Preliminary Results of Global Time Petri Net Analysis Applied to Embedded Software Prototyping

Leticia Mara Peres[1], **Eduardo Todt**[1], **Luis Allan Kunzle**[1]

[1]Computer Science Department, UFPR, Curitiba, Brazil

`lmperes@inf.ufpr.br,todt@inf.ufpr.br,kunzle@inf.ufpr.br`

Abstract. *We propose in this paper an approach based on Time Petri Nets (TPN) to analyze time aspects and to schedule embedded software during the prototyping activities. Firstly, a TPN that represents the interaction between system tasks is modeled using Petri net design patterns. Secondly, a class graph is built with the Global Time Method (GTM) and interval algebra operations. Finally, firing sequences on GTM graph is found. The firing sequences represent behavior itineraries of software, and their global time based on cyclic scheduling, fixed priority and earliest deadline first scheduling policies. The example shows simple and precise time analysis of behavior itineraries which use a powerful theory based on interval algebra, different of conventional models.*

1. Introduction

Real time embedded systems are constrained about functionalities and resources. In these systems, time constraints are so important as functional constraints.

Petri nets (PN) and their algebraic properties are used to model and to analyze systems involving parallelism, concurrency and synchronization. Several extensions of the basic formalism have been proposed to increase their power of modeling. In this work we are interested in Time Petri nets (TPN), where quantitative time restrictions can be considered [1]. The classic technique of analysis is based in an enumerative method to generate the reachable state space of a TPN [2]. This method finds the relative time interval which the system remains in each state, being therefore efficient to verify net properties and to analyze time constraints relative to a given class. This method itself does not present global time information of the net behavior since its initial marking. In order to obtain more accurate results, Dill [3] describes data structures Difference Bounds Matrix (DBM) and proposes a method for using delay information in state graph verification of finite state concurrent systems. These structures have been used in association with model checking and timed automata approaches [4], [5], [6]. We propose a global time alternative method based on interval algebra.

TPN are used in different applications of embedded system verification, scheduling and synthesis methods. Cortes et al. [4] define a method of embedded software modeling and verification using PRES+, a type of TPN. Lime and Roux [5] propose SETPN to model real-time systems, especially embedded systems. They present a set of PN design patterns to model tasks with preemptive scheduling. In [6], Lime and Roux use their scheduling TPN design patterns of [5] to model tasks and deal with Fixed Priority and Earliest Deadline First policies.

We present in this paper the Global Time Method (GTM) which is an approach to generate a new type of class graph for TPN. This method finds relative and global time information about the current state of the net. The global time information is correct for both limits of time interval of any class, even when the net represents concurrency among events [7]. The remainder of this paper is organized as follows. The Section 2 defines basic concepts of interval operations, TPN and class graph. Section 3 establishes the class graph based on GTM, and presents an example of application. Section 4 presents an application of GTM on the context of embedded software scheduling, while Section 5 concludes the article.

2. Basic Concepts

Let two rational numbers, a and b, such that $a \leq b$. We denote $[a, b]$ as the set $\{x \in R : a \leq x \leq b\}$, defined as closed interval from a to b. An interval $[c, d]$ is denoted as not proper when $d < c$, with c and d rationals. Given the intervals $[a, b]$ and $[c, d]$, proper or not proper, we define the following operations: $[a, b]+[c, d] = [a+c, b+d]$; $[a, b]-[c, d] = [max\{0, a-d\}, max\{0, b-c\}]$; $[a, b] \ominus [c, d] = [max\{0, a-c\}, max\{0, b-d\}]$. The interval subtraction $\ominus$ is used to adjust time coefficients which can be represented by not proper time intervals.

A Time Petri net (TPN) is a tuple $TPN = (P, T, Pre, Post, M_0, I)$ [8], where:

- P is a finite set of places, $p \in P$,
- T is a finite set of transitions, $t \in T$,
- Pre is an input application such as $Pre : (P \times T) \rightarrow \mathbb{N}$,
- $Post$ is an output application such as $Post : (P \times T) \rightarrow \mathbb{N}$,
- M_0 is the initial marking, and
- $I : T \rightarrow (\mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$ such as $(t, e(t)) \in I$ and $e(t) = [a, b]$, where a and b are positive rational numbers.

A marking is an assignment of marks to places of the net. The marking of a place $p \in P$ is denoted $M(p)$. A TPN has one static time interval $e(t) = [a, b]$, with $a \leq b$, associated to each transition $t \in T$. The limits a and b represent, respectively, the earliest and the latest possible firing time of transition t, counted from the instant when t is enabled ($\forall p, M(p) \geq Pre(p, t)$). When t fires in a marking M_{k-1} a new marking of the net is given by marking $M_k = M_{k-1} + Post(t) - Pre(t)$ ($c_{k-1}[t_f > c_k$).

