Petri nets with causal time for system verification

Cécile Bui Thanh, Hanna Klaudel, Franck Pommereau

Electronic Notes in Theoretical Computer Science 68 No. 5 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 16 pages

Petri nets with causal time for system verification

C. Bui Thanh [1], H. Klaudel [2] and F. Pommereau [3]

Université Paris 12, LACL
61 avenue du général de Gaulle
94010 Créteil, France.

Abstract

We present a new approach to the modelling of time constrained systems. It is based on *untimed* high-level Petri nets using the concept of *causal time*. With this concept, the progression of time is modelled in the system by the occurrence of a distinguished event, *tick*, which serves as a reference to the rest of the system. In order to validate this approach as suitable for automated verification, a case study is provided and the results obtained using a model-checker on high-level Petri nets are compared with those obtained for timed automata using prominent tools. The comparison is encouraging and shows that the causal time approach is intuitive and modular. It also potentially allows for efficient verification.

1 Introduction

This paper presents a case study in modelling and verification of systems with time constraints. We use an original approach based on *untimed* high-level Petri nets, using a concept of so called *causal time* [17], inspired by [5,18]. This widely differs from the classical approaches where time is introduced in Petri nets in terms of intervals or durations labelling nets elements, as in time or timed Petri nets (see [4] for a survey and a comparison of the different approaches), referring to a progression of time external to the system. The main characteristic of the causal time approach is that the progression of time is modelled *in the system* by a distinguished event, called *tick*. Thus, the occurrences of the other events may depend on the occurrences of *tick*. The time constraints of the kind "at most" or "at least 5 ticks between events t and t'" are realized by counting the appropriate number of ticks between the

[1] Email: bui@univ-paris12.fr
[2] Email: klaudel@univ-paris12.fr
[3] Email: pommereau@univ-paris12.fr

occurrences of t and t'. So, the occurrence of t' is causally dependent on those of *tick* and may only occur if the time constraint is satisfied. The modelled system and the counter of ticks are both represented by high-level Petri nets interacting with each other.

We use a model of high-level Petri nets provided with a structure of process algebra, the *algebra of M-nets* [2], in which Petri nets can be composed together with operators like sequential and parallel composition. The model also allows for synchronous communication, as in CCS [16]. In this context, introducing causal time amounts to consider a net expressing a tick counter being able to interact with the system and to produce the required number of ticks between occurrences of transitions (as proposed for instance in [12]).

The main goal of this paper is to show that the causal time approach in this context allows one to model systems in an intuitive and modular way, with the potentiality of efficient verification. For this purpose, we present a comparative case study concerning the railroad crossing problem and give its specification in terms of timed automata as well as in terms of high-level nets with causal time. Various versions of the specification having different properties (for instance the absence or presence of deadlocks) are then verified using model-checkers Kronos [20] and Uppaal [13] for the timed automata, and MARIA [15] for the high-level Petri nets. The results obtained are very promising since in many cases, the causal time approach allows for a more efficient verification. At the end of the paper, we discuss the current limitations concerning the approach and the tools, and we point out some ways which can lead to significant improvements.

Throughout the paper, we assume that the reader has basic knowledge about timed automata [1,9] and coloured Petri nets [10,2].

2 Railroad crossing system (RC)

The railroad crossing system is composed of n_t trains (each of them moving on its own track) and of a pair of gates which prevent cars from crossing the tracks when a train is present.

The trains move independently and, initially, none is present. Each train starts far from the railroad crossing; it triggers a signal *app* when it approaches close enough to the gates. From this point, it reaches the gates in at least a_m and at most a_M time units. Then, it passes inside the gates during at least e_m and at most e_M time units and finally leaves the gates triggering a signal *exit*.

The gates are initially open. They close in at least g_m and at most g_M time units after receiving a signal *down*. They require the same delay for opening after receiving a signal *up*. It may happen that the gates receive the signal *down* when they are already going up; in this case also, the time needed in order to close is in the same boundaries.

A controller receives the signals from the trains and reacts by sending signals to the gates in at least c_m and at most c_M time units. It must ensure

the *safety property* which states that if a train is present at the crossing, then the gates must be closed.

The purpose of this paper is to show the usability of the causal time approach and to compare its performances with timed automata. This, we will use a simplified specification of the railroad crossing problem. For instance, we do not verify the *availability property* (gates are open as much as possible).

3 A modelling of RC with timed automata

We consider here a version of timed automata [1] which allows, in particular, for *state invariants* [9], *integer variables* (in addition to *clocks* which take real values) and *binary synchronisations*. A state invariant is a condition involving clocks and variables which must be true while the automaton stays in this state. Invariants are often used to express deadlines, for instance, $c \leq max_c$ labelling a state s means that the maximal value of the clock c in s is max_c. A transition label contains three parts separated by bars: a condition called a *guard*, a communication action (such as $act!$ or $act?$, expressing respectively a sending and a receiving on a canal act) and an expression specifying the clocks to be reset and the integer variables to be modified. For instance,

$$c \geq min_c \mid act! \mid c := 0; n := n + 1$$

indicates that the transition is possible if c is greater than min_c; if it occurs, signal $act!$ is sent, clock c is reset and the variable n is incremented. Timed automata may be composed using synchronised product inducing the synchronisation of complementary actions (like $act!$ and $act?$).

