Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

The Verification of Real Time Systems using the TINA Tool ⋆

Pedro M. Gonzalez del Foyo * **Jose Reinaldo Silva** **

* *Dept. Of Mechatronics, University of São Paulo, Av. Prof. Mello Moraes 2231, Cidade Universitária, São Paulo, Brazil (Tel: 55-11-30919848; e-mail: pedro.foyo@poli.usp.br).*
** *Dept. Of Mechatronics, University of São Paulo, Av. Prof. Mello Moraes 2231, Cidade Universitária, São Paulo, Brazil (Tel: 55-11-30915688; e-mail: reinaldo@usp.br).*

Abstract: In this paper, we propose a method for building a Timed Transition Graph (TTG) that uses a single clock from the state class graph of a bounded time Petri Net (TPN). To build this TTG a special state class construction - available in **Tina** - is used. This structure can be used to calculate "quantitative" temporal properties for the worst case scenario. It is possible to check efficiently several properties over the same TTG if no design modifications are made. Such properties are represented by a framework proposed in the literature were TCTL properties are defined on TPN.

Keywords: Real-time systems, Verification, Time Petri Nets

1. INTRODUCTION

The verification of the real time aspects of high-integrity computer-based system can usually be seen as a three stage process which aims to assert the feasibility of the requeriments, its design and implementation (Burns 2003). In this paper we are interested in the first two stages. To examine the requirements and design, it is necessary to build a model that represents the temporal behavior of the proposed system. The model obtained is then checked for feasibility and safety.

In order to express temporal requirements and design properties, a modeling schema must represent the delay and deadline concepts (Burns & Wellings 2001), besides being able to express the inherent concurrency characteristic of real time systems.

Model checking has turned out to be an useful technique for verifying temporal properties of finite-state systems (Larsen et al. 1995). Particularly, real-time model checking has been mostly studied and developed in the framework of Alur and Dill's Timed Automata (TA), that is, using automata extended with clocks that progress synchronously with the time. There now exists a large amount of theoretical knowledge and practical experience for this class of system. However, it is agreed that their main drawback is the complexity blowup induced by timing constraints - most verification problems are at least PSPACE-hard for TA.

Since the work of Alur et al. (1993) a wide variety of model checkers has been developed based on the TA formalism. Some of this modeling formalisms are supported by analysis tools, such as **UPPAAL** (Larsen et al. 1997) and **Kronos** (Daws et al. 1995). Conventional timed model checkers are usually based in two algorithms: the one known as forward, which can work on-the-fly, and another called backwards. This last one, although does not fit for on-the-fly composition, but provides the basis for verification of the complete TCTL logic (Alur et al. 1993).

Petri nets have been used successfully in formal description and analysis of concurrent systems in their original formulation but do not deal explicitly and quantitatively with time, what makes them unsuitable to the modeling and specification of strict real-time systems. Therefore, Petri nets have been augmented in several ways to allow the description of time dependent phenomena. The two main time extensions are Time Petri Nets (TPN) (Merlin & Faber 1976) and Timed Petri Nets (Ramchandani 1974).

While a transition can be fired within a given interval for TPN, in Timed Petri Nets, transitions are fired as soon as possible. There are also numerous ways for representing time. TPN are mainly divided in P-TPN, A-TPN and T-TPN whether a time interval is relative to places (P-TPN), arcs (A-TPN) or transitions (T-TPN). (David & Alla 2005) has an interesting survey on that subject.

For real-time systems, TPN is an adequate choice because it can represent delays, deadlines and the inherent concurrency of that systems. Time is represented in terms of time intervals, capturing the non deterministic nature of processes such as code execution durations, timeouts and communication delays. It can also be used to model different events and situations that appear in RTS, e.g., concurrence, asynchronous transfer of control, events, periodic and aperiodic activities.

The proposal of a verification framework in this paper is related to those presented in (Virbitskaite & Pokozy 1999) and (Lime & Roux 2006). In (Virbitskaite & Pokozy

⋆ This work was partially supported by CAPES, and CNPq, the national research agences.

