all medical records of the hospital patients. These records should be accessed only by the director represented by t_1.
- p_2 is a database containing all the results of the analysis. The doctor which is represented by t_3 is the only one who is authorized to have access to it.
- p_3 is the administrative information channel of the secretary t_2.
- p_4 is the doctor's message channel.
- p_5, p_6 and p_7 are the message channels of the surgeon t_4 and t_5.

Transitions are the active entities of the system. We associate with each transition *t* a function *F(t)* which describes the resources handled during its execution and those produced.

In order to insure confidentiality and integrity of information within the system, we use the multilevel security model MLS proposed by Bell-LaPadula where security levels are assigned to the objects and subjects (users) of the system. Security requirement are characterized by two axioms:
(a) No user may read information classified above his security level ("*No read up*");

(b) No user may lower the classification of information ("*No write down*").

For our example, we consider the set of security classes *SC={U (unclassified), C (confidential), S (secret)}* with $U \leq C \leq S$. We link to each place the set of allowed security class in order to describe the clearance of the different subjects to various informations. This set is specified as its color domain. For this example, we suppose that:

$Cd(p_1)= \{U,C\}$, $Cd(p_2)= SC$, $Cd(p_3)= \{U\}$,
$Cd(p_4)= \{U,C\}$, $Cd(p_5)= \{U\}$, $Cd(p_6)=SC$, $Cd(p_7)= SC$.

So, the security classes *{U, C}* of p_1 (director's channel) means that the director is authorized to handle only data of security classes *C* and *U*. For that, each information will have a security class which describes its confidentiality.

We also need to define the information flow assertions through all transitions. For need to simplification, we assume the following security requirement such as information cannot be downgraded by a transition. This is to be applicable to all transitions except transition t_5. Each transition will be linked by a function *F* defined by:

- $F(t_1)((p_1,E))= (p_3, E, [1,2]) + (p_4, E, [0,2]);$
- $F(t_2)((p_3,E))= (p_6, E, [0,2]);$
- $F(t_3)((p_2, E) + (p_4, E')) =(p_5, \max(E,E'), [1,3]);$
- $F(t_4)((p_5, E) + (p_6, E')) =(p_7, \max(E,E'), [0,0]);$
- $F(t_5)((p_7, E)) = (p_1, \min(E,C), [0, \infty]) + (p_2, E, [1,2])$

where $E, E' \in \{U, C, S\}$.

$F(t_1)$ means that when the director decides to treat a medical record of a patient with a security class *E*, he produces two data about this patient:

- Administrative information that will be transferred to the secretary's channel. This information will have a time interval *[1,2]* and the security class of the consumed token.
- Medical information intended for the doctor's channel. Time interval and security class of this information are respectively *[0,2]* and *E*.

To simplify the explanation, we suppose that we have only one patient in each database (p_1 and p_2). The initial state class of the model is: $\alpha_0 = (SM_0, FT_0)$, where: $SM_0 =(p_1, C, h_1, [0, \infty]) + (p_2, S, h_2, [0, \infty])$ and $FT_0 = (h_1 = 0 \wedge h_2 =0)$.

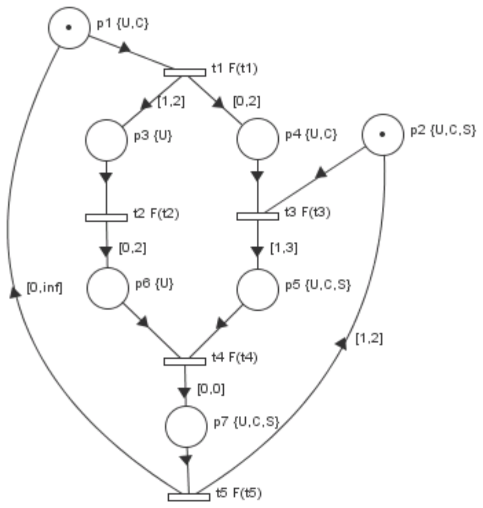

Fig.7. Sample medical process

Our interest is to verify some security properties such as integrity and confidentiality according to Bell-LaPadula's rules: "No Read up" and "No Write down". "No Read up" states that a low-level subject is not allowed to read high-level objects while "No Write down" states that no user may lower the classification of information.