It is easy to give a modelling of RC with timed automata. The variant presented here is depicted in figure 1. A train is modelled by the automaton *Train* and the gates by the automaton *Gate*. The link between trains and gates is obtained by the automaton *Controller*. The complete specification is the synchronised product of *Gate*, *Controller* and n_t copies of the automaton *Train*.

Initially, the controller is idle, and the variable n is set to 0, the gates are open and all the trains are far from them. If a train approaches, the controller receives a signal *app* and reacts sending *down* to the gates and incrementing n. When a train leaves the crossing, it sends *exit* to the controller which decrements n. If it was equal to 1, then, *up* is sent to the gates, otherwise, no special reaction is needed.

One may notice that when the controller is in state AppDown it cannot receive any signal (*app* or *exit*) and delays their reception until it reaches the state Idle. This is unrealistic since trains cannot be stopped; however, we preferred to use this simplified version since our goal is more a comparison than a complete case study.

The tools used for this work are Kronos [20] and Uppaal [13] because they have the reputation to offer efficient verification. Deadlock freeness and safety

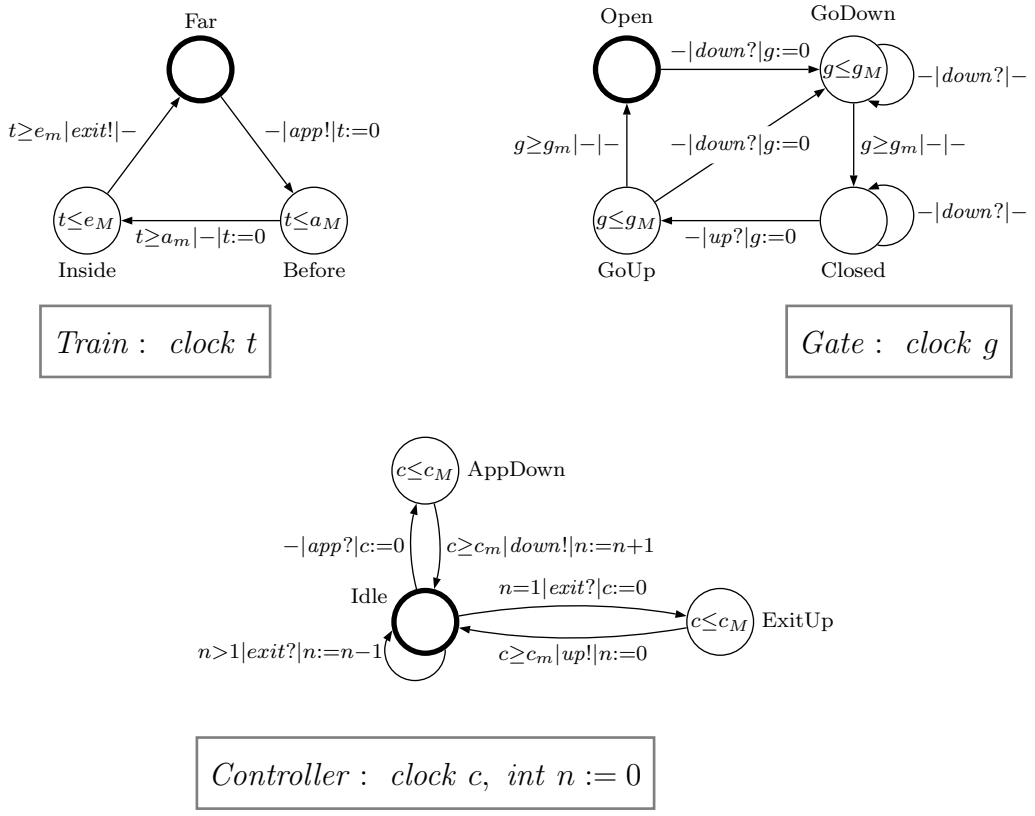

Fig. 1. The timed automata *Train*, *Gate* and *Controller*. The initial states are depicted with bold circles.

properties may be expressed through temporal logic formulas; for instance, with Uppaal, we have:

$$\forall\Box(\neg \mathit{deadlock}) \wedge ((\mathit{Train}_1.\text{Inside} \vee \cdots \vee \mathit{Train}_{n_t}.\text{Inside}) \Rightarrow \mathit{Gate}.\text{Closed}) \quad .$$

The automata presented above are directly usable with Uppaal; a non-trivial translation is necessary in order to adapt them for Kronos. Indeed, this tool uses a lower-level model without integer variables and with multi-way synchronisation. So, the automata used for Kronos are much more complicated than those presented above but they are functionally equivalent. Notice also that while Uppaal have a very nice user-friendly interface, Kronos is much more a low-level tool.

4 Composable high-level Petri nets with causal time

In this paper, we use modular high-level Petri nets, called M-nets [2], which are well suited for specifying large concurrent systems. As usual for high-level nets, their places, transitions and arcs are annotated in a specific way. In the simplest case each place has a *type* which is the set of values (tokens) it can hold; each arc is labelled by a multi-set of expressions (the simplest ones

being just values or variables); and each transition carries a *guard* which is a boolean expression playing the role of an execution condition.

