Modeling Flexible Real Time Systems with Preemptive Time Petri Nets

G.Bucci, A.Fedeli, L.Sassoli, E.Vicario

Dipartimento Sistemi e Informatica - Università di Firenze, Italia
E-mail {bucci,fedeli,sassoli,vicario}@dsi.unifi.it

Abstract

Preemptive Time Petri Nets are obtained by extending Time Petri Nets with an additional mechanism of resource assignment which makes the progress of timed transitions be dependent on the availability of a set of preemptable resources, and with the capability to make transition times and priorities be dependent on the marking.

The combination of these capabilities supports description and verification of flexible real time systems running under preemptive scheduling, with periodic, sporadic and one shot processes, with non-deterministic execution times, with semaphore synchronizations and precedence relations deriving from internal task sequenzialization and from interprocess communication.

The expressive capabilities of the model and the type of results that can be derived through symbolic enumeration of its dense-timed state space are illustrated with reference to a flexible system mixing dynamic acceptance and performance polymorphism.

1. Introduction

In hard real-time systems, timeliness can be maintained under transient overloading conditions through mechanisms of flexibility which adapt the operation so as to guarantee a degraded but still correct functionality. In the *imprecise computation* approach, tasks are decomposed into a mandatory part that must always be served and an optional part that monotonically increases the computation value and can be terminated as soon as necessary [17] [15] [14] [20]. In the *dynamic acceptance* approach, a task can be either mandatory or optional; in the latter case, the task can be rejected, but, if accepted, it must be completed on time without causing any previously guaranteed task to fail [21] [18]. In *performance polymorphism*, a computation can have multiple implementations differing in the amount of time and resources consumed and in the value of produced results [18]; during the run-time, different implementations are selected depending on the actual availability of resources.

The adoption of flexibile computations has a twofold impact on the problem of predictability: on the one hand, flexibile computations tend to invalidate assumptions about the tasking set structure which enable the application of low-complexity schedulability analysis techniques such as Rate Monotonic Theory; on the other hand, flexibility provides the system with adaptation capabilities which smooth the tradeoff between the need for dynamic behavior and the power of static analysis. This motivates the interest in verification techniques based on the joint usage of operational specification and state space analysis, which can encompass complex tasking models at the expense of high computational complexity.

In [11] [12], a timed process algebra is used to specify both the application processes the policy used to regulate their contention on resources. This captures complex tasking models running under various contention scheduling disciplines (e.g. fixed priority and earliest deadline first), and naturally encompasses flexible computations. The operational semantics of the specification opens the way to simulation and to reachability analysis and algebraic equivalence checking supporting a joint approach to schedulability analysis and verification of safety properties [24]. In [16], the task model is described as a set of timed systems, which captures complex tasking sets, running under preemptive scheduling, with natural representation of non-deterministic timing. The task model includes choices that are supposed to be controllable by the scheduler. This opens the way to a synthesis problem which consists in deriving a control invariant that determines controllable actions so as to confine system behavior within a fragment of the state space where safety and timeliness requirements are satisfied. During the run-time, enforcement of the actions that maintain the invariant is achieved through a system controller that tracks executed events with respect to the specification of the system.

As a common trait, both [11] and [16] rely on the assumption of a *discrete* model of time to support explicit enumeration of the state space. This permits the treatment of models with rich expressivity, but also leads to an inherent conflict between the fineness of the tick assumed in the modeling stage and the size of the state space enumerated in the analysis step. Also, the assumption of a discrete tick may not fit the actual characteristics of systems composed through asynchronous interaction.

Symbolic enumeration methods have been largely studied as a means to overcome both the problems through the enumeration of classes of states that are equivalent for the objectives of analysis. In particular this has been largely developed for models with *dense* or continuous timing, such as Timed Automata [2] [6] and Time Petri Nets [8]. Unfortunately these modeling notations are not able to capture the case of clocks that are suspended and then resumed, preventing the description of systems which run under preemptive scheduling. Hybrid Automata [1] can represent suspended clocks, but they require that the system be cast into a much wider class of models involving a much higher complexity of analysis. Multirate automata reduce the expressive power of hybrid automata and still permit representation of preemption. However, despite their reduced expressivity, no specific methods have been proposed to simplify their analysis.

