Real-Time Synchronised Petri Nets

Giovanna Di Marzo Serugendo[1], Dino Mandrioli[2],
Didier Buchs[3], and Nicolas Guelfi[4]

[1] Computer Science Department, University of Geneva,
CH-1211 Geneva 4, Switzerland,
[2] Dipartimento di Elettronica e Informazione,
Politecnico di Milano, Milano 20133, Italy
[3] LGL-DI, Swiss Federal Institute of Technology,
CH-1015 Lausanne, Switzerland,
[4] Department of Applied Computer Science
IST - Luxembourg University of Applied Science,
L-1359 Luxembourg-Kirchberg

Abstract. This paper presents the combination of two well established principles: the CO-OPN synchronisation mechanism, and the Merlin and Farber time Petri nets. Real-time synchronised Petri nets systems are then defined such that a Petri net is an object that can ask to be synchronised with another net, and whose transition firing is constrained by relative time intervals. Our proposal enables to define complex systems with compact specifications, whose semantics is given through a small set of Structured Operational Semantics (SOS) rules. The applicability of the new model is shown by applying it to a traditional benchmark adopted in the literature of real-time systems.
Keywords: CO-OPN, Petri nets, real-time, inhibitor arcs.

1 Introduction

This paper is a first step to enhance a high-level class of Petri nets with real-time constraints. Starting from a simplified version of the CO-OPN [4] language, where Petri nets are able to request synchronisation with each other, we have augmented the syntax and semantics with time intervals attached to the transitions in a way similar to that of Merlin and Farber nets [11].

A real-time synchronised Petri nets specification is *object-based*, i.e., it is made of a fixed number of objects that exist since the beginning of the system. An object is a real-time Petri net with inhibitor arcs. Such a net has two kinds of transitions: external transitions, called *methods*, and internal ones simply called *transitions*. Methods and transitions may request a *synchronisation* with methods, provided no cycles are formed. A method `m` that requests synchronisation with `m'` can fire only if `m'` can fire simultaneously. However, neither methods nor transitions can request a synchronisation with a transition.

Inhibitor arcs [10] provide a symmetric enabling of methods and transitions wrt usual pre-conditions: a transition cannot fire if a place connected to it through an inhibitor arc contains tokens in a number exceeding the label of

the arc. In addition, inhibitor arcs can be used as a priority mechanism among methods and transitions. Such a mechanism is quite useful to achieve time predictability in real-time systems. Time stamped *black tokens* are used for populating places.

A *time interval* is attached to any method and any transition. The firing of a method or a transition is considered to be instantaneous, it takes place within a time interval that is relative to the time when the method or transition becomes enabled (wrt the pre-set).

Figure 1 shows a simple real-time synchronised Petri net system made of two objects (two Petri nets) called **O1** and **O2**. Object **O1** contains a place `p1`, and a method `move`. The current marking of this place contains three black tokens stamped at time 0. Object **O2** contains a place `p2`, a method `put`, and a transition `processing`. Place `p2` is initially empty. There is an inhibitor arc, with label `0`, between place `p2` and method `put`. The dashed arrow from method `move` to method `put` states that method `move` requests a *synchronisation* with method `put` whenever it fires. In this particular case, the synchronisation corresponds to the classical fusion of transitions, even if its asymmetry makes it more general. Method `move` has the time interval `[5..15]` attached, method `put` has the time interval `[2..10]` attached, and transition `processing` the time interval `[1..9]`.

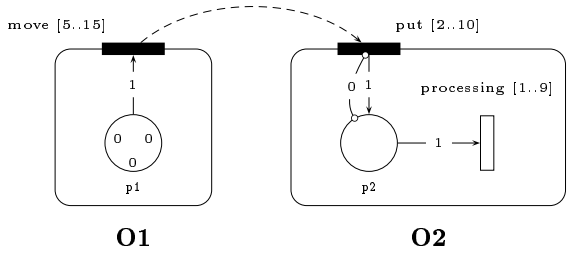

Fig. 1. A Real-time Synchronised Petri net

The intuitive semantics of this real-time synchronised Petri net system is the following: (1) method `move` requires to be synchronised with `put`, therefore `move` is enabled if its pre-condition is satisfied, and if `put` is enabled; (2) whenever `move` fires, `put` fires simultaneously, i.e., whenever `move` fires, a token is added in place `p2`. It is worth noting that `put` can fire alone without the firing of `move`; (3) the time interval attached to `move` means that `move` must fire instantaneously in the interval given by: `5` time slots after it becomes enabled, and `15` time slots after it becomes enabled. In addition, it *must* fire at the latest `15` time slots after it becomes enabled; (4) similarly, method `put` and transition `processing` must fire in the relative time interval `[2..10]`, and `[1..9]` respectively; (5) `move` requires to be synchronised with `put`, and their respective time intervals must be respected, this means that `move` and `put` must fire simultaneously in an interval corresponding to the intersection of the time interval of `move`, and that of `put`, i.e., `[5..10]` (since both are enabled at time 0); (6) the inhibitor arc associated

to `put` means that this method can fire only if the number of tokens in place `p2` is less or equal 0. Therefore, transition `processing` has a higher priority wrt method `put`. Indeed, method `put` cannot fire, if transition `processing` has not fired before, and thus emptied place `p2`. A second firing of method `put` can occur as soon as transition `processing` fires, but not before.

The main contribution of this paper consists in the definition of a compact syntax and semantics of these real-time synchronised Petri nets. The paper is structured in the following manner: section 2 defines the syntax of real-time synchronised Petri nets, section 3 describes their semantics, and section 4 gives an applicative example. A full version of this paper can be found in [5]. It includes a more detailed analysis of some intricacies in the semantics of the model.

2 Syntax

A *Petri net with inhibitor arcs* is a Petri net with two kinds of transitions: external ones, called methods, and internal ones, simply called transitions. Three kinds of arcs between places and methods/transitions are defined: input, output, and inhibitor arcs. Input, output arcs are traditional pre- post-conditions of nets. Inhibitor arcs prevent the firing of a single method or transition if the number of tokens in the place is greater than the number of tokens associated with the arc.

Definition 1. *Petri Net with Inhibitor Arcs.*
A Petri Net with inhibitor arcs is given by a 6-tuple $(P, M, T, Pre, Post, In)$ *where:* P *is a finite set of places;* M *is a finite set of methods;* T *is a finite set of (internal) transitions;* $Pre, Post : M \cup T \rightarrow (P \rightarrow \mathbb{N}\backslash\{0\})$ *are total functions, they define traditional Petri nets arcs removing or inserting black tokens respectively; to every method or transition is associated a partial function that maps places to a positive natural number.* $In : M \cup T \rightarrow (P \rightarrow \mathbb{N})$ *is a total function defining inhibitor arcs; to every method or transition is associated a partial function that maps places to a natural number.*

Remark 1. If $Pre(m)(p)$ is undefined there is no pre-set condition between place p and method m. It is similar for $Post(m)(p)$ and $In(m)(p)$. If $Pre(m)(p)$ is defined, then $Pre(m)(p) = 0$ is not allowed (idem for $Post(m)(p)$). If $In(m)(p)$ is defined, then $In(m)(p) = 0$ means there is an inhibitor arc with weight 0 between place p and method m.