An example of such a marked high-level net is shown in figure 2. This net may evolve by *firing transitions*. During the execution, the variables in the guards and in the arc annotations are *bounded* to values. A transition may fire if its guard is true and if the arcs carry only tokens belonging to the types of adjacent places. A possible execution of the net of figure 2 starts by firing t_1, which consumes the token $\bullet$ from its unique input place and produces a new marking composed of a token $\bullet$ and a token 0, each in the corresponding output place of t_1. Then, the transition t is the only enabled because of the guard of t_2 which is false for the *binding* associating z to 0, denoted $\{z \mapsto 0\}$. The firing of t with the binding $\{x \mapsto 0\}$ consumes 0 and produces 1 instead. One more firing of t is possible producing the marking 2 in its output place. Then, t_2 becomes enabled with the binding $\{z \mapsto 2\}$ and its firing consumes tokens $\bullet$ and 2 from its input places and produces $\bullet$ in the output place. With this marking, the net may not evolve anymore.

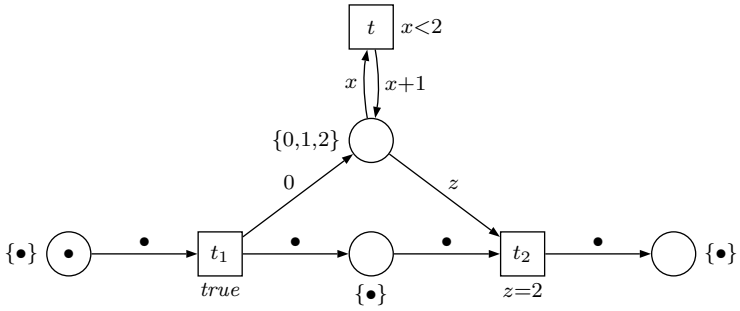

Fig. 2. A high-level Petri net.

The behaviour of an initially marked high-level Petri net may be described by a *reachability graph* whose nodes are the reachable markings and whose arcs correspond to the bounded transitions allowing to produce one marking from another. The set of all paths (starting from the initial marking) in this graph corresponds to an *interleaving semantics* of the Petri net. Several concurrent semantics may be considered, including step [7] or partial order semantics [14,6], however they are not considered in this paper.

These high-level nets may be composed in parallel by simply putting the nets side by side. They may also by synchronised (using the operation called *scoping*) in order to enforce all synchronous communications between transitions. For this purpose, we consider for the high-level nets used in this paper an labelling on transitions allowing for synchronisations. Some examples of such extended initially marked nets are given in figure 4. Their transitions are decorated by additional *labels* (the guards being as before) which are multi-sets of CCS-like communication actions (possibly with arguments which are variables or constants) as, for instance, app, $down$, $clock(x, a, b)$ or $\widehat{app}$, $\widehat{down}$, $\widehat{clock}(z, 2, c)$.

Notice that the following are always omitted in the figures: empty transition labels; guards which are always satisfied; arcs inscriptions and place types of the form $\{\bullet\}$.

The parallel composition of nets N_{Ga}, N_{Tr}, N_{Co} and N_{Cl} is $ParSys = N_{Ga} \| N_{Tr} \| N_{Co} \| N_{Cl}$ represented in figure 4. The scoping (which is a synchronisation followed by a restriction) is illustrated in figure 3; it is applied to a fragment of the net $ParSys$ with transitions t_1, t_0 and t_4 coming from nets N_{Ga}, N_{Cl}, and N_{Tr}, respectively. The synchronisation of $ParSys$ w.r.t. action $clock$ yields new transitions: t_{10} (gluing t_1 and t_0) and t_{04} (gluing t_0 and t_4). These new transitions are obtained in several steps. First, the variables appearing in the surroundings of t_1, t_0 and t_4 are renamed in order to avoid name clashes. This is necessary because, by synchronisation, these surroundings are combined into a single one. Then, a new transition is created for each pair of actions $clock$ and $\widehat{clock}$ if there is an unifier for their arguments. For instance, $\{z \mapsto x, t \mapsto c_1, c_2 \mapsto 0\}$ is a unifier allowing to synchronise t_1 and t_0. Finally, the guard of the new transition is the conjunction of the two constituent substituted guards; its label is the multi-set sum of the two constituent substituted labels, without the matching pair of actions; the arcs are all those of both former transitions (with substituted inscriptions). A restriction of the resulting net w.r.t. $clock$ gives a net in which all transitions whose labels contain at least one action $clock(\cdots)$ or $\widehat{clock}(\cdots)$, together with their surrounding arcs, are deleted, see the right hand side of figure 3 which corresponds also to the scoping of the net w.r.t. $clock$, denoted $ParSys \,\mathsf{sc}\, clock$.

Scoping may be applied with respect to a set of actions (because synchronisation is commutative and so is restriction [2]). Moreover, scoping w.r.t. action $clock$ is possible even if a transition holds several instances of this action as on t_2 in 4. In such a case, one action, say $clock(y, t, \omega)$, is first chosen for synchronisation, leading to a new transition which still holds the second action (here, $clock(y', t', 0)$). This new transition is then synchronised, yielding a new transition holding without action $clock$ and inheriting the arcs from t_2 and two pairs of arcs from t_0 (one pair for each synchronisation).