In this paper, we introduce an extension of Time Petri Nets [22], which we call Preemptive Time Petri Nets, which allows representation of complex and densely timed tasking models running under priority scheduling and applying mechanisms of dynamic acceptance and performance polimorphism. Models can be analyzed through a symbolic state space enumeration method which supports reachability analysis and evaluation of tight bounds on the time elapsed between events along critical execution sequences.

The rest of the paper is organized in three sections. In Sect.2, Preemptive Time Petri Nets are introduced, and the principle applied to support their analysis in a dense domain is briefly described. In Sect.3, the proposed modeling technique is applied to a case example which illustrates the expressivity of the model and the kind of results that can be derived in its analysis. Conclusions are drawn in Sect.4.

2. Preemptive Time Petri Nets

Preemptive Time Petri Nets extend the basic model of Petri Nets [25] [13] with the timing semantics of Time Petri Nets [22] [3], with an original mechanism of resource assignment which conditions the advancement of timers of enabled transitions, and with the capability to let transition firing intervals and priorities be dependent on the net marking.

2.1. Syntax

A Preemptive Time Petri Net (pTPN) is a tuple

$$\begin{aligned} pTPN = \;& \langle P; T; A^-; A^+; A^{\cdot}; M; \\ & FI^s(M); \tau; \\ & Res; Req; Prio(M) \rangle \end{aligned} \tag{1}$$

- The first six members correspond to the basic model of Petri Nets [25] with the addition of inhibitor arcs: P and T are disjoint sets of *places* and *transitions*, respectively; A^-, A^+ and $A^{\cdot}$ are relations on places and transitions called precondition, postcondition and inhibitor arcs, respectively ($A^- \subseteq P \times T$ and $A^+ \subseteq T \times P$ and $A^{\cdot} \subseteq P \times T$). A place p is said to be an *input* or an *output* place for a transition t if there exists a precondition or a postcondition from p to t or viceversa, (i.e. if $\langle p, t\rangle \in A^-$ or $\langle t, p\rangle \in A^+$, respectively). A place p is an inhibitor for transition t if there exists an inhibitor arc from p to t (i.e. if $\langle p, t\rangle \in A^{\cdot}$). M is the (initial) marking, associating each place p with a natural number of *tokens* ($M : P \to \mathbb{N}$).

  P, T, A^-, A^+ and $A^{\cdot}$ comprise a bipartite graph which is represented by drawing places as circles, transitions as bars, and preconditions and postconditions and inhibitor arcs as directed edges (see Fig.1).

- $FI^s(M)$ associates each transition t with a *firing interval* delimited by an *earliest firing time* $EFT^s(t, M)$ and a (possibly infinite) *latest firing time* $LFT^s(t, M)$ dependent on the marking M. Besides, τ associates each transition with a *time to fire* value.

  In the graphic representation, static firing intervals are annotated close to their corresponding transitions in square brackets (see Fig.1).

- Res, Req and $Prio(M)$ account for resources, for resource requirements issued by transitions, and for priorities used to serve requests: Res is a set of *resources* (disjoint from T and P); Req associates each transition with a subset of Res (i.e. $Req : T \to 2^{Res}$); and $Prio(M)$ associates each transition with a priority level that may depend on the marking M (i.e. $Prio : T \times \mathbb{N}^{\#P} \to \mathbb{N}$). In the graphic representation, resource requirements and priorities are represented by labeling each transitions with the set of its required resources (in curl brackets) and its priority level. Priorities are not reported for transitions which do not require any resource.

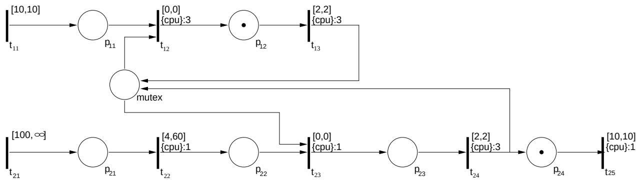

Figure 1. The graphic representation of a Preemptive Time Petri Net.