A state class $c_k = (M_k, W_k)$, where M_k is the current marking of the TPN obtained by the firing of a transition t. W_k is the set of time information for this class. A transition t, enabled in a class c_k, is a *persistent* transition in c_k if t was enabled in a class c_{k-1} and t did not fired in c_{k-1}. A transition t, enabled in c_k, is *newly enabled* if t was not enabled in class c_{k-1} or, the firing of t originated the class c_k and it was re-enabled in c_k.

The state class graph is a directed graph where each node is a state class and each arch is labeled with one transition. The root node of class graph is the start class c_k with level $k = 0$ and has the initial marking M_0 of TPN.

3. Global time method (GTM)

The information set W_k of class c_k has two types of time information: relative and global. The relative time information is the accumulated time for each transition since its enabling in class c_k. Global time information refers to the accumulated time since the initial marking (class c_0).

Let $r_k(t_i)$ a relative time interval of a transition t_i calculated in a class c_k such that $c_{k-1}[t_f > c_k$, and defined as:

$$r_k(t_i) = \begin{cases} e(t_i) & \text{if } t_i \text{ is \textit{newly enabled} in } c_k \\ r_{k-1}(t_i) - r_{k-1}(t_f) & \text{if } t_i \text{ is \textit{persistent} in } c_k \end{cases}$$

The relative time interval $r_k(t_i)$ of each enabled transition t_i of a class c_k is used to identify which ones are fireable. A transition t_f with $r_k(t_f) = [a_f, b_f]$ is fireable in c_k if, and only if, t_f is enabled in c_k and there is not another transition t_i with $r_k(t_i) = [a_i, b_i]$ enabled in c_k such that $b_i < a_f$.

The persistence adjustment coefficient $ac_k(t_i)$ of an enabled transition t_i in a class c_k such that $c_{k-1}[t_f > c_k$, is defined as:

$$ac_k(t_i) = \begin{cases} r_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{if } t_i \text{ and } t_f \text{ are both newly enabled in } c_{k-1} \\ ac_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{if } t_i \text{ is persistent and } t_f \text{ is newly enabled both in } c_{k-1} \\ r_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{if } t_i \text{ is newly enabled and } t_f \text{ is persistent both in } c_{k-1} \\ ac_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{if } t_i \text{ and } t_f \text{ are both persistent in } c_{k-1} \end{cases}$$

The persistence adjustment coefficient $ac_k(t_i)$ prevents the increase of imprecision to computing the global time.

The global time interval $g_k(t_i)$ of a fireable transition t_i in a class c_k such that $c_{k-1}[t_f > c_k$ is:

$$g_k(t_i) = \begin{cases} e(t_i) & \text{if } k = 0\text{, i.e. if is the initial class} \\ g_{k-1}(t_f) + r_k(t_i) & \text{if } k \neq 0 \text{ and } t_i \text{ is newly enabled in } c_k \\ g_{k-1}(t_f) + ac_k(t_i) & \text{if } k \neq 0 \text{ and } t_i \text{ is persistent in } c_k \end{cases}$$

The global time interval $g_k(t_i)$ is counted from the initial marking until the firing instant of t_i in class c_k.

Let t_f be the fired transition in a class c_k such that $c_k[t_f > c_{k+1}$. The upper bound of global time interval $g_k(t_f) = [a_f, y]$ of fired transition t_f, must be adjusted by the lowest upper bound of intervals calculated to all t_i in class c_k, where $y = min\{b_i \mid g_k(t_i) = [a_i, b_i], \forall t_i$ fireable in $c_k\}$.

The successive firing of two or more transitions in a TPN from any class c_k to any other class c_{k+n}, also called firing sequence, is represented by $c_k[s > c_{k+n}$. The global time of a firing sequence s of $c_0[s > c_k$ is the resulting of global time interval $g_{k-1}(t_f)$, being t_f the last transition fired to reach class c_k, that is, $c_{k-1}[t_f > c_k$.

The class graph can express the staying time in each class, i.e., how long the system remains in the state represented by the class. The staying time of a net in a certain reachable class c_k is given by: $i_k = [x, y]$, where $x = min\{a_i \mid r_k(t_i) = [a_i, b_i]\}$ and $y = min\{b_i \mid r_k(t_i) = [a_i, b_i]\}$, $\forall t_i$ fireable in c_k.

4. Application

We based the application of GTM on the work of Lime and Roux, 2009 [6]. It defines a special TPN with scheduling layer and, among other things, allows to map each place of the net to a task. We propose to use parts of this layer to model TPN according design patterns of work of Lime and Roux, 2003 [5], associating tasks to transitions and places of the net. Then, we generate the GTM state class graph and analyze firing sequences that satisfies "Earliest Deadline First" and "Fixed Priority" scheduling policies.