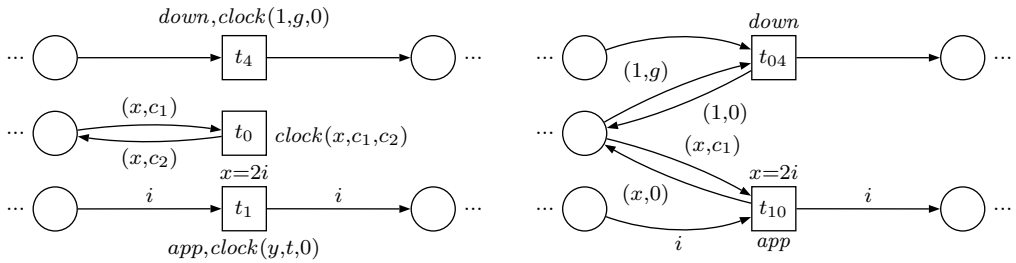

Fig. 3. A fragment of the net $ParSys$ (on the left) and a fragment of $ParSys \,\mathsf{sc}\, clock$ (on the right).

4.1 Introduction of causal time in high-level Petri nets

The basic idea behind the concept of causal time is to represent the occurrence of successive ticks (modelling the progression of time) in the same way as any other event in the system. In the context of Petri nets, events are represented by occurrences of transitions, and so, a time scale may be built by the firing of some reference transition, called *tick*. For instance, figure 2, represents a time constrained system composed of transitions t_1 and t_2 where two occurrences of t (representing the tick) are enforced between those of t_1 and t_2.

It is possible to temporally constrain a given system in a modular way. The approach consists in considering a particular net modelling a *clock*, being able to generate occurrences of ticks, to evolve in parallel to the system and to synchronise with it in order to enforce some temporal constraints. If the system has more than one independent time constraint, the clock net should be capable to manage several counting requests concurrently.

For the railroad crossing problem, we consider the clock net N_{Cl}, represented on the bottom of figure 4; it manages n_c+1 counting requests. Initially, the place Time carries $n_c + 1$ pairs of the form (j, c) where $j \in \{0, \ldots, n_c\}$ is the number of the request and c is the current value of the corresponding tick counter. Each request j has a fixed maximum value of its tick counter, max_j, which cannot be overtaken. A tick counter $c = \omega$ for some request means that this request is unused. The constant ω is assumed to be equal to $max + 1$, where max is the maximum of all the max_j's (for $0 \leq j \leq n_c$), and we state $\omega + 1 = \omega$. The transition *tick* may occur at any time provided that its guard is true (which is the case if all the temporal constraints are fulfilled and will still be true after the tick, and, in particular, if no max_j is reached). The occurrence of *tick* increments the tick counters of all requests. Initially, all the requests are unused and can be started at any time by the firing of a transition coming from the synchronisation w.r.t. *clock*.

5 A modelling of RC using Petri nets with causal time

RC is modelled by the net:

$$ParSys \,\mathsf{sc}\, \{clock, down, up, app, exit\} \quad .$$

The resulting net has the same places as *ParSys* but different transitions coming from the scoping w.r.t. all the communication actions. The scoping w.r.t. *up* and *down* ensures that the gates move exactly as the controller allows it. Analogously, the scoping w.r.t. *app* and *exit* enforces the communication between the trains and the controller. The scoping w.r.t. *clock* ensures that all counting requests are correctly handled.

The number of tick counters in clock N_{Cl} depends on the number of trains in the system because we use two counters for each train, with the following setting.