1999) a model-checking algorithm for TCTL over TPN was proposed based on the *region graph approach*. The quantitative part is checked adding a special transition to check the time of the TCTL formula. Therefore it is necessary to check an extra transition per formula. The complexity of such algorithm is similar to the one proposed in (Alur et al. 1993). A partial order reduction method is used to decrease the state-space size in order to improve the complexity of the model checking algorithm. The use of TPN as a model formalism instead of TA allowed the to represent concurrent systems in a more natural way.

Lime & Roux (2006) also used TPN to model the system behavior. They used the *state class approach* to build a TA that preserves the behavior of TPN using less clock variables as possible. The system is then verified using **UPPAAL**. They also presented a definition of TCTL for TPN. The difference from the definitions presented in (Alur et al. 1993) is that the atomic propositions usually associated with states are properties of marking.

Our proposal also use the TPN to model the system behavior and then use the state class approach to build the TCTL structure where the verification must be done. We claim that labeling algorithms could be executed over such structure with better complexity results.

This paper is organized as follows: Section 2 gives a formal semantics for TPN in terms of timed transition systems and present the state class graph definitions used in **TINA** (Berthomieu et al. 2004). Then, Section 3, a proposal is presented to extend this construction that allows to build the state class graph as a TTG with one clock. In Section 4, we use a framework for checking TCTL properties on the TPN presented in Lime & Roux (2006) and based on our structure. The verification method is shown throughout an example also in Section 4. Finally, Section 5 brings some analysis and conclusions about the Petri Net based method for model checking.

2. TIME PETRI NETS AND STATE CLASSES

Merlin's Time Petri Net (TPN) extend Petri Nets with temporal intervals associated with transitions, specifying firing delays range. Time Petri Nets was defined in Merlin & Faber (1976).

This section gives a formal definition of TPN and for states in TPN according to Berthomieu & Diaz (1991). The notion of state and state class will be useful to discuss behavioral analysis for TPN.

A Time Petri Net is a tuple (P, T, B, F, M_o, SIM) where:

- P is a finite non-empty set of places p_i;
- T is a finite non-empty set of transitions t_i;
- B is the backward incidence function
  $$B : P \times T \longrightarrow N;$$
  where N is the set of nonnegative integers;
- F is the forward incidence function
  $$F : T \times P \longrightarrow N;$$
- M_o is the initial marking function
  $$M_o : P \longrightarrow N$$
- SIM is a mapping called static interval
  $$SIM : T \longrightarrow Q^* \times (Q^* \cup \infty)$$
  where Q^* is the set of positive rational numbers.

The static interval is composed by two positive rational numbers (α, β), where α represents the earlier firing time (EFT) and β the latest firing time (LFT). Assuming that a transition t became enabled at time τ , then t cannot fire before $(\tau + \alpha)$ and no later than $(\tau + \beta)$ unless disabled by firing some other transition.

In TPN, the enabling condition of a transition is the same as in Petri Nets. According to the definition of TPN the "enabling condition" will be:

$$(\forall p)(M(p) \geq B(t_i, p)); \tag{1}$$

Some transitions may be enabled by marking M, but not all of them may be allowed to fire due to the firing constraints of transitions (EFT′s and LFT′s). Thus, the "firing condition" depends on two conditions:

(1) the transition is enabled, formally expressed by Eq. (1)
(2) express the fact that enabled transitions may not fire before its EFT and must fire before or at its LFT unless another transition fires before, modifying the marking M in a way that the transition is no longer enabled.

According to the second condition, a transition t_i enabled by M at absolute time τ could be fired at the firing time θ, iff θ is not smaller than the EFT of transition t_i and not greater than the smallest of the LFT's of all the transitions enabled by marking M, that is: EFT of $t_i \leq \theta \leq \min\{$ LFT of $t_k\}$ where k ranges over the set of all transitions enabled by M. If transition t_i fires, it leads the system to another state, at time $\tau + \theta$.