A *real-time Petri net* is a Petri net with inhibitor arcs having a time interval associated to every method and transition.

Definition 2. *Real-Time Petri Net.*
A Real-Time Petri Net is given by a pair $(O, Time)$ *where* $O = (P, M, T, Pre, Post, In)$ *is a Petri net with inhibitor arcs, and* $Time$ *is a total function that associates a time interval to every method and transition of* O. $Time : M \cup T \rightarrow \mathbb{R}^+ \times (\mathbb{R}^+ \cup \infty)$, *is such that the following condition must hold:*

$$Time(m) = (t_1, t_2) \quad \Rightarrow \quad ((t_2 \geq t_1) \ \vee \ (Time(m) = (t, \infty))) \ .$$

We denote t_1 by $Time_{Inf}(m)$, and t_2 by $Time_{Sup}(m)$.

A real-time synchronised Petri nets system is a set of real-time Petri nets with a synchronisation mapping among them. A method or transition may request to be synchronised with two or more methods simultaneously (//), in sequence (..), or in alternative ($\oplus$).

Definition 3. *Real-Time Synchronised Petri nets System.*
A Real-Time Synchronised Petri nets System is given by $Sys = (O_1, \ldots, O_n, Sync)$:

- *$O_i = ((P_i, M_i, T_i, Pre_i, Post_i, In_i), Time_i)$, $1 \leq i \leq n$, a real-time Petri net;*
- *a total function $Sync : \cup_{i \in \{1,\ldots,n\}}(M_i \cup T_i) \rightarrow Sync_{Expr}$ that defines for each method and transition $m \in \cup_{i \in \{1,\ldots,n\}}(M_i \cup T_i)$ a synchronisation expression.*

The following conditions must hold:

- *$\forall i, j \in \{1, \ldots, n\}$ then $P_i \cap M_j = P_i \cap T_j = M_i \cap T_j = \emptyset$;*
- *$\forall i, j \in \{1, \ldots, n\}, i \neq j$ then $P_i \cap P_j = \emptyset$, $M_i \cap M_j = \emptyset$, and $T_i \cap T_j = \emptyset$;*
- *the set $Sync_{Expr}$ of synchronisation expressions is the least set such that:*

$$\begin{aligned} \epsilon &\in Sync_{Expr} \\ \forall M_i, i \in \{1, \ldots, n\}, M_i &\subset Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \ //\ e_2 \in Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \ ..\ e_2 \in Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \oplus e_2 \in Sync_{Expr}\ , \end{aligned}$$

  where ϵ stands for the empty synchronisation. We write "m with e" to denote $Sync(m) = e$;
- *the Sync function must ensure that a method does not synchronise with itself, and that the chain of synchronisations does not form cycles.*[1]

We denote by P the union of all sets of places P_i of a real-time synchronised Petri nets system. Similarly, we denote M, and T, the union of all methods, and all transitions respectively. We denote by Pre, $Post$, In, and $Time$ the extension of the pre-conditions, post-conditions, inhibitor arcs and time intervals to the Petri nets system.

Every token of the net is stamped with its arrival time. Several tokens may arrive in a place at the same time, as a result of the post-condition. The marking of a real-time synchronised Petri nets system is then a mapping that associates to every place a multiset of non-negative real numbers.

Definition 4. *Marking, Set of Markings.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a real-time synchronised Petri nets system. A marking is a total mapping:

$$mark : P \rightarrow [\mathbb{R}^+]\ .$$

We denote by $Mark$ the set of all markings of Sys.

[1] For simplification purposes, we impose this limitation in order to prevent infinite behaviour.

A multiset of $\mathbb{R}^+$ is given by a function $f \in [\mathbb{R}^+]$ such that $f : \mathbb{R}^+ \to \mathbb{N}$ evaluates to zero, except on a finite number of cases (thus the number of tokens in a place is finite). Here, $f(t) = j$ means that j tokens arrived at time t. We denote by $\emptyset$ the empty multiset ($\emptyset(t) = 0$, $\forall t$), and $\{t_1, t_1, t_2, t_2, t_2\}$ a multiset containing two tokens arrived at time t_1, and three tokens arrived at time t_2. It is worth noting that $mark(p)(t) = j$ means that place p contains (among others) j tokens stamped with time t.

The sum of two markings returns, for every place, a new multiset made of the union of the two original multisets, where multiple occurrences of the same time stamp are taken into account.

Definition 5. *Sum of Markings.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a real-time synchronised Petri nets system, and $Mark$ be the set of all markings of Sys. The sum of two markings is given by a mapping $+_{Mark} : Mark \times Mark \to Mark$ such that:

$$(mark_1 +_{Mark} mark_2)(p) = mark_1(p) +_{[\mathbb{R}^+]} mark_2(p) \ .$$

For every $t \in \mathbb{R}^+$, $(mark_1(p) +_{[\mathbb{R}^+]} mark_2(p))(t) = mark_1(p)(t) + mark_2(p)(t)$.
In the rest of this paper we simply note $+$ instead of $+_{Mark}$, and $+_{[\mathbb{R}^+]}$.

To every marking corresponds an *unstamped* marking, which returns for every place p the number of tokens present in the place regardless of their arrival time.

Definition 6. *Unstamped Markings.*
Let $Mark$ be the set of all markings. The Unstamped Markings are given by the the total mapping:

$$U : Mark \to (P \to \mathbb{N}) \ ,$$

where $U(mark)$ is a total mapping, s.t.

$$U(mark)(p) = \begin{cases} \sum_{t \in K_p} mark(p)(t) \\ 0, \text{if } mark(p)(t) = 0, \forall t \in \mathbb{R}^+ \end{cases}$$

where $K_p = \{t \in \mathbb{R}^+ \mid mark(p)(t) > 0\}$.

$U(mark)$ returns for every place p the number of tokens present in the place. The sum is finite since the multiset $mark(p)$ has only a finite number of elements (K_p is the finite carrier set of $mark(p)$).

Definition 7. *Initial Marking.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a real-time synchronised Petri nets system. An initial marking is a marking, $mark_k : P \to [\mathbb{R}^+]$, such that for every $p \in P$:

$$\begin{aligned} mark_k(p)(0) &\geq 0 \\ mark_k(p)(t) &= 0, \forall t > 0 \ . \end{aligned}$$

An initial marking is such that a place p contains tokens stamped at time 0. The places do not contain tokens stamped with a time greater than time 0.

Definition 8. *Marked Real-Time Synchronised Petri Nets System.*
A marked Synchronised Petri nets system is a pair $(Sys, mark_k)$ where Sys is a real-time synchronised Petri nets system, and $mark_k$ is an initial marking for Sys.