2.2. Semantics

Preemptive TPNs are associated with an operational semantics. The state is the pair $\langle M, \tau \rangle$, where M is the *marking* and τ is the *time to fire*, and it evolves according to a transition rule made up of two clauses of *firability* and *firing*.

- *Firability:* A transition t_o is *enabled* if each of its input places contains at least one token and none of its inhibiting places contains any token.

  An enabled transition t_o is *progressing* if and only if every resource it requires is not required by any other enabled transition with a higher level of priority. Transitions that are enabled but not progressing are said to be *suspended*.

  A transition t_o is *firable* if it is progressing and its time to fire $\tau(t_o)$ is not higher than the time to fire of any other progressing transition.

  According to these rules, Res represents a set of preemptable resources that condition the progress of computations modeled by transitions. Resources are assigned to transitions according to a priority preemptive protocol based on $Prio$. The protocol for resource contention could be easily refined without affecting the essence of analysis techniques that can be applied on pTPNs, provided that resource assignment can be determined on the basis of transition properties and of the current marking of the net. This does not encompass earliest deadline first which would require the assignment be decided also on the basis of timer values.

  In a uni-processor system, the set of resources is usually made up of a single element, the cpu, which is required by any transition modeling a computation. Note that a transition which is not associated with any resource requirement is progressing whenever it is enabled, which usually serves to model the advancement of a timer.

  In the context of Petri Net applications, modeling of preemptive behavior was addressed in [5]. In that formulation, each transition can be associated with one out of three execution policies differing in the way the clock (there called *age*) is maintained or reset when the transition is disabled by the lack of a token in any input place. In particular, for transitions associated with the so called *preemption resume* policy, the clock is frozen when the transition is disabled and it is resumed when the transition is enabled again. Whereas, for transitions following a *preemption repeat different*, the clock is lost when the transition is disabled.

  The key difference with respect to that formulation is that pTPNs consider two separate mechanisms that condition the progress of times to fire: while the lack of a token in any input place resets the time to fire of the transition, the lack of a required resource suspends the progress of the time to fire which is then resumed when the transition is assigned the resource. The combination of the two mechanisms permits a separate representation of explicit process dependencies due to interprocess communication from the management of preemptable resources which is driven by the operating system scheduler. This permits a direct representation of realistic preemptive scheduling mechanisms without impacting on the complexity of models.

- *Firing:* When a transition t_o fires, the state of the net $\langle M, \tau \rangle$ is replaced by a new state $\langle M', \tau' \rangle$. The marking M' is derived from M by removing a token from each input place of t_o, and by adding a token to each output place of t_o. In the transformation, a temporary marking M_{tmp} is also derived:

$$\begin{array}{ll} M_{tmp}(p) = M(p) - 1 & \forall p.\langle p, t_o \rangle \in A^- \\ M'(p) = M_{tmp}(p) + 1 & \forall p.\langle t_o, p \rangle \in A^+ \end{array} \quad (2)$$

Transitions that are enabled both by the temporary marking M_{tmp} and by the final marking M' are said *persistent*, while those that are enabled by M' but not by M_{tmp} are said *newly enabled*. If t_o is still enabled after its own firing, it is always regarded as newly enabled.

The time to fire τ' of any transition enabled by the new marking M' is computed in a different manner for newly enabled transitions, for persistent-progressing transitions, and for persistent-suspended transitions. For any newly enabled transition t_a, the time to fire takes an non-deterministic value falling in the static firing interval associated with the transition under the new marking M':

$$EFT^s(t_a, M') \le \tau'(t_a) \le LFT^s(t_a, M') \quad (3)$$

For any persistent transition t_i that was progressing in the previous state, the time to fire $\tau(t_i)$ is reduced by the value of the time to fire of t_o:

$$\tau'(t_i) = \tau(t_i) - \tau(t_o) \quad (4)$$

Finally, for any persistent transition t_x, that was suspended in the previous state, the time to fire remains unchanged:

$$\tau'(t_x) = \tau(t_x) \quad (5)$$

2.3. Example

Referring to the net of Fig.1, consider the case that the marking is $p_{12}p_{24}$ (as shown in the picture). Enabled transitions are t_{11}, t_{13}, t_{21}, and t_{25}. Among these, t_{11}, t_{13}, and t_{21} are progressing, while t_{25} is suspended due to the concurrency with t_{13} on resource cpu. Suppose now that in the current state enabled transitions have the following times to fire: $\tau(t_{11}) = 7.1$, $\tau(t_{21}) = 82.3$, $\tau(t_{13}) = 1.5$, and $\tau(t_{25}) = 1.2$; in this case, the next firing transition is t_{13}, as t_{11} and t_{21} have higher time to fire, and t_{25} is suspended. At the firing of t_{13}, a token is removed from p_{12} and one is added to $mutex$. Transitions t_{11}, t_{21}, and t_{25} persist, no other transitions are newly enabled, but transition t_{25} becomes progressing. The time to fire of t_{11} and t_{21} are reduced by 1.5 time units, while the time to fire of t_{25} holds the previous value.

2.4. State Space Analysis of pTPN models

The state of a pTPN depends not only on the discrete marking but also on transition timers which take values in a dense space. To cope with this density, various concepts of equivalence among states have been proposed to support symbolic analysis of the state space of such formalism as Timed Automata [7] [1] [23] and Time Petri Nets [4] [3] [8].

In particular, for Time Petri Nets, a discretely enumerable reachability relation is obtained by collecting together the states that are reached through the same firing sequence but with different firing times [4] [3]. This equivalence yields a compact partitioning of the state space in *state classes* which are sufficiently represented by a marking and by a *firing domain*. These domains, also called *time zones* [7], are expressed as a set of linear inequalities in the form of a Difference Bounds Matrix (DBM) constraining the times to fire of enabled transitions. This kind of representation, which results in polynomial complexity both in the space required to encode each class and in the time needed to derive a class successor, enables reachability analysis, identification of feasible execution sequences, and evaluation of a tight profile for the variety of feasible timings that can be applied to each execution sequence [8].

Unfortunately, the form of a Difference Bounds Matrix is not sufficient to represent timing constraints induced by the contemporary presence of progressing and suspended transitions, which occurs in the modeling of a preemptive system: the repeated application of the succession transformation of Equations (3) through (5) yields more and more complex domains until becoming a general linear programming problem with a number of inequalities exponential in the number of enabled transitions. This results in exponential complexity both in the representation and in the manipulation of firing domains.

To circumvent both size and time complexities, in [9] each firing domain is replaced with the tightest approximation which fits the form of a difference bounds matrix. This permits the enumeration of an approximate, but conservative, relation of succession among state classes. This relation can be captured into a *reachability graph* which permits to obtain necessary conditions for the reachability of a state or for the feasibility of an execution sequence.

In addition, for any execution sequence identified in the reachability graph, an exact profile of feasible timings can be derived from timing constraints included in visited state classes through the solution of a general linear programming problem with a number of unknown values equal to the numebr of classes visited along the trace. The profile permits to evaluate the minimum and the maximum time that can elapse between a stimulus (e.g. a task release) and its response (e.g. the task completion). As a relevant by product, the derivation of the profile provides a necessary and sufficient condition for the actual feasibility of the execution sequence. This permits to clean up false behaviors induced by the approximation in the enumeration of the successor relation between classes.

Analysis methods supporting the treatment of preemption have been implemented in an extension of the ORIS tool [10][8], which is a mixed C++/Java implementation running on Windows and LINUX platforms. ORIS supports visual editing and interactive animation of pTPN models, symbolic state space enumeration, and interactive trace analysis. In particular, ORIS is effectively employed for the detection of the worst (and best) case execution time between a stimulus and a response event.

3. Modeling and Verifying Tasking Sets with Flexible Computations

Preemptive Time Petri Nets support description and analysis of complex tasking models running under preemptive or cooperative scheduling with various mechanisms of flexibility. In particular, the tasking model can include periodic, sporadic and one-shot processes process, with nondeterministic execution times, with semaphore synchronizations and precedence relations deriving from internal task sequentialization and from interprocess communication.