The state of the system can be defined as a pair $S = (M, I)$ where:

- M is a marking
- I is a firing interval set which is a vector of possible firing times. The number of entries of this vector is given by the number of transitions enabled by marking M.

The single behavior "transition t_i is firable from state S at time θ and its firing leads to a state S'" will be denoted by $S \xrightarrow{(t_i,\theta)} S'$.

The firing rule permits the computation of states and a reachability relation among them. The set of states that are reachable from the initial state, or the set of firing schedules feasible from the initial state, characterizes the behavior of the TPN, the same way that the set of reachable markings characterize the behavior of a Petri net.

In general, using this set of states for analysis purpose is not feasible, once this set could be infinite. That is why Berthomieu & Menasche (1983) introduce an enumerative approach for analyzing TPN using what they call state classes.

A state class, denoted by C will be defined by a pair (M, D) where M is a marking and D is a firing domain. The firing domain D will finitely represent the infinite number of firing times possible from marking M, through the set of solutions of a system of inequalities that capture the global timed behavior of the TPN.

The number of state classes will be finite if the underlying Petri net is bounded. Proof of boundeness for TPN has been shown to be undecidable but a number of sufficient conditions could be stated (Berthomieu & Diaz 1991). **TINA** (TIme petri Net Analyzer) proposes several state class space constructions, preserving some properties expressed either in linear time temporal logics (such as LTL) or in branching time temporal logics (such as CTL).

In this paper, we are concerned with a class of properties of great practical interest for which no dedicated construction has been proposed by the authors of **TINA**. That "quantitative" temporal properties may be expressed, for instance, in TCTL logic. However, **TINA** has other constructions such as atomic state classes (ASCG), which preserve the branching properties just as CTL or CTL*. Any graph of classes bisimilar with the state graph of the net preserves all its branching properties, but, time information is omitted on this graph in **TINA** (Berthomieu et al. 2004). This will be addressed in the next section.

3. THE TIMED TRANSITION GRAPH

In this paper we are proposing a modification in the Timed Transition Graph in order to simplify the model checking algorithm. The TCTL logic can be verified over such structure in polynomial time, in a similar way that the labeling algorithm verify TCTL over the structure builded using the region graph approach (Alur et al. 1993).

Our Timed Transition Graph is a tuple $G = (S, \mu, s_0, A, E, It)$, where

- S is a finite set of nodes;
- $\mu : S \rightarrow P$ is a labeling function assigning to each node the set of atomic propositions that are true in this node;
- $s_0 \in S$ is the start node;
- A is a set of actions or events.
- $E \subseteq S \times A \times S$ is a set of edges which has a bijection with the set A;
- $It : E \longrightarrow Q^* \times (Q^* \cup \infty)$ is the minimum and maximum time to change the current state to the next through the occurrence of $a \in A$; (where Q^* is the set of positive rational numbers)

Consider the example of Figure 1 that depicts a TPN model of a real time system composed by two processes: one for control and one for supervision. In this example the two processes share an A/D D/A card. The control process execute a control loop, composed for three routines: reading (read the variable to control using and A/D channel), computing (executes the control algorithm to determine the control action) and action generation (write the control action to the correspondent output channel in the card). This routines and its durations are represented respectively for transitions $t3$, $t4$ and $t6$. The supervision task is composed by two routines: reading (read the variable to supervise) and recording (create a history of the variable for supervision purposes). Transitions $t10$ and $t11$ represents those routines. Place $p9$ represents the resource shared by this two processes, in this case, the A/D D/A card.

The time intervals for $t3, t4, t6, t10$ and $t11$ contain the Best Case Execution Time (BCET) and the Worst Case Execution Time (WCET) of the code, respectively, while time intervals in transitions $t0$ and $t7$ represent the period of each task. The state of the system is identified in a TPN