Using LTL, these security requirements about states and flows can be expressed as follows:

- Integrity: *G (SecureState)*. This property means that all reachable states are secure. The proposition SecureState is evaluated to true for some state if and only if the security class of each token *(p, c, h, [a,b])* of the state conforms with the security classes of its place (i.e. $c \in Cd(p)$). It is a general way to ensure integrity because it helps to verify that only authorized subjects are allowed to operate with the data in the system.
- Confidentiality: *G (SecureFlow)*. This property means that all flows are secure. The proposition SecureFlow is evaluated to true for some state if and only if the security class of each token conforms to the security classes of its place and is not smaller than the security classes of all tokens participating to its creation.

In the state class graph, all states agglomerated within the same class share the same marking. Therefore, some state within a state class satisfies proposition SecureState iff, all states within the state class satisfy the proposition. By construction of the state class graph, if two state classes α and α'are connected by an arc then there is at least one state in α which leads by the arc to some state in α'. These two features of our state class graphs make them suitable to verify the security properties above. Hence the verification of these properties is performed on the state class graph by exploring its nodes and arcs and checking whether propositions SecureState and SecureFlow are satisfied not.

As an example, consider the state class graph (Figure *8*) of the model shown in Figure *7* and the state class $\alpha_1=(SM_1, FT_1)$ reachable by firing event $e_1=(t_1, (P_1, C, h_1, [0, \infty])$ from the initial state:

$SM_1= (p_2, S, h_2, [0,\infty]) +(p_3, C, h_3, [1,2]) + (p_4, C, h_4, [1,2]),$

$FT_1 = (0 \leq h_2 \leq \infty \wedge h_3 = 0 \wedge h_4 = 0).$

States of this class are insecure because confidentiality and probably integrity of some information is compromised. Indeed, the security class of token deposited in the place p_3 does not belong to the set of security classes admitted in this place, i.e.: $C \notin Cd(p_3)$. This problem may result in a Read-up operation because the secretary may read information classified above her clearance level. Therefore, both properties *G (SecureState)* and *G (SecureFlow)* are not satisfied. Another problem concerns the arc (α_5, t_5, α_0). This transition downgrades the security classes of information but is supposed to act as a filter process to remove parts of information that are not eligible to be received by a particular subject. It is defined in this way in order to prevent secret information to be handled by unauthorized subjects. Therefore, insecure information flows should occur only in the filter process of the system that is considered as a trusted component.

Note that to verify both properties (and some others such as reachability and invariant), we can use an abstraction by inclusion to further attenuate the state explosion problem. When a state class is explored, there is no need to explore another state class which is included in it. During the construction process, classes are computed in the same way as in section IV, but when a new class is computed, we check for inclusion instead of equality. All classes such that one of them englobes the others are grouped together. In *[9],* authors have shown that abstraction by inclusion has a good impact on performances. Both computation times and graph sizes are reduced by a factor reaching hundred in certain cases.

Another important feature of our state class representation is that it simplifies the test of inclusion. Indeed, a state class $\alpha =(SM,FT)$ is included in a state class $\alpha'=(SM',FT')$ iff, it is possible to rename clocks in *(SM',FT')* so as to obtain:

$SM = SM'$ and $\forall(x,y) \in V_{SM}^2, H(x,y) \leq H'(x,y)$.

Table *3* compares graph sizes obtained with and without abstraction by inclusion for the model shown in Figure *7*. We have considered different initial markings; parameter *n* is the number of patients in each database (i.e.: number of tokens initially in each place p_1 and p_2).

Table 3 Using abstraction by inclusion

n		Using =	Using ⊆	Ratio

2	Nodes	*193*	*54*	*3,6*
	Arcs	*430*	*128*	*3,4*
	CPU(s)	*0,01*	*0*	-
3	Nodes	*7572*	*686*	*11*
	Arcs	*25956*	*2466*	*10,5*
	CPU(s)	*0,97*	*0,10*	*9,7*
4	Nodes	*358681*	*11376*	*31,5*
	Arcs	*1681788*	*54949*	*30,6*
	CPU(s)	*130,54*	*3,7*	*35,3*
5	Nodes		*232872*	
	Arcs	-	*1414042*	
	CPU(s)		*191,1*	