Figure 1 is the graphical notation of the marked real-time synchronised Petri nets system $(Sys, mark_k)$ given by:

$$
\begin{aligned}
Sys &= (O_1, O_2, Sync) \\
O_1 &= (\{p1\}, \{\texttt{move}\}, \emptyset, Pre_1, Post_1, In_1, Time_1), \\
O_2 &= (\{p2\}, \{\texttt{put}\}, \{\texttt{processing}\}, Pre_2, Post_2, In_2, Time_2) \\
Pre_1(\texttt{move})(\texttt{p1}) &= 1, Pre_2(\texttt{processing})(\texttt{p2}) = 1 \\
Post_2(\texttt{put})(\texttt{p2}) &= 1, In_2(\texttt{put})(\texttt{p2}) = 0 \\
Time_1(\texttt{move}) &= (5, 15), Time_2(\texttt{put}) = (2, 10), Time_2(\texttt{processing}) = (1, 9) \\
Sync(\texttt{move}) &= \texttt{put}, Sync(\texttt{put}) = \epsilon, Sync(\texttt{processing}) = \epsilon \\
mark_k(\texttt{p1}) &= \{0, 0, 0\}, mark_k(\texttt{p2}) = \emptyset\ .
\end{aligned}
$$

3 Semantics

Real-time synchronised Petri nets are a timed extension of a simplified version of CO-OPN/2 nets [2]. The establishment of their semantics follows that of CO-OPN/2: it is based on the use of *Structured Operational Semantics* (SOS) rules, similar to those of CO-OPN/2, but adapted to the real-time constraints.

We first build, using an initial set of rules, a *weak transition system* that contains transitions belonging to the weak time semantics (an enabled transition may not fire even if the time of occurrence elapses). Second, on the weak transition system, we apply a condition that enables to retain only those transitions that belong to the strong time semantics (an enabled transition *must* fire when the time of occurrence elapses) [6]. We obtain what we call the *strong transition system*. Third, on the strong transition system, we apply another set of rules (taking into account synchronisations) that enables us to obtain an *expanded transition system*. Then, we retain only those transitions necessary for the (observable) *strong time semantics*. Finally, in some cases, it is more valuable to consider a subset of the *strong time semantics* representing what we are actually interested to observe. We call this subset, the *system view semantics*.

Let us first give some preliminary definitions. An observable event is one of the following: the firing of a method, the firing of a transition, or the parallel ($//$) or sequence ($..$) firing of two observable events, or the alternative ($\oplus$) between two observable events.

Definition 9. *Observable Events.*
Let Sys be a real-time synchronised Petri nets system. The set of observable

events of Sys, denoted by Obs_{Sys}, is the least set such that:[2]

$$M \cup T \subset Obs_{Sys}$$
$$e_1, e_2 \in Obs_{Sys} \Rightarrow e_1 \ // \ e_2 \in Obs_{Sys}$$
$$e_1, e_2 \in Obs_{Sys} \Rightarrow e_1 \ .. \ e_2 \in Obs_{Sys}$$
$$e_1, e_2 \in Obs_{Sys} \Rightarrow e_1 \oplus e_2 \in Obs_{Sys} \ .$$

An event is any observable event, but also an event of the form "m with e", where the synchronisation is explicitly required.

Definition 10. *Events.*
Let Sys be a real-time synchronised Petri nets system. The set of events of Sys, denoted by Event, is the least set such that:

$$e \in Obs_{Sys} \Rightarrow e \in Event$$
$$e = m \text{ with } e', \ m \in M \cup T, \text{ and } e' \in Sync_{Expr} \Rightarrow e \in Event \ .$$

Transition systems for real-time synchronised Petri nets are made of 4-tuples, made of two markings, an event (not necessarily an observable one), and a time of occurrence.

Definition 11. *Transition System.*
Let Sys be a real-time synchronised Petri nets system. A transition system, trs, for Sys is such that:

$$trs \subseteq Mark \times Event \times Mark \times \mathbb{R}^+ \ .$$

We represent a 4-tuple $(mark_1, e, mark_2, t)$ by $mark_1 \xrightarrow[t]{e} mark_2$.

3.1 Weak Transition System

The rules for constructing the weak transition system of a real-time synchronised Petri net are given by rules BasicBeh and BasicSyncBeh formally described in Definition 12 below.

BasicBeh covers the case of the firing at time t of a single transition or method m that does not require any synchronisation. From a given marking $mark_1$, the rule enables to compute the new marking after the firing of transition m alone. The 4-tuple is produced if several conditions are met: (1) the inhibitor arc condition is satisfied ($In(m) \geq U(mark_1)$); (2) the time of occurrence t is in the absolute time interval. The absolute time interval is computed from the relative one, given by $Time(m) = (Time_{Inf}(m), Time_{Sup}(m))$, and by the greatest time of arrival of the tokens that will be removed ($\max(mark)$);[3] (3)

[2] an observable event has the same structure as a synchronisation expression. However a transition may appear in an observable event, while only methods are part of a synchronisation expression.

[3] Marking *mark* is a way of representing the pre-condition in a stamped form; it stands for the marking that will be removed from the places when the transition fires

the number of tokens that will be removed must match the pre-condition, i.e. $U(mark) = Pre(m)$; (4) the pre-condition is satisfied ($mark_1 \geq mark$); (5) the new marking, after the firing of m is obtained by removing $mark$ from marking $mark_1$, and inserting new tokens stamped at time t ($Post(m)_t$).

BasicSyncBeh covers the case of the firing of a single transition or method that requires to be synchronised with some other methods, given by synchronisation expression e. The resulting 4-tuple is the same as rule BasicBeh, except the event part, which is of the form "m with e". The new marking is obtained from $mark_1$ by considering the firing of m alone (without e). Such tuples will be exploited in Section 3.3 to define the semantics of synchronisation. Indeed, the "with e" part of the event serves as a hook for combining transitions.

Definition 12. *Rules, Weak Transition System.*
Let Sys be a real-time synchronised Petri nets system. The weak transition system, denoted by trs_{weak}, is the set obtained by the application of the inference rules BasicBeh and BasicSyncBeh to Mark. In these rules: $m \in M \cup T$, $e \in Sync_{Expr}$, and $mark_1, mark \in Mark$ are markings.

$$\text{BasicBeh}\ \frac{\begin{array}{l} Sync(m) = \epsilon \\ In(m) \geq U(mark_1) \\ Time_{Inf}(m) + \max(mark) \leq t \leq Time_{Sup}(m) + \max(mark) \\ U(mark) = Pre(m) \\ mark_1 \geq mark \end{array}}{mark_1 \xrightarrow[t]{m} mark_1 - mark + Post(m)_t}$$

$$\text{BasicSyncBeh}\ \frac{\begin{array}{l} Sync(m) = e \\ In(m) \geq U(mark_1) \\ Time_{Inf}(m) + \max(mark) \leq t \leq Time_{Sup}(m) + \max(mark) \\ U(mark) = Pre(m) \\ mark_1 \geq mark \end{array}}{mark_1 \xrightarrow[t]{m \text{ with } e} mark_1 - mark + Post(m)_t}\ .$$

In the above rules, the following conventions are used:

- $In(m) \geq U(mark_1)$ holds if $In(m)(p) \geq U(mark_1)(p)$ for every p where $In(m)$ is defined;
- $U(mark) = Pre(m)$ holds if $U(mark)(p) = Pre(m)(p)$ for every p where $Pre(m)$ is defined, and $U(mark)(p) = 0$, otherwise;
- $\max(mark) = \max(\{0\}, \cup_{p \in P} K_p)$.
  It is the greatest time of arrival of the tokens that will be removed (i.e., tokens in $mark$). Indeed, $K_p = \{t \in \mathbb{R}^+ \mid mark(p)(t) > 0\}$ gives the time stamps of tokens in $mark$. Whenever $Time_{Sup}(m) = \infty$, then $Time_{Sup}(m) + \max(mark) = \infty$;

– $Post(m)_t : P \rightarrow [\mathbb{R}^+]$ is a marking such that all tokens are stamped at time t:

$$Post(m)_t(p)(t) = Post(m)(p)$$
$$Post(m)_t(p)(t') = 0, \forall t' \neq t \ ;$$

– $mark_1 \geq mark$ holds if $mark_1(p)(t) \geq mark(p)(t)$, for every $p \in P$, and $t \in \mathbb{R}^+$.