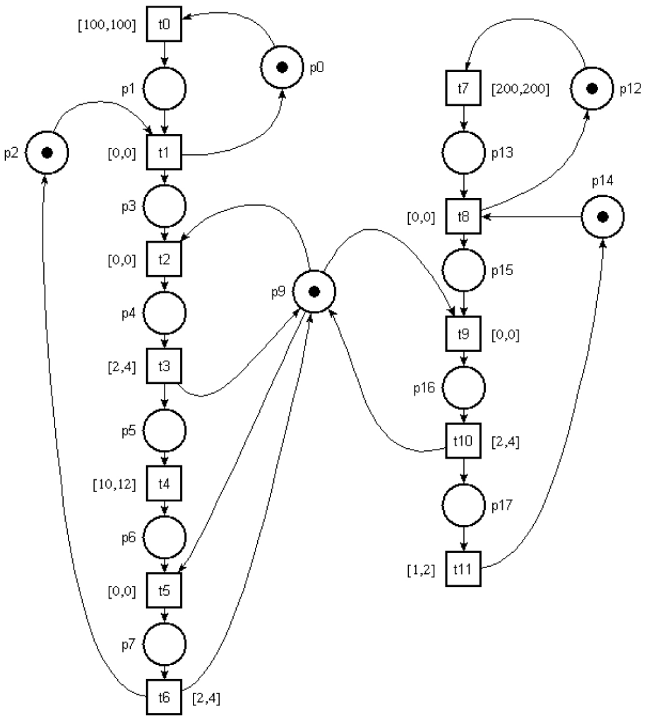

Fig. 1. Example of real time system.

as $S = (M, I)$. Let τ be the time when the system achieve the S state. Consider that exists at least one transition enabled in S, then exists a time interval in which S could leave to state S' through the occurrence of that transition. These could be represented as:

$$S \xrightarrow{t_1[\theta_1,\theta_2]} S'$$

where $\tau + \theta_1$ is the minimum time in which the system could achieve state S' coming from S through the occurrence of transition t_1 and $\tau + \theta_2$ is the maximum time in which the system could achieve state S' coming from S through the occurrence of transition t_1.

Consider that there is more than one transition enabled at S, so:

$$S \xrightarrow{t_2[\theta_3,\theta_4]} S''$$

of course, $\theta_1 \leq \theta_4$ and $\theta_3 \leq \theta_2$. Transitions t_1 and t_2 are present in I, t_1 is present in I'' and t_2 is present in I'.

If we have the reachability graph, the state classes and the static time interval for each transition in the net we could calculate the time interval It for each edge in the TTG. States S in the TTG correspond to the state classes of the TPN. The labeling function μ is constructed assigning to each state $s \in S$ the marked places in its equivalent state class.

Let be $C_1 \xrightarrow{t_i[l,u]} C_2$ a state class transition where $C_1 = (M_1, D_1)$ and $C_2 = (M_2, D_2)$, ti is an enabled transition at M_1 and $[l, u]$ is a fair time interval of global time for the firing of transition t_i in state class C_1 leading to the

state class C_2. Of course, that interval must be consistent with the firing domain of both classes.

The consistency checking for the firing domains is given by the solution of the following set of inequations:

$$\begin{gathered} a_i \leq \theta_i \leq b_i,\ t_i \text{ is the fired transition;} \\ a_j \leq \theta_j \leq b_j,\ \forall j \,|\, t_j \in \text{enb}(M_1) \cap \text{enb}(M_2); \\ a'_k \leq \theta_k + t \leq b'_k,\ \forall k \,|\, t_k \in \text{enb}(M_1) \cap \text{enb}(M_2); \end{gathered}$$

where a_i, b_i represents the static interval for transition t_i; a_j, b_j belongs to the firing domain D_1 and a'_k, b'_k belongs to the firing domain D_2. The sets $enb(M_1)$ and $enb(M_2)$ contain the transitions enabled by marking M_1 and M_2 respectively. Then, the solution of variable t is the fair time interval of global time for the firing of transition t_i. The algorithm to solve that follows from the set of inequations:

$$a'_j - a_j \leq t \leq b'_j - b_j,\ \forall j | t_j \in (\text{enb}(M_1) \cap \text{enb}(M_2)) \cup \{t_i\}$$

So, the interval $It : [l, u]$ for transition $C_1 \xrightarrow{t_i[l,u]} C_2$ will be:

$$\begin{aligned} l &= \max(0, \max(a'_j - a_j)) \\ u &= \max(0, \min(b'_j - b_j)) \end{aligned}$$

Even when the complexity of the state class generation algorithm has not been established by its creators, we can say that the algorithm is proportional to the size of the reachability graph of the TPN. If we consider the input of the algorithm a k-bounded TPN the complexity will be $O(|P|^k \times |T|)$ which is an exponential algorithm.