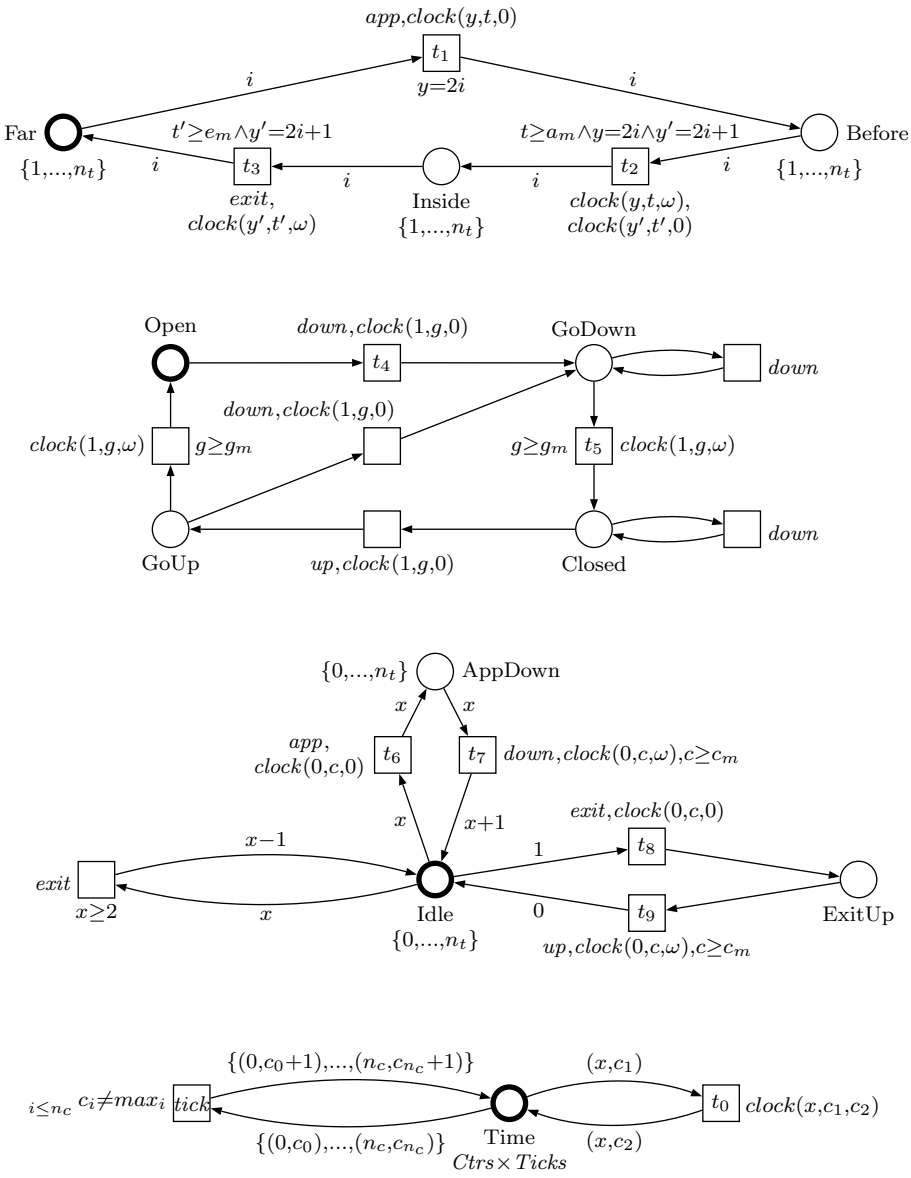

Fig. 4. The nets N_{Tr}, N_{Ga}, N_{Co} and N_{Cl} (from top to bottom, if taken separately), or their parallel composition, *ParSys* (if taken as a single net). In the figure, n_t is the number of trains, $n_c = 2n_t + 1$ is the greatest counting request number, $Ctrs = \{0, \ldots, n_c\}$ is the set of all these numbers, and $Ticks = \{0, \ldots, \omega\}$ is the set of the possible values of tick counters. Places in bold are initially marked as follows: $\{1, \ldots, n_t\}$ for Far; $\{\bullet\}$ for Open; $\{0\}$ for Idle; and $\{(0, \omega), \ldots (n_c, \omega)\}$ for Time.

The counter 0 is reserved to the controller, and its maximal value is $max_0 \stackrel{\text{df}}{=} c_M$ (see section 2). This counter is reset when a train is approaching (see transition t_6) and is used in order to ensure that signal *down* is sent to the gates after at least c_m ticks (see transition t_7). The maximum number of ticks allowed here, c_M, is enforced in the guard of transition *tick* in the clock. Then, the same counter is used once again (for a different purpose) when

the last train leaves the crossing (see transitions t_8 and t_9). Notice that if we have had different constraints in these two cases, we should have used two different counters. (This is not an intrinsic limitation of causal time but rather a limitation of the simple clock we choose to use.)

The counter 1 is reserved to the gates and its maximal value is $max_1 \stackrel{\text{df}}{=} g_M$. It is reset when the gates receive the signal to go down (see transition t_4) and it ensures that the gates are down after at least g_m and at most g_M ticks (see transition t_5 and the guard of *tick*). The same counter is used in order to ensure the opening of the gates under the same time constraints.

For each train i, for $i \in \{1, \ldots, n_t\}$, we use two distinct counters: $2i$ and $2i+1$, with $max_{2i} \stackrel{\text{df}}{=} a_M$ and $max_{2i+1} \stackrel{\text{df}}{=} e_M$, respectively. When a train approaches, at least a_m and at most a_M ticks can occur between the sending of signal *app* and the arriving of the train between the gates. This constraint is ensured by the counter $2i$ (see transitions t_1 and t_2). The counter $2i+1$ ensures that there must be at least e_m and at most e_M ticks between the crossing of the road by a train and its leaving sending the signal *exit* (see transitions t_2 and t_3). In particular, t_2 fires when the train enters the crossing; the counter $2i$ must then indicate a value greater than a_m (thanks to $t \geq a_m$ in the guard t_2) and the counter $2i+1$ is reset.

In this system, the controller holds only one token and is synchronised to all the other nets, and so, most events are interleaved. Moreover, almost all the transitions of the system are synchronised on action *clock* and thus the resulting transitions are in conflict with *tick*. This reduces again the concurrency in the system which is in fact purely sequential (which is suitable for a comparison with timed automata).

5.1 Tools used

We modelled the above specification using PEP toolkit [8] which proposes a lot of tools gathered in a convenient graphical interface. In particular, it allows one to edit high-level nets, to apply scoping on them and to convert the resulting nets into low-level (place/transition) nets which are suitable for verification using one of the model-checkers integrated with PEP. Unfortunately, we were not able to use PEP from the beginning to the end. The reason is mainly the size of the low-level nets equivalent to our high-level specification, which cannot be handled by PEP.