This expressive power is illustrated in this section with reference to a flexible real time system composed of a set of dependent processes, scheduled by priority on a single processor. The pTPN model is derived from an intuitive (yet unambiguous) description of the tasking set (see Fig.2), in a manner that could be made automatic. The pTPN model is then developed to highlight the complexity of the analysis and the kind of results that it can yield. This includes not only the derivation of worst completion times, but also the automatic identification of example behaviors (*witnesses*) yielding critical execution conditions. In the presentation, these behaviors are figured through conventional time-line diagrams that are reconstructed automatically from the analysis of the graph (see for instance Fig.3.3). This prospects the use PTPNs and their analysis methods as hidden nucleus of a verification environment where the designer describes and analyzes a complex tasking set using intuitive and practical notations.

3.1. A Case of Study

The system is comprised of four processes P^1, P^2, P^3, P^4 sketched in the intuitive (yet unambiguous) representation of Fig.2:

- P^1 is periodic with period of 5 time units, it runs at priority level 4, and it is composed of two sequential steps. The first step has a computation time of 1 time unit. The second step requires the control over a $mutex$ semaphore, and it accepts two different *polymorphic* implementations which take either 1 or 2 time units [18].

- P^2 is periodic, with period of 15 time units, it runs at priority level 2, and it is composed of two sequential steps lasting 2 and 1 time units: the first step requires the control of $mutex$ semaphore.

- P^3 is sporadic with a minimum interarrival time of 10 time units. Tasks of P^3 are optional and can be *dynamically discarded* under overloading conditions [21] [18]. If accepted, tasks run at priority level 3, and must be served in time.

- P^4 is a periodic process with period of 30 time units running at priority level 1. It is composed of a single step with execution time uniformly distributed between 2 and 6 time units.

Process P^1, P^2 and P^4 have deadlines coincident with the period. For P^3 the deadline is equal to the minimum interarrival time.

3.2. A pTPN model

Fig.3 shows a pTPN model of the tasking set, which closely reflects the structure of the intuitive representation of Fig.2.

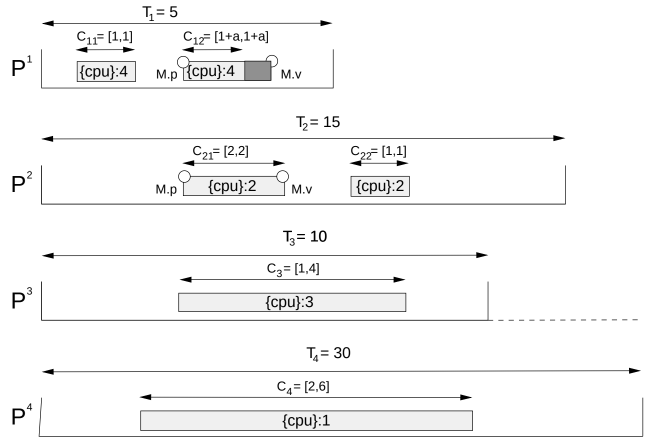

Figure 2. A tasking set with four processes P^1 through P^4.

- Process P^1 is modeled by the sequence of transitions t_{10} through t_{13}: t_{10} releases tasks periodically; t_{11} is the first computation step; t_{12} acquires the *mutex*; t_{13} is the second computation step. Note that the firing interval of t_{13} is expressed in terms of a parameter a which can be either 0 or 1 so as to account for a *polimorphic implementation*.

- Process P^2 is modeled in similar manner by transitions t_{20} through t_{23}.

- In process P^3 transition t_{30} models task releases with a minimum but not with a maximum interarrival time; transitions $accept$ and $discard$ have dynamic priorities depending on the net marking so as to implement different *dynamic discard* policies, transition t_{31} accounts for the computation.

- Process P^4 is modeled by transitions t_{40} and t_{41}.

3.3. Achieving correctness through dynamic acceptance

If every optional task of process P^3 is accepted (i.e. *accept* has always priority over *discard*), all four processes happen to miss their deadlines and state space enumeration diverges towards an apparently unbounded number of tokens in places p_{22} and p_{40}.