Region graph and state class approaches are both timed-abstraction bisimulations from the modeled systems. The state class approach produces a better state space partition since it don't uses transitions to model the time progress. According to Yovine (1996) this yields to a coarsest partition of the state space. The algorithm to build the TTG from the state class graph is polynomial in the number of state classes but since the computation of the state classes is ttime exponential, the construction of the TTG remains exponential.

The example shown in Figure 1 was analyzed using **TINA** and 41 state classes were obtained. The results show that the TPN is bounded and there is no dead class or dead transition. Thus, the time interval was calculated for each transition obtaining the TTG in Figure 2.

4. THE VERIFICATION APPROACH

In this paper we use Timed Computation Tree Logic (TCTL) that extend CTL model-checking (E. M. Clarke & Sistla 1986) to the analysis of real time systems whose correctness depend on the magnitudes of the timing delays. To deal with specifications, the syntax of CTL was extended to allow quantitative temporal operators in Alur et al. (1993). TCTL is a branching temporal logic for the specification of dense-time systems.

In (Lime & Roux 2006) was presented a definition of TCTL for TPN. The difference to the definition presented in (Alur et al. 1993) is that atomic propositions usually associated with states are properties of markings. We

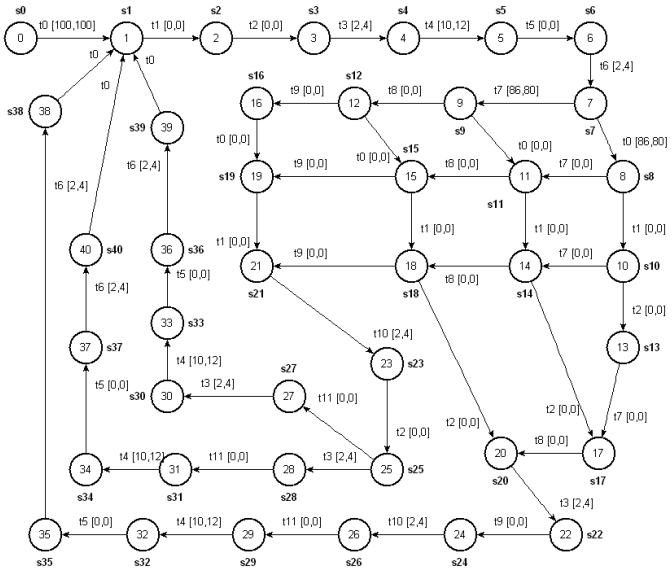

Fig. 2. Timed Transition Graph for the example shown in Figure 1.

use that definition for TCTL model checking in order to construct formulas that represent properties to be checked.

Usually, several properties must be checked to guarantee the adequate behavior of the system. As stated in (Alur et al. 1993), the labeling algorithm runs in polynomial time with the size of the graph, once the TCTL structure has been computed using the region graph approach. Our claim is that if we do the construction of the entire state space once, with all time increments over an unique clock variable, we can then verify several properties without the necessity to recompute the state space - which is the exponential part of the verification method. Thus, we run once an exponential algorithm and several times a polynomial one.

Since we are using an unique clock variable, there is no need to specify the clock variable in the TCTL formula. On the other hand, real time system processes could be executed in different devices (such as different PLC´s), and for the sake of verification purposes only certain events contribute to time system execution, what explains why a clock projection must be defined for each formula to be checked.

The clock for the formula could be defined as a set of edges, i.e. $clk \subseteq A$, which affects the time of the process being verified. Going back to the example of Figure 1, consider that we need to check whether the control action is generated in at least 20 time units since the beginning of the process.