We used instead a high-level tool, MARIA [15], in order to check our specification against deadlock-freeness and safety. Such a tool does not need to produce low-level nets and thus it does not generate more than necessary, contrasting with the transformation from high-level nets to low-level ones which may generate, for instance, many places which will never be marked. This is particularly true in our specification, where place Time in N_{Cl} cannot hold arbitrary combination of tokens because the progression of time is not arbitrary itself and only a small subset of possible markings are actually reachable. MARIA works on the reachability graph of coloured Petri nets and allows

one to check for deadlocks and for the reachability of partial markings (sub-markings) during the generation of the graph. Deadlock freeness and the safety property could be expressed as:

```
deadlock fatal;
reject !(place Inside equals empty)
       && (place Closed equals empty) && fatal;
```

The first line specifies that if a deadlock is found, the computation of the reachability graph must be interrupted and the error reported. The rest specifies states which has to be rejected if reached. It is a C-like boolean expression on the marking of places, with lazy evaluation: if place Inside is marked and then, if place Closed is not marked, then `fatal` is evaluated, leading to abort the computation and to report the encountered rejected state.

The files produced by PEP have been converted to the file format supported by MARIA. Then, these files have been made generic, and so we are able to produce the specification for any number of trains and all kind of time constraints using a simple preprocessing. At the current state of the work, only a preprocessor and MARIA are involved in the generation and the verification of the railroad specification, but PEP was necessary in the first steps in order to produce the scoping of the nets.

6 Results

We report now the performances of the different tools during the deadlock analysis and safety verification of various versions of the specification. All the checks have been performed on a Sun Sparc station at 440Mhz, with 1Gb of physical memory and 1Gb of swap space. We worked in the `/tmp` directory which, thanks to Sun's TMPFS file system [19], is located in the virtual memory so all the work, even file accesses, was actually made in memory with proper swap. When a pre-compilation of some files has been necessary, the time consumed is included into the durations given below. This was the case for Kronos which needs to synchronise timed automata before to check them, and for MARIA which can build the guards of the transitions into libraries being then dynamically loaded by the tool in order to speed-up the evaluation. Finally, we used Unix `time` tool in order to measure the time consumed by each process (the "real" time is the one reported below).

We checked safe and deadlock-free systems for one to six trains with the following values for the different constants: $a_m = 4$, $a_M = 5$, $c_m = 0$, $c_M = 1$, $e_m = 4$, $e_M = 6$, $g_m = 0$ and $g_M = 2$. The times measured for each tool are reported in the top part of figure 5 (see also the left graph on figure 6). After about 12h30m, Uppaal exhausted all the memory and begun to be heavily swapped, using less than 1% of CPU, so we preferred to stop it since the reported time would have been meaningless.

Unsafe systems were produced with the same constants values as those used for good systems except for g_M which was here set to 3. Thus, the gates

trains	1	2	3	4	5	6
safe systems with no deadlock						
MARIA	0.2s	0.2s	0.9s	12s	4m	1h12m
Kronos	0s	0.1s	1.6s	20s	5m	1h36m
Uppaal	0.3s	0.5s	0.7s	27s	57m	-
unsafe systems with no deadlock						
MARIA	0.1s	0.2s	0.2s	0.2s	0.3s	0.4s
Kronos	0s	0.2s	1.7s	21.4s	6m57s	5h59m
Uppaal	0s	0s	0s	0s	0s	0s
deadlocking safe systems						
MARIA	0.1s	0.2s	0.2s	0.2s	0.3s	0.4s
Kronos	0s	0.2s	1.7s	21.4s	6m55s	6h02m
Uppaal	0s	0s	0s	0s	0s	0.1s

Fig. 5. The performances of the tools for "good" systems (top part), unsafe systems (middle part) and deadlocking ones (bottom part). We used specifications taking into account up to six trains.

could go down too slowly and a train could cross the road while they are not yet closed. The performances are given in the middle part of figure 5 (see also the right graph on figure 6). Notice that the line for Uppaal is correct: this tool was incredibly fast with wrong systems (*i.e.*, unsafe and deadlocking ones).

Systems with deadlock were produced with the same values as for good ones. We suppressed the capability for the gates to receive a signal *down* when being or going down, by removing two transitions in each specification. Notice that for one train only, this does not produce a deadlock. The performances are reported in the bottom part of figure 5 (see also the right graph on figure 6).