Rules of Definition 12 provide the weak time semantics since nothing forces enabled methods (or transitions) to fire within the given time interval.

Remark 2. The weak transition system is simply a set of 4-tuples. Trying, at this point, to make sequences of transitions based on markings may lead to paths where time goes backward. Therefore, we do not consider building paths at this stage.

3.2 Strong Transition System

The strong transition system is obtained as a subset of the weak transition system, by applying a condition that removes from the weak transition system, every transition, of the form $mark_1 \xrightarrow[t]{e} mark_2$, for which there exists another transition $mark_1 \xrightarrow[t']{e'} mark_2'$ that should have fired before ($t' < t$).

Definition 13. *Strong Transition System.*
Let Sys be a real-time synchronised Petri nets system, and trs_{weak} be its weak transition system. The strong transition system, denoted by trs_{strong}, is the maximal subset of trs_{weak} such that:

$$\forall (mark_1 \xrightarrow[t]{e} mark_2) \in trs_{strong} \ \Rightarrow \ Cond(mark_1, mark_2, t) \ holds \ .$$

Condition $Cond(mark_1, mark_2, t)$ is such that:

$$\begin{aligned} Cond(mark_1, mark_2, t) := \nexists \, (m', mark', t') \in (M \cup T) \times Mark \times \mathbb{R}^+ \ s.t. \\ & t' < t \ \wedge \\ & t' = (Time_{Sup}(m') + \max(mark')) \ \wedge \\ & ((mark_1 \xrightarrow[t']{m'} mark_1 - mark' + Post(m')_{t'}) \in trs_{weak} \ \vee \\ & (mark_1 \xrightarrow[t']{m' \text{ with } e'} mark_1 - mark' + Post(m')_{t'}) \in trs_{weak}) \ . \end{aligned}$$

Remark 3. Condition $Cond$ guarantees the strong time semantics: a transition m' *must* fire at time t' if it is enabled at t', and if time t' is the maximal bound of firing of the transition ($t' = Time_{Sup}(m') + \max(mark')$). However, when two transitions or more reach their maximal bound of firing at the same time, $Cond$ does not apply, and one of the transitions may still disable the other.

3.3 Expanded Transition System

The expanded transition system is obtained from the strong transition system by adding tuples regarding synchronisation, simultaneity, alternative and sequence.

Sync handles the case of the synchronisation. From two transitions, one with a requested synchronisation, and one with the corresponding synchronisation that occur *at the same time* t, the rule produces a transition, occurring at t, where the synchronisation expression is abstracted. The new marking takes into account the effects of the simultaneous firing of m and e, but in the produced transition, the observable event m replaces "m with e". This rule produces transitions where only the firing of m is observable, but the result of the firing of m takes into account the behaviour of e. *Sim* handles the case of simultaneity of observable events. From two transitions: one for e_1 and one for e_2 that occur *at the same time* t, it builds the transition for event $e_1 \,//\, e_2$. *Alt.1* corresponds to the alternative case where e_1 fires. *Alt.2* corresponds to the case where e_2 fires. *Seq* defines transitions for sequential events: from two transitions whose final and initial marking correspond, and whose time of occurrence of the second is greater or equal to the time of occurrence of the first one, the rule produces the transition corresponding to their sequence. Definition 14 formally describe these rules.

Definition 14. *Rules, Expanded Transition System.*
Let Sys be a real-time synchronised Petri nets system, and trs_{strong} its strong transition system. The expanded transition system, denoted by trs_{expand}, is the least set obtained by the successive application, to trs_{strong}, of the inference rules Sync, Sim, Alt.1, Alt.2, and Seq below. In these rules: $m \in M \cup T$, $e \in Sync_{Expr}$, $e_1, e_2 \in Obs_{Sys}$, and $mark_1, mark_2, mark'_1, mark'_2, mark \in Mark$ are markings.

$$\text{Sync}\ \frac{\begin{array}{c} In(m) \geq U(mark_1) + U(mark'_1) + Post^*(e) \\ In^*(e) \geq U(mark_1) + U(mark'_1) + Post(m) \\ mark_1 \xrightarrow[t]{m \text{ with } e} mark_2 \quad mark'_1 \xrightarrow[t]{e} mark'_2 \end{array}}{mark_1 + mark'_1 \xrightarrow[t]{m} mark_2 + mark'_2}$$

$$\text{Sim}\ \frac{\begin{array}{c} In^*(e_1) \geq U(mark_1) + U(mark'_1) + Post^*(e_2) \\ In^*(e_2) \geq U(mark_1) + U(mark'_1) + Post^*(e_1) \\ mark_1 \xrightarrow[t]{e_1} mark_2 \quad mark'_1 \xrightarrow[t]{e_2} mark'_2 \end{array}}{mark_1 + mark'_1 \xrightarrow[t]{e_1 \,//\, e_2} mark_2 + mark'_2}$$

$$\text{Alt.1}\ \frac{mark_1 \xrightarrow[t]{e_1} mark_2}{mark_1 \xrightarrow[t]{e_1 \oplus e_2} mark_2} \qquad \text{Alt.2}\ \frac{mark'_1 \xrightarrow[t]{e_2} mark'_2}{mark'_1 \xrightarrow[t]{e_1 \oplus e_2} mark'_2}$$

$$\text{Seq}\ \frac{\begin{array}{c} t' \geq t \\ mark_1 \xrightarrow[t]{e_1} mark'_1 \quad mark'_1 \xrightarrow[t']{e_2} mark_2 \end{array}}{mark_1 \xrightarrow[t]{e_1 \,..\, e_2} mark_2}\ .$$

In the above rules, $In^*(e)$ stands for the minimal value associated to an inhibitor arc of a method or transition that takes part in the behaviour of e. $Post^*(e)$ stands for the sum of the post-conditions of all methods and transition taking part in e. Therefore, the conditions on the inhibitor arcs implies that the strongest condition applies. Formal definitions of $In^*(e)$ and $Post^*(e)$ are given in [5].