Let, according [6]:

- $\tau \in$ *Tasks*, being *Tasks* the set of tasks of the system, where there is no task migration between processors.
- *Sched:Procs* $\mapsto$ {*FP,EDF*} the function that maps a processor to a scheduling policy, being FP "Fixed Priority" and EDF "Earliest Deadline First" ;
- Π*: Tasks* $\mapsto$ *Procs* the function that maps a task to its processor;
- ϖ*: Tasks* $\mapsto \mathbb{N}$, for $Sched(\Pi(\tau)) = FP$, gives the priority of the task on the processor;
- δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$, for $Sched(\Pi(\tau)) = EDF$, gives the deadline interval of the task on relative to its activation time.

In order to map each place of the TPN to a task, we use the function $\gamma : P \mapsto$ Tasks $\cup\{\phi\}$, where ϕ denotes that the place is not mapped to any real task. We, as in [6], assume that for each transition, there is at most one place p such that $p \in Pre(t)$ and $\gamma(p) \neq \phi$. If $\forall p \in Pre(t)$, $\gamma(p) = \phi$, then t is not bound to any real task and we say that it is *part of* ϕ (denoted by $\gamma(t) = \phi$). Otherwise, for each transition t, we say that t *is part of* the task τ, and we denote it $t \in \tau$ if one of its input places is mapped to $\tau : t \in \tau \Leftrightarrow \exists p \in Pre(t)$, s.t. $\gamma(p) = \tau$. So, $\gamma(t)$ is the task s.t. $t \in \tau$.

As in [6], each task τ is thus modeled by a subnet of the TPN composed of places mapped to τ by γ and of transitions with static time, which are parts of τ. As in [6] we assume that at most one instance of each task is active at a given instant, which is expressed by the restriction that at most one place mapped to τ by γ is marked at a given instant. Let $B(\tau)$ be the set of transitions which *start* the task τ and similarly, let $E(\tau)$ be the set of transitions which terminate τ. These two sets are user-defined as part of the modeling phase. After TPN was modeled, we propose generate a GTM state class graph, as presented at section 3. The definitions of scheduling layer reflect on TPN and on each scheduling policy.

The mapping between scheduling policies and GTM graph defines a criterion for path enumeration on GTM state class graph, where the path is the firing sequence, satisfying some scheduling policy. We have established criteria for "Earliest Deadline First" and "Fixed Priority" scheduling policies. For *Sched:Procs* $\mapsto$ {*CE*}, where CE is "Cyclic Executive", the TPN model represents only one task which is typically realized as an infinite loop in *main()*. Because *CE* has not a specific criterion in order to satisfy, this policy is achieved only by modeling TPN and it is not necessary formalize this function in relation to GTM state class graph enumeration.

Fixed Priority ($Sched(\Pi(\gamma(t))) = FP$**)**: After class graph building, each transition t_i has a priority of the task on the processor associated to it ($\varpi(t_i)$). Then, the function ϖ*: Tasks* $\mapsto \mathbb{N}$ guides the firing sequence enumeration. We choose the fireable transition which has the higher priority. At the end of this enumeration, we already have the total time for a firing sequence according GTM. In the case of tasks with the same priority at some point, one of these criterion can guide the firing sequence enumeration between the processes with the same priority: a FIFO choice; an earliest deadline first considering the

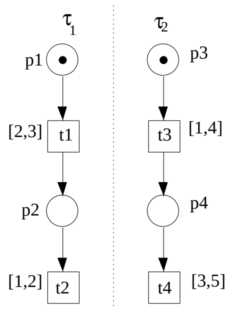

Figure 1. TPN of two tasks on one processor, from [5]

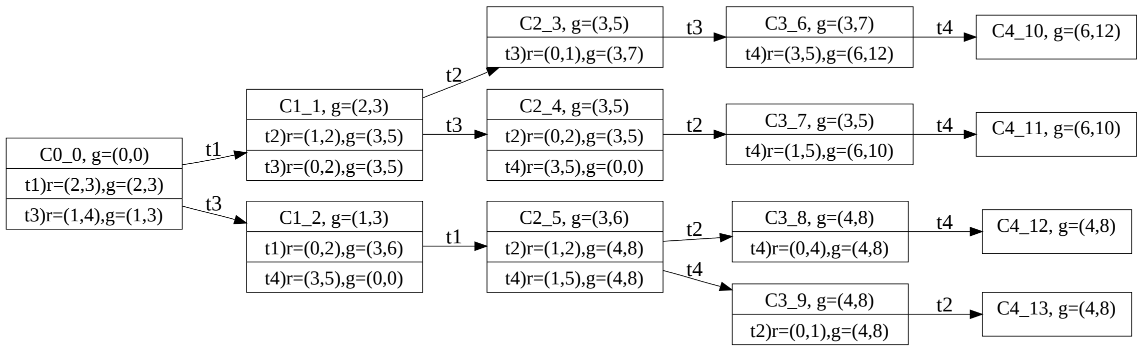

Figure 2. Class graph of TPN representing two tasks on one processor, of Fig. 1

static time $e(t_i)$ for each transition t_i, as we present in the following section; and a random choice.