Seeking for a dynamic acceptance strategy that can avoid the failure, we reject any optional task which is released while a computation of the low priority process P^4 is pending. This is obtained by assigning $accept$ a higher priority than $discard$ iff place p_{40} is empty.

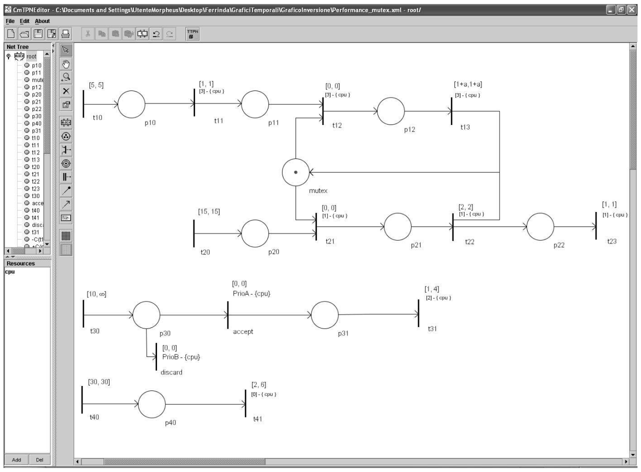

Figure 3. ORIS supports visual editing and interactive animation of pTPNs: a model of the tasking set described in Fig 2.

State space enumeration terminates with 2238 classes. Trace analysis indicates that: P^2 misses the deadline along a trace where a task of P^3 is accepted while P^2 is still pending in its second step (i.e. a token is in place p_{22}); P^1 misses the deadline along a trace where a priority inversion occurs between processes P^1 and P^3. A witness identified by ORIS for the latter failure is illustrated in Fig.3.3.

To overcome the problem: the priority of the computation of process P^2 in the critical section (i.e. transition t_{22}) is boosted to level 4 according to a *priority ceiling emulation* protocol [19]; and the acceptance policy is restricted so as to discard optional tasks also in the case that an instance of P^2 is pending in the second step (a token in place p_{22}).

Under this policy, state space enumeration terminates with 2135 classes and trace analysis indicates that all processes meet all their deadlines. Specifically, the worst case completion times for the processes P^1 through P^4 are equal to 4, 10, 13 and 28, respectively.

In particular, Fig.3.3 shows how the system correctly manages the same phasing of arrivals that led to the priority inversion illustrated in Fig.3.3.

3.4. Enhancing quality of service through polymorphic computations

Once correctness is guaranteed through the policy of dynamic acceptance, the quality of service of the system can be enhanced by adapting the time allocated for the computation in the critical section of process P^1 (i.e. by setting the parameter a of the firing interval of transition t_{13}).

If the computation is always performed at the high level

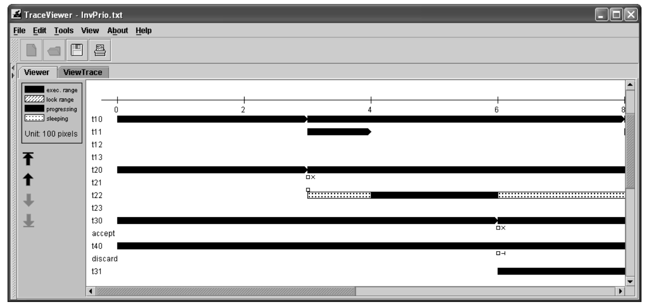

Figure 4. The ORIS tool supports automated identification and interactive visualization of traces realizing minimum and maximum stimulus-response delays. The example shows a trace leading to the failure of process P^1 due to a priority inversion. At time +3 after some initial state: both processes P^1 and P^2 release a task (transitions t_{10} and t_{20} fire); while P^1 carries out the first computation step (transition t_{11}), P^2 acquires the semaphore and its first step (transition t_{22}) becomes ready but suspended (dashed filling indicates suspension). At time +4: P^1 completes the first step but cannot acquire the mutex (t_{12} is not enabled), and P^2 becomes running (transition t_{22}). At time +6: the sporadic process P^3 releases a task (transition t_{30} fires) which preempts P^2 (t_{22} becomes suspended); in this condition, P^1 is blocked by P^2 which is suspended by P^3, i.e. priorities of P^1 and P^3 are inverted through the mutex hold by P^2. At time +8: P^1 is still blocked on the mutex and misses its deadline (5 time units after release occurred at time +3).

of quality (i.e. if a is always set equal to 1), both P^2 and P^4 miss some of their deadlines (see Fig. 3.4): to avoid the failure, performance enhancement must be selective.