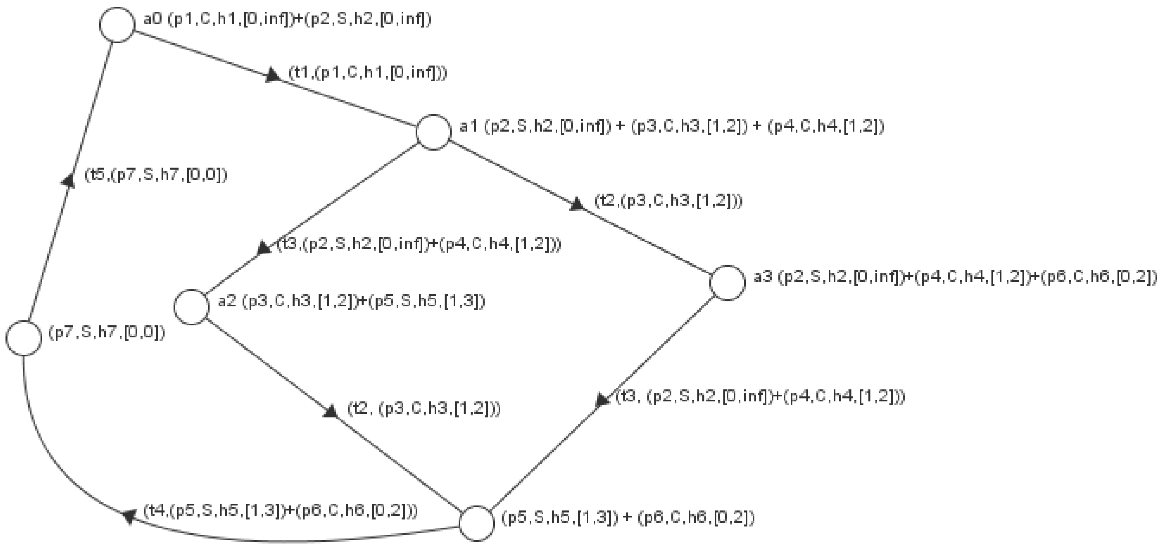
a0 (p1,C,h1,[0,inf])+(p2,S,h2,[0,inf])
(t1,(p1,C,h1,[0,inf]))
a1 (p2,S,h2,[0,inf]) + (p3,C,h3,[1,2]) + (p4,C,h4,[1,2])
(t5,(p7,S,h7,[0,0])
(t2,(p3,C,h3,[1,2]))
(t3,(p2,S,h2,[0,inf])+(p4,C,h4,[1,2]))
a3 (p2,S,h2,[0,inf])+(p4,C,h4,[1,2])+(p6,C,h6,[0,2])
a2 (p3,C,h3,[1,2])+(p5,S,h5,[1,3])
(p7,S,h7,[0,0])
(t3, (p2,S,h2,[0,inf])+(p4,C,h4,[1,2]))
(t2, (p3,C,h3,[1,2]))
(t4,(p5,S,h5,[1,3])+(p6,C,h6,[0,2]))
(p5,S,h5,[1,3]) + (p6,C,h6,[0,2])

Fig. 8. The state class graph of the ITCPN of Figure *7*

8 Conclusion

This paper has considered the Interval Timed Colored Petri Nets (ITCPNs) proposed by *Van der Aalst* in *[19]*. This model allows to describe, in a concise way, large and complex systems, but due to time density, its state space is generally infinite and then not useful for model checking techniques. To apply the linear model checking techniques to this model, we have to contract its state space into a finite graph preserving linear properties of the model. In this way, linear properties of the model can be checked on the obtained graph using the classical linear model checking techniques.

Van der Aalst proposed a contraction for the ITCPN state space which does not necessarily preserve the linear properties of the model. Moreover, this technique generates infinite graphs for models allowing infinite firing sequences. We developed here an efficient contraction which does not have these drawbacks. For bounded ITCPNs, our approach generates finite graphs which preserve linear properties. The resulting graphs are then useful to model check linear properties of the model. In addition, to deal with timed linear properties, we showed how to compute by exploring a path of the state class graph, the minimal and maximal times to execute the firing sequence of the path. Finally, we showed by means of an example how to verify using Büchi automata the timed linear properties.

Note that we have considered here only equivalences based on clocks for further agglomerations, state class graphs can be more contracted with equivalences based on colors as shown in *[6]*.