The formula to be checked representing this property is,

$$\forall(p4 \vee p5 \vee p6 \vee p7) U_{\leq 20}(p2); \tag{2}$$

The clock for this formula is the set of significant events for the control process.

$$clk = (t0, t1, t2, t3, t4, t5, t6, t9, t10)$$

$t9$ and $t10$ are in clk due to the resource sharing. If no clock is specified for a formula then the default is used

which is all transitions count to compute the time for that formula.

Figure 3 shows the TTG where states are marked light gray when subformula $(p4 \vee p5 \vee p6 \vee p7)$ evaluate to *true* and marked dark gray when subformula $p2$ evaluate to *true*. The clock is updated related to the process control only. The time is obtained as the sum of the maximum time from the interval where the subformula $(p4 \vee p5 \vee p6 \vee p7)$ was evaluated to *true*. Note that $t11$ is reset to [0,0] because $t11 \notin clk$ The test fails because there is at

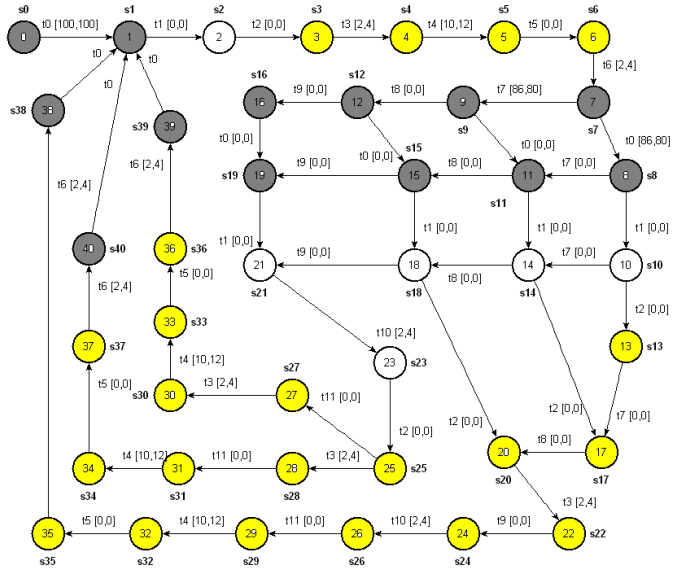

Fig. 3. Timed Transition Graph evaluated to formula 2.

least one computation path where $(p4 \vee p5 \vee p6 \vee p7)$ could delay 24 time units.

$$S_{13} \xrightarrow{(t_7,0)} S_{17} \xrightarrow{(t_8,0)} S_{20} \xrightarrow{(t_3,4)} S_{22} \xrightarrow{(t_9,0)} S_{24} \xrightarrow{(t_{10},4)} S_{26} \xrightarrow{(t_{11},0)} S_{29} \xrightarrow{(t_4,12)} S_{32} \xrightarrow{(t_5,0)} S_{35} \xrightarrow{(t_6,4)} S_{38}$$

Therefore, modifications must be introduced even in time constraints or in the system itself in order to converge to a solution. The test shows that the constraint must be relaxed to 24 time units or the routines represented by $t3, t4, t6$ and $t10$ must be executed faster.

If the system design don't change, other verifications can be done without constructing another TTG. The labeling algorithm evaluates each subformula on each state and then check whether the general formula is valid or not. At most, depending on the property, a new clock for the formula must be established.

Consider that we want to check if the supervision task could delay more than 10 time units to capture the measured variable since it was called. The formula to be checked representing such property is,

$$\exists(p13 \vee p15 \vee p16) U_{>10}(p17); \qquad (3)$$

The clock for this formula is the set of significant events for the supervision task.

$$clk = (t2, t3, t7, t8, t9, t10, t11)$$

$t2$ and $t3$ are in clk due to the resource sharing. Notice that there is no need to recompute the state class graph since the state class graph have the maximum and minimum time increment over the global clock. Thus, we only need to specify the transitions which time counts for the computation of that formula. The verification of formula 3 fails since the maximum time spending in the supervision process until the measured variable is capture is 8 time units as we can see in Figure 1.