6.1 Causal time w.r.t. dense time and consistency of the results

Using causal time is very natural provided that one has in mind that time constraints are expressed with respect to a time scale built by a causal clock, *i.e.*, by the occurrence of a tick which is not itself directly observable (but its consequence on the marking can be observed). Therefore, the causal time is available through tick counters, which is fairly different from reading values on a dense (or real) time scale. For instance, if c is a tick counter, equation $c = 3$ on a causal time scale means of course that "exactly three ticks occurred", but this may mean also that a fourth tick is just about to occur. On a dense time scale, this would be expressed as $3 < c < 4$. Notice that we do not have $3 \leq c$

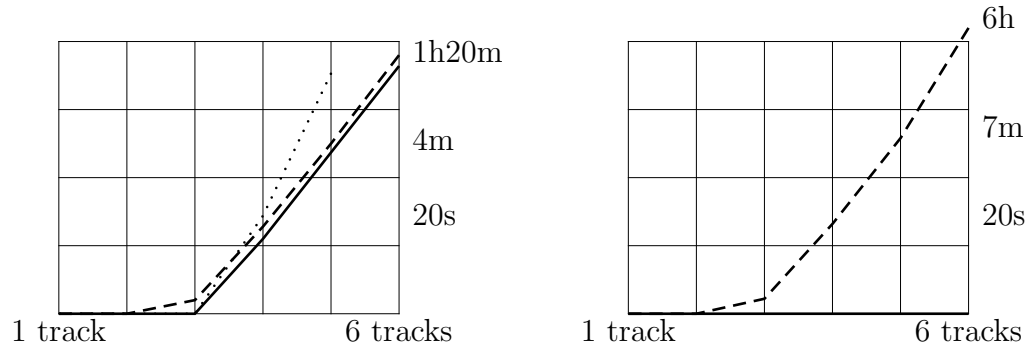

Fig. 6. The graphical representations of the performances measured for MARIA (continuous lines), Kronos (dashed lines) and Uppaal (dotted lines). The left graph is for good systems, the other for deadlocking or unsafe ones. Notice that the vertical scales are logarithmic. On the right graphics, lines for MARIA and Uppaal completely overlap.

because the third tick has to be counted, and thus must have occurred. (We assume that actions which cannot occur concurrently are not simultaneous, and thus are separated by a non zero delay.)

Another example is the segment $5 < c \leq 6$ on a causal time scale which corresponds to $6 < c < 7$ on a dense time scale. This is not surprising if one remembers that the first constraint has to be read as "strictly more than 5 and at most 6 ticks occurred" which corresponds to "at least 6 ticks and strictly less than 7 occurred".

One can see that causal time differs from real time in many ways. However, in our case study, we used for each specification the more natural expression of time constraints, regardless of the introduced differences. Actually, we conjecture that a wide class of timed automata can be translated this way and that we can have a bisimilarity relation between the automata and the translated M-nets. In order to verify in practice this intuition, and before to obtain theoretical results, we checked many different versions of RC for many different values of the constants and with or without deadlock. These results are not presented here since they do not give more information than what we already provided. But it is worth noting that all the checks were consistent. For instance, if a given set of constants led to an unsafe system with one tool, the same happened for the other tools. Moreover, the tools always reported equivalent counter examples. As an illustration, consider the unsafe system described above for 3 trains. MARIA reports a transition sequence leading to an unsafe state which corresponds to:

(i) The token 1 in place Far (identifying the first train) is moved to place Before while the token in the controller moves from Idle to AppDown.

(ii) One tick.

(iii) The token in the controller moves from AppDown to Idle and the token for the gates from Down to GoDown

(iv) Three ticks.

(v) Train 1 moves from Before to Inside.

With Uppaal (the case of Kronos is similar), we obtain a trace which corresponds to:

(i) Train 1 goes from state Far to state Before while the controller goes from Idle to AppDown.

(ii) Delay of 1 time unit.

(iii) The controller goes from AppDown to Idle while the gates goes from Open to GoDown.

(iv) Delay of 3 time units.

(v) Train 1 goes from Before to Inside.

One may notice that a delay of three ticks with Petri nets corresponds to a delay of exactly three time units with Uppaal (and actually with Kronos also).

6.2 State space explosion

In the results reported above, it happens that the performances obtained with MARIA are generally better than those obtained with the other tools. This optimistic results have to be moderated a little bit. Actually, using MARIA, the causal time approach suffers of the well known *state space explosion* problem: when we increase the constants in the system, the number of reachable markings increases very fast. Since MARIA explicitly generates these markings, its performances become very bad.

One way to alleviate this problem would be to abstract from the net the intermediary states generated by counting ticks between two boundaries. For instance, if counter c is used in order to ensure a constraint $1 \leq c \leq 6$, only values 1 and 6 are interesting for this counter. Removing the intermediary values would reduce the number of states while preserving the interesting behaviour. This would amount for this example to consider three "meta-values": "before 1", "between 1 and 6", and "after 6". With this kind of technique, the preformances of the causal time approach would be still dependent on the number of tick counters, but not on the values of the constants compared to them.

In Petri nets, there are also other techniques trying to provide a solution to the state space explosion problem. They are typically based on the independence of some actions, often relying on the partial order view of concurrent computation. Based on such a view, the entire state space of a system may be represented implicitly, using an acyclic net in order to represent system actions and local states (see MacMillan's finite prefixes of Petri net unfoldings [14,6]). Such techniques are so far limited to low-level models of Petri nets, but recent researches in this area showed that it is possible to produce high-level prefixes of high-level nets [11]. It is even possible to improve dramatically the efficiency of this analysis by defining an equivalence between markings, which gathers

many states in the generated prefixes. This amounts to abstract data from the Petri net when it has no influence on the execution. For instance with our railroad example, place Time would appear in the prefix only when the values it holds lead to a new branch in the execution of the Petri net. Similarly, most occurrences of the transition *tick* would not be present in the prefix.