Remark 4. According to rules *Sync*, and *Sim*, an event of the form $e \,//\, (e_1 \,..\, e_2)$, or e with$(e_1 \,..\, e_2)$ occurs only if both e and $(e_1 \,..\, e_2)$ occur at the same time. Intuitively, such events should occur if e and e_1 occur simultaneously, and e_2 occurs later. Therefore, in rule *Seq*, the time of occurrence of a transition whose event is of the form $e_1 \,..\, e_2$ is the time of occurrence of e_1.

As a consequence of this choice, rule *Seq* builds 4-tuples where the resulting marking may contain tokens stamped at a time which is over the time of firing of the whole transition. These tokens result from the firing of e_2 which actually occurs later. Such tokens are actually not available; they will take part in transition firings (pre- post-conditions) only when the time will have advanced. However, they are taken into account for inhibitor arc evaluation even though they are not actually available. If such situations are not desired, the use of inhibitor arcs combined with the sequential operator should be avoided.

3.4 Strong Time Semantics

Similarly to the weak transition system, the expanded transition system contains only 4-tuples, i.e., no paths are considered. The strong time semantics builds meaningful paths from the 4-tuples available in the expanded transition system. Therefore, the semantics of a real-time synchronised Petri nets system is obtained by retaining, from trs_{expand}, all the sequences of transitions containing: (1) observable events only (no m with e); (2) markings reachable from the initial marking on a path where time is monotonic. A path p is a sequence of 4-tuples. We denote $tail(p)$ the last 4-tuple of the path.

Definition 15. *Strong Time Semantics.*
Let $(Sys, mark_k)$ be a marked real-time synchronised Petri nets system, trs_{expand} be the expanded transition system obtained with the rules of Definition 14. The strong time semantics of $(Sys, mark_k)$, denoted by Sem, is the least set of paths such that:

$$(mark_k \xrightarrow[t]{e} mark') \in trs_{expand} \;\wedge\; e \in Obs_{Sys} \Rightarrow (mark_k \xrightarrow[t]{e} mark') \in Sem$$

$$(mark'_1 \xrightarrow[t]{e} mark_2) \in trs_{expand} \wedge e \in Obs_{Sys} \wedge \exists\, p \in Sem \;s.t.\; tail(p) = (mark_1 \xrightarrow[t']{e'} mark'_1) \;\wedge\; t' \leq t \Rightarrow p\,(mark'_1 \xrightarrow[t]{e} mark_2) \in Sem\,.$$

Remark 5. If we want to obtain the weak time semantics, instead of the strong time semantics, we proceed from the weak transition system given by Definition 12, then we apply the rules of Definition 14, without applying the condition *Cond*, and finally we apply Definition 15.

3.5 System View Semantics

In some cases, the whole observable strong time semantics is too vast, and we want to keep only a subset of behaviours in order to analyse them. We model a lot of behaviours, but we want to observe actually only few of them.

For this reason, in addition to the strong time semantics, we define the *System View Semantics* representing only those paths that we want to observe. We will see in the example how it is useful to not see part of the behaviour of some component.

The system view semantics is obtained from the strong time semantics by retaining from the transition system only the paths labelled with methods and transitions that we want to observe. Therefore, we need first to choose a set $View \subseteq (M \cup T)$ of methods and transitions that we want to observe. On the basis of $View$, we define the set of *observed* events, in a similar manner to the observable events.

Definition 16. *Observed Events.*
Let Sys be a real-time synchronised Petri nets system. Let $View \subseteq (M \cup T)$ be the set of methods and transitions that we actually want to observe. The set of observed events of Sys, denoted by $View_{Sys}$, is the least set such that:

$$View \subset View_{Sys}$$
$$e_1, e_2 \in View_{Sys} \;\Rightarrow\; e_1 \;//\; e_2 \in View_{Sys}$$
$$e_1, e_2 \in View_{Sys} \;\Rightarrow\; e_1 \;..\; e_2 \in View_{Sys}$$
$$e_1, e_2 \in View_{Sys} \;\Rightarrow\; e_1 \oplus e_2 \in View_{Sys}\ .$$

An observed event has the same structure as an observable event. However only methods and transitions being part of the $View$ set can appear in an observed event.

The system view semantics is then obtained from the expanded transition system, trs_{expand}, by retaining the transitions containing observed events only, and whose markings are reachable from the initial marking.

Definition 17. *System View Semantics.*
Let $(Sys, mark_k)$ be a marked real-time synchronised Petri nets system, let $View \subseteq (M \cup T)$, trs_{expand} be the expanded transition system obtained with the rules of Definition 14. The system view semantics of $(Sys, mark_k)$, denoted by Sem_{View}, is given by the least set of paths such:

$$(mark_k \xrightarrow[t]{e} mark') \in trs_{expand} \;\wedge\; e \in View_{Sys}$$
$$\Rightarrow (mark_k \xrightarrow[t]{e} mark') \in Sem_{View}$$

$$(mark'_1 \xrightarrow[t]{e} mark_2) \in trs_{expand} \wedge e \in View_{Sys} \wedge$$
$$\exists\, p \in Sem_{View}\ s.t.\ tail(p) = (mark_1 \xrightarrow[t']{e'} mark'_1) \ \wedge\ t' \leq t$$
$$\Rightarrow p\,(mark'_1 \xrightarrow[t]{e} mark_2) \in Sem_{View}\ .$$

The system view semantics Sem_{View} is actually a subset of Sem, where we remove all the branches of the semantics tree labelled with events made with methods or transitions that are not part of $View$. Paths of Sem_{View} are made of transitions whose events are exclusively those that we want to observe.

It is important to note that even though a method `m` is not part of $View$, its behaviour is taken into account if it is requested for synchronisation by another method `m' ` $\in View$.

3.6 Example

Figure 2 shows a partial view of the tree of reachable markings of the strong time semantics of the real-time synchronised Petri nets system of Figure 1.

The initial marking is made of three tokens present at time 0 in place `p1`, and the empty place `p2`; it is denoted $\{0,0,0\},\emptyset$. Method `move` has to occur between time 5 and time 10 (because of `put`), and method `put` has to occur between time 2 and time 10. In this example, the relative time intervals are also the absolute ones, since `move` and `put` are enabled at 0, and no token may arrive in their pre-set. The figure shows several cases. For instance, the firing of `move` occurring at time 5 leads to a new marking where a token has been removed from `p1`, and a token stamped at 5 is in `p2`. This corresponds to transition: $\{0,0,0\},\emptyset \xrightarrow[5]{\text{move}} \{0,0\},\{5\}$.

The formula below shows how the inference rules are applied in order to obtain this transition:

$$\dfrac{\dfrac{}{\{0,0,0\},\emptyset \xrightarrow[5]{\text{move with put}} \{0,0\},\emptyset}\,\text{BasicSyncBeh} \qquad \dfrac{}{\emptyset,\emptyset \xrightarrow[5]{\text{put}} \emptyset,\{5\}}\,\text{BasicBeh}}{\{0,0,0\},\emptyset \xrightarrow[5]{\text{move}} \{0,0\},\{5\}}\,\text{Sync}\ .$$

First, BasicBeh and BasicSyncBeh are applied. The obtained transitions are part of the weak transition system. Then rule Sync is applied on the two transitions. The resulting transition is then part of the strong time semantics, since it starts from the initial marking.