Finally, we think our approach opens interesting research paths in analysis of timed colored Petri Nets that we intend to explore further. Our immediate goal is to complete the implementation of our approach while trying to contract more state spaces without wasting linear properties of the model. Afterwards, we will interest to the contraction of state spaces, preserving CTL* properties. A similar work has already been done in *[4, 14, 18, 21]* for the Time Petri Net model (TPN).

9 References

1. R. Alur, T. Feder, T. Henzinger, *The benefits of relaxing punctuality*, Journal of ACM 43(1), 1996.
2. R. Alur, D. Dill, *Automata for modeling real-time systems*, 17ème ICALP, LNCS 443, Springer-verlag, 1990.
3. J. Bengtsson, *Clocks, DBMs and States in Timed Systems*,.PhD thesis, Dept. of Information Technology, Uppsala University, 2002.
4. B. Berthomieu, F. Vernadat, *State class constructions for branching analysis of Time Petri nests*, LNCS 2619, 2003.
5. B. Berthomieu, M. Diaz, "Modeling and verification of time dependent systems using time Petri nets", *IEEE Transactions on Software Engineering*, vol 17, n°3, March 91.
6. G. Berthelot, H. Boucheneb, *Occurrence graphs for interval timed coloured nets*, 15th International Conference on Application and Theory of Petri Nets, Zaragoza (Spain), LNCS 815, Springer-verlag, June 1994.
7. H. Boucheneb, G. Berthelot, "Contraction of the ITCPN state space", *ENTCS* vol.6, Issue 5, June 2002.
8. H. Boucheneb, G. Berthelot, *Towards a simplified building of time Petri Net Reachability graphs*, in proc. of Petri Nets and Performance Models PNPM'93, IEEE Computer Society Press, October 1993.
9. P. Bouyer, *Timed Automata May Cause Some Troubles*, Research Report LSV-02-9, 2002.
10. S.*Christensen*, L.M.Kristensen, T.Mailand, *Condensed state spaces for timed Petri Nets*, 22nd International Conference On Application and Theory Of Petri Nets, 2001.
11. C. Daws, A. Olivero, S. Tripakis and S. Yovine, *The tool Kronos*, In Hybrid Systems III, Verification and Control, LNCS 1066, Springer-verlag, 1996.
12. K. Etessami, G. Holzmann, *Optimizing Buchi automata*, 11th International Conference on Concurency Theory (CONCUR), 2000.
13. G. Gardey, O. H. Roux, O. F.Roux, *Using Zone Graph Method for Computing the State Space of a Time Petri Net*, Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2003.
14. R. Hadjidj, H. Boucheneb., Much compact time petri net state class spaces useful to restore CTL* properties, in Proc. of the Fifth International Conference on Application of Concurrency to System Design (ACSD'2005), IEEE Computer Society Press, 2005.
15. T. A. Henzinger, P-H. Ho, H. Wong-Toi, *HyTech: A Model Checker for Hybrid Systems*, Software Tools for Technology Transfer 1: 110-122, 1997.
16. *Pao-Ann Hsiung*, Chuen-Hau Gau, "Formal synthesis of real-time embedded software by time-memory scheduling of Colored Time Petri Nets", *ENTCS*, vol. 6, June 2002.
17. K. Jensen, *Coloured Petri Nets: Basic concepts, Analysis Methods and Practical use*, volumes 1 and 2, EATCS Monographs on Theoretical Computer Science, Springer-verlag, 1982.
18. W. Penczek, A. Polrola, *Abstraction and partial order reductions for checking branching properties of time Petri nets*, In Proc. Of ICATPN, LNCS 2075, pages 323-342, 2001.
19. W.M.P.*Van der Aalst*, *Interval Timed Coloured Petri Nets and their Analysis*, 14th International Conference of Application and Theory of Petri Nets, Chicago, 1993.
20. E.Vicaro, "Static Analysis and Dynamic Steering of Time Dependent Systems", *IEEE Transactions on Software Engineering*, Vol.2, No.8, 2001.
21. T.Yoneda, H. Ryuba, "CTL Model Checking of Time Petri Nets Using Geometric Regions", *IEICE Trans. Inf. & Syst.*, Vol.E99-D, No.3, 1998.

***Hanifa Boucheneb** is a Professor at the Department of Computer Engineering of École Polytechnique of Montréal (Canada). Her research areas deal with formal verification of timed and complex systems. She is interested in developing and applying model checking techniques to real time and security systems.*