To this end, we assume that the computation in the critical section of process P^1 is performed at the higher level of quality iff process P^2 is not blocked on the $mutex$ semaphore and it is not suspended in the second computation step (i.e. iff (no tokens in p_{20}) and (no tokens in p_{22})). Under this policy, trace analysis indicates that all deadlines are met. In particular, worst case completion time for processes P^1 through P^4 is equal to 4, 10, 15, and 30, indicating that no further load can be accepted by the system

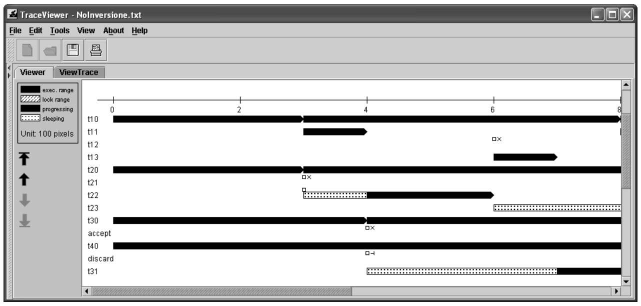

Figure 5. A trace for a variation of the model of in which the priority of transition t_{22} is set equal to 4, according to a priority ceiling emulation protocol. After acquisition of the semaphore (t_{21}), P^2 raises its priority to the level of P^1, thus becoming non preemptable by P^3. Process P^1 completes its computation(transition t_{13}) with 1 time before the deadline at time +8.

4. Conclusions

Preemptive Time Petri Nets extend Time Petri Nets with an additional mechanism of resource assignment which makes the advancement of computations be dependent on the availability of a set of preemptable resources and by letting transition parameters be dependent the net marking. The resulting notation permits a convenient application of Petri Nets in the modeling of complex tasking sets running under preemptive scheduling with mechanisms of dynamic acceptance and performance polimorphism.

pTPN models can be validated with respect to requirements on the logical sequencing of events (e.g. the satisfaction of a precedence or a mutual exclusion) as well as on their quantitative timing (e.g. the maximum time elapsed between any two given events). This combines tight schedulability analysis with exhaustive verification of the correctness of logical sequencing. In addition, the method identifies witnesses of critical execution sequences, which provide insight into system behavior during the design process.

Similar modeling and analysis capabilities can be achieved through other methods reported in the literature, and notably in [11] and [16]. As opposed to those models, which are discretely timed, preemptive Time Petri Nets are associated with a dense model of time and are analyzed using a symbolic enumeration approach.

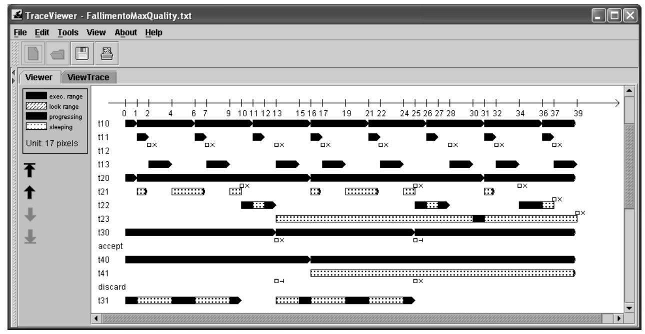

Figure 6. A trace for the model in which process P^1 always computes at higher level of quality. Processes P^2 and P^4 miss their deadline.