If we come back to Figure 2, the firing of `move` must be followed by the firing of the `processing` transition, because of the inhibitor arc. Since `processing` is enabled at 5, it can fire in the time interval `[6..14]`. For instance, it fires at time 6, and `p2` becomes empty. At this point, `put` can fire alone, or `move` can fire (for instance at time 7), but still within their absolute time interval. After the second firing of `move`, there is necessarily a firing of `processing`, which can occur at time 8 or after. Then, we can have either a firing of `move` or a firing of `put`. For

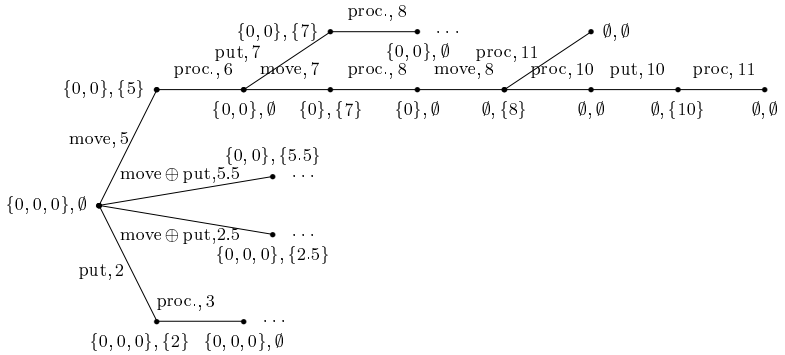

Fig. 2. Tree of reachable markings - Strong Time Semantics

instance, `move` fires a last time at 8 (all tokens in the pre-set are consumed), then it is followed by `processing`. If `processing` occurs between 9 and 10, then `put` has to occur at the latest at 10, then `processing` occurs a last time at 11 or after. If `processing` occurs after 10, then `put` cannot fire since the current time is over its time interval. In both cases, the system has reached its end, no method or transition may fire. From the initial marking, it is also possible to fire `put` at time 2 for instance, thus producing a new marking $\{0,0,0\},\{2\}$; or to fire the alternative event `move` $\oplus$ `put`. The tree shows the case of `move` $\oplus$ `put` firing at 2.5, corresponding to the firing of `put`, or at 5.5, corresponding to the firing of `move`. It is not possible to fire event `move` // `put`. Since `move` requires the firing of `put`, an event such as `move` // `put` requires that `put` fires twice simultaneously. Method `put` *cannot* fire two or more times simultaneously because the inhibitor arc is set to 0. The tree depicted by Figure 2 is not complete: it is also possible to fire `move` at any time in the interval `[5..10]`; and to fire `put` at any time in the interval `[2..10]`.

Figure 3 shows the system view semantics, with $View = \{$`move`,`processing`$\}$. It is a subset of the tree of Figure 2, where method `put` alone cannot fire.

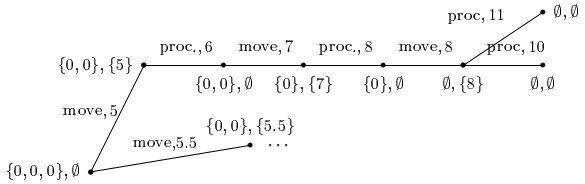

Fig. 3. Tree of reachable markings - System View Semantics

Remark 6. Zeno Behaviour. The semantics given above allows cases, where the number of sequential firings of a transition or a method, at a given time t, may be uncountable. These cases occur for transitions or methods having no pre-set, and no inhibitor arc, or when intervals [0..0] are used. It is recommended,

to insert extra places with pre-sets, or inhibitor arcs, in order to prevent time stuttering.

4 An Applicative Example: the Railroad Crossing System

In this section we illustrate the applicability of the real-time synchronised Petri nets to practical cases through a fairly classical "benchmark" for real-time system formalisms, i.e., the Generalised Railroad Crossing (GRC) system [8].

The GRC system consists of one or more train trails which are traversed by a road. To avoid collisions between trains and cars a bar is automatically operated at the crossing. Let us call I the portion of train trails which crosses the road. To properly control the bar, sensors are placed on the trails to detect the arrival and the exit of trains: the arrival of a train must be signalled somewhat in advance wrt the train entering region I, whereas the exit is signalled exactly when a train exits I. We call R the portion of train trails included between the place where entering sensors are placed and the beginning of I (see Figure 4). All trains have a minimum and a maximum speed so that they take a minimum and a maximum, yet finite and non-null, time to traverse R and I.

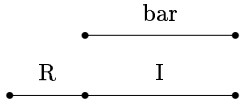

Fig. 4. Critical Section

The control of the bar operates as follows. Whenever a train enters R, this is detected and signalled by a sensor; similarly when a train exits I. If a train enters R and the bar is open (up), then a command is issued to the bar to close. This takes a fixed amount of time (γ). As soon as no more trains are in R or in I (this must be computed by the control apparatus on the basis of entering and exiting signals) the opening command is issued to the bar, which again takes γ time units to open (notice that, in this description, we assume that if a train enters R while the bar is opening, the control must wait until the bar is open before restarting closing it). The designer's job is to set system parameters (e.g., the length of R and the duration γ) in such a way that the following properties hold:

- Safety property: when a train is in I the bar is closed;
- Utility property: the bar is closed only for the time that is strictly necessary to guarantee the safety property.

Let us now formalise the GRC system through real-time synchronised Petri nets. Figure 5 shows two objects: `Train` and `Level Crossing`. The `Train` object represents the entry and the exit of trains into a critical region: a train enters into the critical region with method `entry`, it stays first in the section corresponding to R (place `p2`), for a certain amount of time (at least t_1). Then, it enters region I, represented by place `p3`, and finally it leaves the critical region with method

exit. Several trains may be simultaneously in the critical region, however their entry is not simultaneous. Indeed method entry can fire more than once, since place p1 contains always one token. However, method entry cannot fire twice or more simultaneously, and there is a delay of at least t'_1 between two trains entering the critical region. The fact that two trains may be simultaneously in region R or in region I depends also on the values of $t_1, t_2, t'_1, t''_1, t''_2$. Indeed, if $t_2 + t''_2 < t'_1$ then there will be at most one train in the critical region.

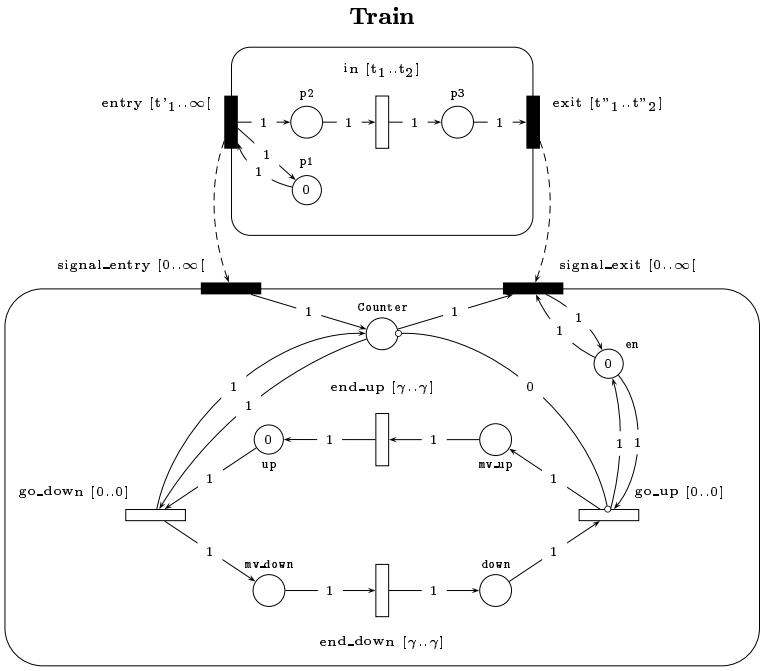

Fig. 5. Train and Level Crossing

The Level Crossing object represents the behaviour of the critical region, i.e., the bar's behaviour: it must be up iff no train is currently in the critical region, or entering it.