Another method that can be used to verify RTS with **TINA** consists in add to a TPN model a place (or places) that represent the property that must be checked. This approach known as "TPN + observer" was proposed in Toussaint et al. (1997). The problem with that approach is that the "observer net" could be as big as the net if the property to be verified involves markings, and thus, the number of classes may be squared. Furthermore, each property requires an specific observer, and thus a new computation of the state class graph (Lime & Roux 2006).

Using the reachability analysis of **TINA**, other verifications could be done. Consider again the example of Figure 1 but this time, in the supervisor routine, calculations are made to determine whether or not changes in the control algorithm must be done. In case of a change, the new setpoint value must be sent to the control routine. Figure 4 shows the TPN model for this problem. Now we

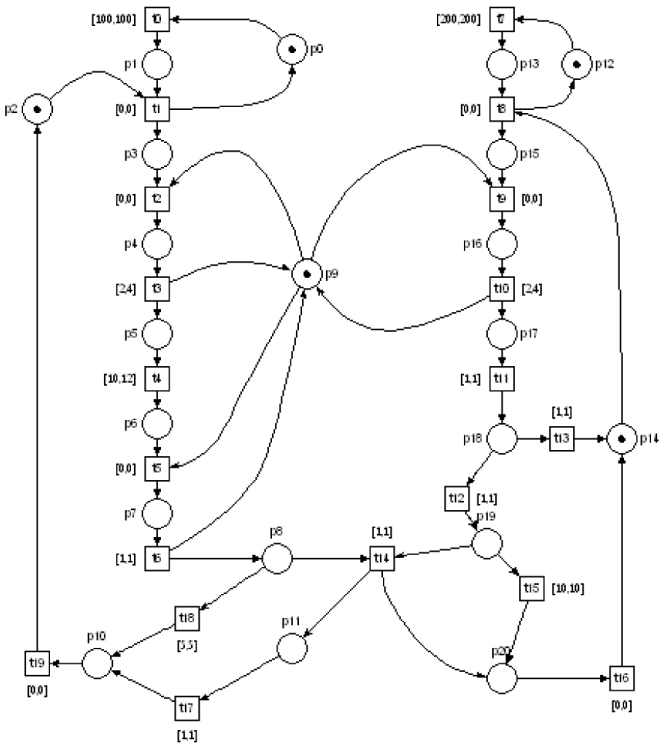

Fig. 4. TPN model of the example with new supervision task.

are interested in determining the *timeout* values that allow no missed informations between the processes. In Figure 4, transition $t12$ represents the fact that the setpoint must be changed. The occurrence of $t17$ indicates that the communication was successfully completed, $t15$ and $t18$ are the *timeout* transitions.

We began with values 10 and 5 for $t15$ and $t18$ respectively. The verification problem was reduced to find the *timeout* values that turns those transitions dead. This values must be high enough to allow no missing packages and low enough to guaranty the scalability of both processes. Some iterations where done to achieve the values shown in Figure 4.

5. CONCLUSIONS

In this paper, we proposed a method for building a structure in which the TCTL can be verified from the state class graph of a TPN. Such structure is based on the results of the atomic state classes (ASCG), which preserves the branching properties like CTL or CTL*, supported by **TINA**. The time interval attached to each edge of the automaton allow us to calculate quantitative temporal properties like those expressed in TCTL. Such properties have proved to be useful in practical applications.

Even when Lime & Roux (2006) uses the state class approach in their verification framework, the reduction in complexity achieved in this work is due to the use of symbolic and on-the-fly techniques applied in **UPPAAL** and because of that a new computation is needed for each formula to be checked. There is also the restriction in the subset of TCTL formulas that can be verified by **UPPAAL**.

Since Virbitskaite & Pokozy (1999) uses the region graph approach in their verification framework the complexity is quite similar of that in (Alur et al. 1993). The difference is the use of TPN to model the system behavior. In general, a recomputation of the region graph is needed for each formula to be checked.