Earliest Deadline First ($Sched(\Pi(\gamma(t))) = EDF$**)**: Another type of firing sequence can be enumerated on class graph, using the Earliest Deadline First scheduling policy. Then, the function δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$ guides this enumeration. Our criterion is to choose the transition which has the lower deadline given by $\delta(\tau)$ as following. Let $\delta(\tau)$ of a transition t_i calculated in a class c_k such that $c_{k-1}[t_f > c_k$, and defined as: $\delta(\tau) = LFT(r_k(t_i))$. The latest firing time (LFT) of the time $r_k(t_i)$ for each transition t_i is the guide for firing sequence enumeration. As the FP policy, at the end of enumeration already has the total time for a firing sequence.

Considering the TPN of Figure 1. The task τ_1 has priority $\varpi = 1$ and one preemption point. The task τ_2 has also one preemption point, but priority $\varpi = 2$. Then, $\varpi(t_1) = 1$, $\varpi(t_2) = 1$, $\varpi(t_3) = 2$ and $\varpi(t_4) = 2$. The TPN class graph according GTM is presented in the Figure 2. For $Sched(\Pi(\tau)) = FP$, the firing sequence is: t_3, t_1, t_4, t_2. It is interesting to note that t_1 is the only one fireable in the class $C2\_5$ (class C, at level 2, with unique identification 5), even t_4 being enable and t_3 being in the same task that t_4; t_4 executes after t_1 because t_4 has highest priority. The global time of this sequence is $g_{4\_13} = [4, 5]$. For $Sched(\Pi(\tau)) = EDF$, the firing sequence can be t_1, t_2, t_3, t_4, with global time $g_{4\_10} = [6, 9]$, or t_1, t_3, t_2, t_4, with global time $g_{4\_11} = [5, 9]$.

5. Conclusions

GTM avoids the imprecision increase in time information when analyzing transition firing sequences which represents the time interval of system behavior itineraries. This happens when the modeled system presents many concurrent or persistent transitions. Also, GTM state classes describe intervals in both global, based on the simulation beginning, and relative, based on the class entry moment, time information. This increases the analysis power of our approach.

The essence our approach is to verifiy scheduling scenarios generated using GTM from TPN modeled using design patterns. This design patterns represent a set of tasks and their interactions as proposed by [5] and can be tasks on one processor, cyclic tasks synchronized *via* a semaphore, semaphore for mutual exclusion and CAN bus access.

The main contribution of our work is to apply the global time method to real-time software based on tasks with fixed priority and earliest deadline first. The main limitation of the proposed approach is the endless enumeration of classes in cyclic nets, according to the indefinite increase the global time. For the application in the context of prototyping this problem is currently treated by limiting the number of execution cycles of tasks, reflected by the class levels during the generation of graph. Even with this limitation the analysis is still useful as corresponds to the repetition of the initial critical instant for real-time systems based on cyclic tasks.

References

[1] P. Merlin, “A study of recoverability of computer systems,” Ph.D. dissertation, University of California IRVINE, 1974.

[2] B. Berthomieu and M. Menasche, “A state enumeration approach for analyzing time petri nets,” in *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, Italy, sep 1982.

[3] D. L. Dill, “Timing assumptions and verification of finite-state concurrent systems,” in *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. London, UK: Springer-Verlag, 1990, pp. 197–212.

[4] L. A. Cortés, P. Eles, and Z. Peng, “Verification of embedded systems using a Petri net based representation,” in *ISSS ’00: Proceedings of the 13th international symposium on System synthesis*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 149–155.

[5] D. Lime and O. H. Roux, “Expressiveness and analysis of scheduling extended time Petrinets,” in *5th IFAC Int. Conf. on Fieldbus Systems and Applications,(FET’03)*. Aveiro, Portugal: Elsevier Science, Jul. 2003, pp. 193–202.

[6] ——, “Formal verification of real-time systems with preemptive scheduling,” *Journal of Real-Time Systems*, vol. 41, no. 2, pp. 118–151, 2009.

[7] E. A. Lima, R. Lüders, and L. A. Künzle, “Uma abordagem intervalar para a caracterização de intervalos de disparo em redes de petri temporais,” *SBA. Sociedade Brasileira de Automática*, vol. 19, no. 4, p. 379, 2008, in Portuguese, http://dx.doi.org/10.1590/S0103-17592008000400002.

[8] B. Berthomieu and M. Diaz, “Modeling and verification of time dependent systems using petri nets,” *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259–273, March 1991.