Each time a train enters the critical region, the signal_entry method fires. This is due to the fact that signal_entry is requested by entry. The signal_entry method increases the number of tokens in place Counter, whose role is to count the number of trains that are currently in the critical region. If the barrier is up and if a train arrives in the critical region, transition go_down fires and the barrier begins to go down (place mv_down). After a certain amount of time γ, represented by transition end_down, the barrier is finally down (place down).

Each time a train leaves the critical region, by activating method exit, the signal_exit method fires simultaneously. This method simply decreases by one the number of trains that are currently in the critical region. As soon as there are no more trains in the critical region, i.e. place Counter is empty, transition

go_up fires. Indeed, the inhibitor arc of weight 0 prevents the firing of go_up before all trains leave the critical region, and time interval [0..0] attached to go_up enforces go_up to fire as soon as it is enabled. The place en maintains the time of exit of the last train. Then, the barrier begins to go up (place mv_up), and after a certain amount of time γ, represented by transition end_up, it arrives in the up position (place up).

When trains are in the critical region, and the barrier is already down, neither method go_down nor method go_up can fire. Therefore, the barrier remains down until all trains have leaved the critical region.

For the System view semantics we chose $View$ = {entry,exit,go_down, end_down,go_up,end_up}. We do not consider paths where methods signal_entry and signal_exit fire without being requested by methods entry and exit respectively. Indeed, train and level crossing are obviously two different objects. However, the level crossing is such that the commands activating the bar are issued from the trains, i.e., methods signal_entry, and signal_exit fire whenever a train enters R or exits I respectively, but these two methods should not fire alone. Let us now see how the safety property is satisfied.

The sequence of transitions (1) shows an incorrect sequence[4] of transitions wrt the safety property.

$$\{.\} \xrightarrow[t]{\text{entry}} \{.\} \xrightarrow[t]{\text{go\_down}} \{.\} \xrightarrow[t+t_1]{\text{in}} \{.\} \xrightarrow[t+\gamma]{\text{end\_down}} \{.\} \qquad (1)$$

This sequence of transitions corresponds to the case a train entering region R at t (entry), and region I at $t+t_1$ (in) *before* the bar is down at $t+\gamma$ (end_down). This case occurs whenever $\gamma \geq t_1$. The first condition to impose on γ is then $\gamma < t_1$. The sequence of transitions (2) shows the correct sequence of the bar going down when one train enters R and $\gamma < t_1$. Sequence (2) shows as well the exit of the train at $t+t_1+t_1^{''}$ (exit), followed by the bar immediately beginning to go up, and finally reaching the up position $t+t_1+t_1^{''}+\gamma$ (end_up). Whenever a second train enters R at $t+t'$, after the bar is up, then the bar begins to go down normally, and the safety property is still satisfied.

$$\begin{aligned} &\{.\} \xrightarrow[t]{\text{entry}} \{.\} \xrightarrow[t]{\text{go\_down}} \{.\} \xrightarrow[t+\gamma]{\text{end\_down}} \{.\} \xrightarrow[t+t_1]{\text{in}} \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{exit}} \\ &\{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{go\_up}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{end\_up}} \{.\} \xrightarrow[t+t']{\text{entry}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{go\_down}} \{.\} \end{aligned} \qquad (2)$$

It is similar, if a second train enters region R while the first train has not yet leaved I: the bar remains down until the second train leaves I. However, condition $\gamma < t_1$ is no longer sufficient, if the second train enters region R just after the exit of the first train, but before the bar is up. Indeed, the bar has received the signal to go up, and it will have to completely go up before being able to come down. The first train enters R at t, leaves I at $t + t_1 + t_1^{''}$. The second train

[4] for space purposes, markings are not shown, only transitions and time of firing are represented.

enters R at $t+t'$ ($t_1+t_1^{''} \leq t'$). Since the first train exits at $t+t_1+t_1^{''}$, and the second train is not yet in R, the bar begins to go up at $t+t_1+t_1^{''}$, reaches the up position at $t+t_1+t_1^{''}+\gamma$. Since in the meanwhile, the second train has entered R, the bar immediately begins to go down, and reaches the down position at $t+t_1+t_1^{''}+2\gamma$.

The incorrect case (3) occurs whenever the second train arrives after the exit of the first train: $t_1+t_1^{''} \leq t'$; and the bar is down (`end_down`) *after* the train has entered I (`in`): $t'+t_1 \leq t_1+t_1^{''}+2\gamma$, i.e., $t' \leq t_1^{''}+2\gamma$.

By combining these two conditions, we obtain the following equation: $t_1+t_1^{''} \leq t' \leq t_1^{''}+2\gamma$. It reduces to: $t_1+t_1^{''} \leq t_1^{''}+2\gamma$, and finally $t_1/2 \leq \gamma$. Therefore, in order to prevent the incorrect behaviour, γ must be such that: $\gamma < t_1/2$.

$$\begin{array}{l} \{.\} \xrightarrow[t]{\text{entry}} \{.\} \xrightarrow[t]{\text{go\_down}} \{.\} \xrightarrow[t+\gamma]{\text{end\_down}} \{.\} \xrightarrow[t+t_1]{\text{in}} \\ \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{exit}} \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{go\_up}} \{.\} \xrightarrow[t+t']{\text{entry}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{end\_up}} \\ \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{go\_down}} \{.\} \xrightarrow[t+t'+t_1]{\text{in}} \{.\} \xrightarrow[t+t_1+t_1^{''}+2\gamma]{\text{end\_down}} \{.\} \end{array} \tag{3}$$

In the correct case (4), the bar reaches the down position *before* the second train arrives in I.

$$\begin{array}{l} \{.\} \xrightarrow[t]{\text{entry}} \{.\} \xrightarrow[t]{\text{go\_down}} \{.\} \xrightarrow[t+\gamma]{\text{end\_down}} \{.\} \xrightarrow[t+t_1]{\text{in}} \\ \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{exit}} \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{go\_up}} \{.\} \xrightarrow[t+t']{\text{entry}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{end\_up}} \\ \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{go\_down}} \{.\} \xrightarrow[t+t_1+t_1^{''}+2\gamma]{\text{end\_down}} \{.\} \xrightarrow[t+t'+t_1]{\text{in}} \{.\} \end{array} \tag{4}$$