The better runtime results using our framework are achieved thanks to the use of the state class approach, since the number of state classes are lesser than the number of regions in the region graph approach. TCTL formulas show to be decidable over the structure proposed here. As a tree structure, the labeling algorithm runs in time $O(|\phi|\times|S|+|E|)$ where $|\phi|$ is the length of the formula. Thus, all verifications can be done over the same TTG each one in polynomial time.

On the other hand, TCTL formulas proved to be easier to construct in TPN than in TA since the number of states in PN (represented as markings) are lesser (most of the time significantly less) than in automata.

We show that some properties could be verified directly from the results obtained by **TINA** without the necessity of building the TTG or running the labeling algorithm. Of course, those techniques are applicable only in some cases.

ACKNOWLEDGEMENTS

Partially supported by CAPES, and CNPq, the national research agencies.

REFERENCES

Alur, R., Courcoubetis, C. & Dill, D. L. (1993). Model-checking in dense real-time, *Information and Computation* **104**(1): 2–34.

Berthomieu, B. & Diaz, M. (1991). Modelling and verification of time dependent systems using time petri nets, *IEEE Transaction on Software Engineering* **17**(3): 259–273.

Berthomieu, B. & Menasche, M. (1983). An enumerative approach for analyzing time Petri nets, *in* R. E. A. Mason (ed.), *Information Processing: proceedings of the IFIP congress 1983*, Vol. 9, Elsevier Science Publishers, Amsterdam, pp. 41–46.

Berthomieu, B., Ribet, P. O. & Vernadat, F. (2004). The tool tina - construction of abstract state spaces for petri nets and time petri nets, *Int. J. Prod. res.* **42**(14): 2741–2756.

Burns, A. (2003). How to verify a safe real-time system: The application of model checking and timed automata to the production cell case study, *Real-Time Syst.* **24**(2): 135–151.

Burns, A. & Wellings, A. J. (2001). *Real-Time systems and programming languages (3rd ed.)*, third edition edn, Addison-Wesley, Boston MA.

David, R. & Alla, H. (2005). *Discrete, Continuous and Hybrid Petri Nets*, Springer-Verlag.

Daws, C., Olivero, A., Tripakis, S. & Yovine, S. (1995). The tool KRONOS, *Hybrid Systems III: Verification and Control*, Vol. 1066, Springer, Rutgers University, New Brunswick, NJ, USA, pp. 208–219.

E. M. Clarke, E. A. E. & Sistla, A. P. (1986). Automatic verification of finite state concurrent systems using temporal logic specifications., *ACM transactions on Programming Languages and Systems* **8**(2): 244–263.

Larsen, K. G., Pettersson, P. & Yi, W. (1995). Compositional and symbolic model-checking of real-time systems, *IEEE Real-Time Systems Symposium*, pp. 76–89.

Larsen, K. G., Pettersson, P. & Yi, W. (1997). UPPAAL in a Nutshell, *Int. Journal on Software Tools for Technology Transfer* **1**(1–2): 134–152.

Lime, D. & Roux, O. H. (2006). Model checking of time petri nets using the state class timed automaton, *Discrete Event Dyn Syst* **16**: 179–206.

Merlin, P. & Faber, D. (1976). Recoverability of communication protocols–implications of a theoretical study, *Communications, IEEE Transactions on [legacy, pre-1988]* **24**(9): 1036–1043.

Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by timed petri nets, *Technical report*, Massachusetts Institute of Technology, Cambridge, MA, USA.

Toussaint, J., Simonot-Lion, F. & Thomesse, J.-P. (1997). Time constraints verification methods based on time petri nets, *Proc. 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, IEEE Computer Society, Washington, DC, USA, p. 262.

Virbitskaite, I. & Pokozy, E. (1999). A partial order method for the verification of time petri nets, *in* G. Ciobanu & G. Paun (eds), *FCT*, Vol. 1684, Springer Verlag, pp. 547–558.

Yovine, S. (1996). Model checking timed automata, *European Educational Forum: School on Embedded Systems*, pp. 114–152.