This kind of new developments will certainly soon lead to alleviate the state space explosion problem presented above. In such a case, it would not only solve this problem, but it would also increase again the performances already measured because working on finite prefixes is most of time much more efficient than the exploration of the reachability graph. This gain of performance would of course depend of the degree of concurrency we can introduce in the specification.

7 Final remarks

We presented a new approach to the modelling of time constrained concurrent systems, and developed a case study illustrating how it can be used for verification. It showed that causally timed Petri nets are easy to use (the obtained specification is similar to that given with timed automata), and offers also a quite efficient verification. This paper is the first attempt to use this model for verification, and we are aware of many improvements which could be provided. In particular, we should alleviate the state space explosion problem and the sensitivity to the constants clocks values are compared to. However, even without these optimisations, performances were quite satisfactory, what is very encouraging for the future.

We already plan further investigations in this way. On the practical side, we would like to make more case studies, in order to better appreciate the kind of problems that causal time can address efficiently. On the theoretical side, we wish to give a characterisation of a class of timed automata that could be translated into Petri nets with causal time. We think that this class may be quite wide and that it will be possible to establish a bisimilarity relation between timed automata and their translation.

A very important point in this paper is that we showed that it was possible to use successfully untimed Petri nets for the modelling of systems incorporating time constraints. Usually, various Petri net extensions were used for this purpose, where time was associated to net components like places, transitions, arcs or tokens, under the form of durations or dates. For one of these extensions, time Petri nets, it was proposed to compute branching processes including tick transitions [3]. Contrasting with this approach, we provide this kind of representation of time at the level of modelling and not only as a interpretation of another notion of time for the purpose of verification.

We plan to provide case studies comparing causal time with tools based on extended Petri net models. However, we would like to use a specification which allows for concurrency (which is not the case in this paper) because sequential

systems are often the worst case for many Petri net tools (in particular for those relying on the partial order execution semantics).

Finally, we hope that tools will be developed in order to support the needs of the causal time approach. In particular, we discovered that PEP was unable to generate low-level nets from our high-level specification because of their size. Some work is already in progress in order to solve this problem. Another possibility would be to generate prefixes directly from high-level nets, as proposed in [11].

References

[1] R. Alur and D. Dill. *A theory of timed automata.* Theoretical Computer Science, 126(2). Elsevier, 1994.

[2] E. Best, W. Frączak, R. P. Hopkins, H. Klaudel and E. Pelz. *M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages.* Acta Informatica, 35. Springer, 1998.

[3] B. Bieber and H. Fleishhack. *Model checking of timed Petri nets based on partial order semantics.* CONCUR'99, LNCS 1664. Springer, 1999.

[4] A. Cerone and A. Maggiolo-Schettini. *Time-based expressivity of time Petri nets for system specification.* Theoretical Computer Science, 216. Elsevier, 1999.

[5] R. Durchholz. *Causality, time, and deadlines.* Data & Knowledge Engineering, 6. North-Holland, 1991.

[6] J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming, 23. Elsevier, 1994.

[7] H. J. Genrich, K. Lautenbach and P. S. Thiagarajan. *Elements of General Net Theory.* Net Theory and Applications, Advanced Course on General Net Theory of Processes and Systems, LNCS 84. Springer, 1980.

[8] B. Grahlmann. *The PEP Tool.* Computer Aided Verification, LNCS 1254. Springer, 1997.

[9] T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine. *Symbolic model checking for real-time systems.* LICS'92. IEEE Computer Society, 1992.

[10] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1 of EATCS Monographs on TCS, Springer, 1992.

[11] V. Khomenko, M. Koutny and W. Vogler. *Canonical prefixes of Petri net unfoldings.* CAV'02, LNCS. Springer, 2002 (to appear).

[12] H. Klaudel and F. Pommereau. *Asynchronous links in the PBC and M-nets.* ASIAN'99, LNCS 1742. Springer, 1999.

[13] K. G. Larsen, P. Pettersson and W. Yi. *UPPAAL in a nutshell.* International Journal on Software Tools and Technology Transfer, 1(1-2). Springer, 1997.

[14] K. MacMillan. *A technique of state space search based on unfoldings.* Formal Methods in System Design, 6. Kluwer Academic Publishers, 1995

[15] M. M"akel"a. *MARIA: modular reachability analyser for algebraic system nets.* Online manual, `http://www.tcs.hut.fi/maria`, 1999.

[16] R. Milner. *Calculi for synchrony and asynchrony.* Theoretical Computer Science, 25. Elsevier, 1983.

[17] F. Pommereau. *Modèles composables et conurrents pour le temps-réel.* Ph.D. Thesis. University Paris 12, 2002.

[18] G. Richter. *Counting interfaces for discrete time modeling.* Technical report 26, GMD. Sept. 1998.

[19] P. Snyder. *tmpfs: a virtual memory file system.* White Papers, Sun Microsystems Inc.

[20] S. Yovine. *Kronos: A verification tool for real-time systems.* International Journal of Software Tools for Technology Transfer, 1(1/2). Springer, 1997.