The utility property is simpler to see. Whenever the last train leaves I while the bar is already down, the inhibitor arc and the interval [0..0] attached to `go_up` guarantee that the bar begins to go up immediately when no more trains are in the critical region. Problems arise when the last train leaves I while the bar is *not yet* in the down position, i.e., transition `end_down` fires *after* the `exit` of the last train. The sequences of transitions (5) and (6) below show two occurrences of this case. Sequence (5) shows the case of a single train, entering R at t, and leaving I at $t+t_1+t_1^{''}$ before the bar reaches the down position. This is a special case of (1), it shows a worst behaviour, since the train not only enters I, but even leaves I before the bar is in the down position. Requiring $\gamma > t_1$ is sufficient to avoid this case.

$$\{.\} \xrightarrow[t]{\text{entry}} \{.\} \xrightarrow[t]{\text{go\_up}} \{.\} \xrightarrow[t+t_1]{\text{in}} \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{exit}} \{.\} \xrightarrow[t+\gamma]{\text{end\_down}} \{.\} \tag{5}$$

Sequence (6) shows a particular case of (3), when two trains are involved. As in (3) the second train enters R soon after the exit of the first train: $t_1+t_1^{''} \leq t'$. Due to the bar going up and down, the second train leaves I before the bar is down: $t'+t_1+t_1^{''} \leq t_1+t_1^{''}+2\gamma$, i.e., $t' \leq 2\gamma$.

$$\begin{aligned} \ldots\{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{exit}} \{.\} \xrightarrow[t+t_1+t_1^{''}]{\text{go\_up}} \{.\} \xrightarrow[t+t']{\text{entry}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{end\_up}} \\ \{.\} \xrightarrow[t+t'+t_1]{\text{in}} \{.\} \xrightarrow[t+t_1+t_1^{''}+\gamma]{\text{go\_down}} \{.\} \xrightarrow[t+t'+t_1+t_1^{''}]{\text{exit}} \{.\} \xrightarrow[t+t_1+t_1^{''}+2\gamma]{\text{end\_down}} \{.\} \end{aligned} \tag{6}$$

By combining both equations we have: $t_1 + t_1^{''} \leq t' \leq 2\gamma$, which reduces to: $\gamma \leq (t_1 + t_1^{''})/2$. Therefore, in order to avoid this we must enforce: $\gamma < (t_1 + t_1^{''})/2$. The requirement needed for the safety property: $\gamma < t_1/2$ is sufficient to ensure the utility property too.

The GRC example, though still rather simple, illustrates the suitability of the model for the description of real-time systems. First, modularisation is naturally achieved through the definition of several objects and the use of the synchronisation mechanism to formalise their interaction. Second, the use of inhibitor arcs allows to achieve a good level of generality without resorting to more sophisticated models. In our case inhibitors arcs allow a natural formalisation of the counter mechanism which is essential for a proper description of the system. Notice that pure Petri nets allow only the modelling of simplified versions of the GRC system (e.g. the case where only one train can traverse the regions R and I per time) whereas in order to deal with the general case, more cumbersome formalisations are usually needed [8].

5 Related Works

The model of Communicating Time Petri Nets [3] is close to the one presented in this paper. It combines inhibitor arcs, attaches firing time intervals to transitions, and allows decomposition into modules. It differs in the composition of modules, which is realised through a message passing-based communication among modules. This work has also resulted in an analysis technique of the overall system based on an analysis of each individual modules. The combination of time and Petri nets modules has been also addressed in [9] which introduces the Time Petri Box calculus. Modules are Petri Boxes equipped with a dynamic firing time interval over a discrete time domain, and composition is realised by the means of several operators. The analysis method for single Time Petri nets of [1] is based on a "state-class" technique and allows to build a finite representation of the behaviour of the nets, enabling a reachability analysis similar to that of Petri nets.

6 Conclusion

We introduced *real-time synchronised Petri nets*, a class of high-level Petri nets that combines modularity and abstraction mechanisms (transactional view of synchronisations) proposed by CO-OPN, inhibitor arcs, and a delay time model for Petri nets (using relative time intervals). This paper does not address problems related to reachability analysis, even though the defined semantics constitutes a basis for elaborations of analysis techniques and tools. Efforts towards this direction have already been realised and an operational semantics is now

available [12]. It relies on a symbolic technique that enables to build a finite representation of temporal constraints, which actually defines an uncountable number of state spaces and firings. Future work will concentrate on giving an axiomatisation to these nets in order to enable formal verification of logical properties (safety, liveness), and on pursuing further generalisation. Once the axiomatisation will be available, supporting theorem proving tools will be applicable to mechanise and make system analysis more robust [7].

References

1. B. Berthomieu. La méthode des Classes d'États pour l'Analyse des Réseaux Temporels - Mise en Oeuvre, Extension à la multi-sensibilisation. In *Modélisation des Systèmes Réactifs, MSR'2001*. Hermes, 2001.
2. O. Biberstein, D. Buchs, and N. Guelfi. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In G. Agha, F. De Cindio, and G. Rozenberg, editors, *Advances in Petri Nets on Object-Orientation*, volume 2001 of *LNCS*, pages 70–127. Springer-Verlag, 2001.
3. G. Bucci and E. Vicario. Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets. *IEEE Transactions on Software Engineering*, 21(12):969–992, 1995.
4. D. Buchs and N. Guelfi. A formal specification framework for object-oriented distributed systems. *IEEE Transactions on Software Engineering, Special Section on Formal Methods for Object Systems*, 26(7):635–652, July 2000.
5. G. Di Marzo Serugendo, D. Mandrioli, D. Buchs, and N. Guelfi. Real-time synchronised Petri nets. Technical Report 2000/341, Swiss Federal Institute of Technology (EPFL), Software Engineering Laboratory, Lausanne, Switzerland, 2000.
6. M. Felder, D. Mandrioli, and A. Morzenti. Proving properties of real-time systems through logical specifications and Petri net models. *IEEE Transactions on Software Engineering*, 20(2):127–141, February 1994.
7. A. Gargantini and A. Morzenti. Automated Deductive Requirements Analysis of Critical Systems. *ACM TOSEM - Transactions On Software Engineering and Methodologies*, 10(3):255–307, July 2001.
8. C. Heitmeyer and D. Mandrioli, editors. *Formal methods for real-time computing*. John Wiley & Sons, 1996.
9. M. Koutny. A Compositional Model of Time Petri Nets. In M. Nielsen and D. Simpson, editors, *International Conference on Application and Theory of Petri Nets 2000*, volume 1825 of *LNCS*, pages 303–322. Springer-Verlag, 2000.
10. C. A. Lakos and S. Christensen. A General Systematic Approach to Arc Extensions for Coloured Petri Nets. In R. Vallette, editor, *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *LNCS*, pages 338–357. Springer-Verlag, 1994.
11. P.M. Merlin and D.J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.
12. S. Souksavanh. Operational Semantics for Real-Time Synchronized Petri Nets. Master's thesis, Swiss Federal Institute of Technology (EPFL), Software Engineering Laboratory, Lausanne, Switzerland, 2002.