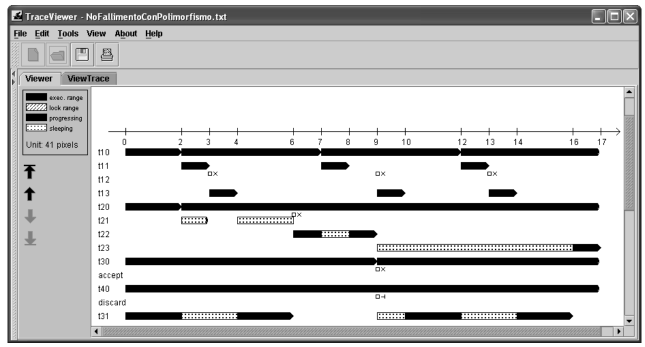

Figure 7. A trace for the model in which process P^1 is performed at different levels of quality depending on loading conditions. All process respect their deadlines.

References

[1] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *IEEE Real-Time Systems Symposium*, pages 2–11, 1993.

[2] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995.

[3] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3), March 1991.

[4] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9, pages 41–46. Elsevier Science Publishers, Amsterdam, 1983.

[5] A. Bobbio, A. Puliafito, and M.Telek. A modeling framework to implement preemption policies in non-Markovian SPNs. *IEEE Transactions on Software Engineering*, 26(1):36–54, January 2000.

[6] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Con-*

trol, volume 1066, pages 208–219, Rutgers University, New Brunswick, NJ, USA, 22–25 October 1995. Springer.

[7] D.Dill. Timing assumptions and verification of finite-state concurrent systems. In *Workshop on Computer Aided Verification Methods for Finite State Systems*, 1989.

[8] E.Vicario. Static analysis and dynamic steering of time dependent systems using time petri nets. *IEEE Trans.Soft.Eng.*, August 2001.

[9] G.Bucci, A.Fedeli, and E.Vicario. Timed state space analysis of fixed priority preemptive systems. also submitted to *IEEE Transactions on Software Engineering*, 2002.

[10] G.Bucci and E.Vicario. Compositional validation of time-critical systems using communicating time petri nets. *IEEE Transactions on Software Engineering*, 26(12), December 1995.

[11] H.Ben-Abdallah, J.-Y.Choi, D.Clarke, Y.-S.Kim, I.Lee, and H.-L.Xie. A process algebraic approach to the schedulability analysis of real time systems. *Real Time Systems*, 15(3):189–219, 1998.

[12] I.Lee, P.Bremond-Gregoire, and R.Gerber. A process algebraic approach to the specification and analysis of resource bound real time systems. *Proceedings of the IEEE*, 82(1):158–171, Jan. 1994.

[13] J.L.Peterson. Petri nets. *ACM Computing Surveys*, 9(3), September 1977.

[14] J.W.S.Liu and W.K.Feng. Algoritms for scheduling real-time tasks with input error and end-to-end deadliness. *IEEE Transactions on Software Engineering*, 1997.

[15] J.W.S.Liu and W.K.Shih. Algoritms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers*, 1995.

[16] K.Altisen and J. G.Goessler. Scheduler modeling based on the controller synthesis paradigm. *Real Time Systems*, 23:55–84, 2002.

[17] K.Jay, L.J.Y.Chung, and J.W.S.Liu. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 1990.

[18] K.Jay, L.Kevin, and B.Kenny. Building flexible real-time system using the flex language. *IEEE Computer*, 1991.

[19] L.Sha, R.Rajkumar, and J.P.Lehoczky. Priority inheritance protocols: an approach to real time synchronization. *IEEE Trans.on Computers*, 39(9), September 1990.

[20] M.Caccamo, L.Abeni, G.C.Buttazzo, and G.Lipari. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 2002.

[21] S. Natarajan and K.J.Lin. Expressing and maintaining constraints with flex. In *Real-Time Symposium*, 1988.

[22] P.Merlin and D.J.Farber. Recoverability of communication protocols. *IEEE Transactions on Communications*, 24(9), September 1976.

[23] R.Alur, C.Courcoubetis, and D.Dill. Model-checking for real-timesystems. In *5th Symp. on Logic in Computer Science*, 1990.

[24] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with PARAGON. *Annals of Software Engineering*, 7:211–234, 1999.

[25] T.Murata. Petri nets: Properties, analysis and applications. *IEEE Proceedings*, 77(4), April 1989.