©2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Static Analysis and Dynamic Steering of Time-Dependent Systems

Enrico Vicario, *Member*, *IEEE Computer Society*

Abstract—An enumerative technique is presented which supports reachability and timeliness analysis of time-dependent models. The technique assumes a dense model of time and uses equivalence classes to enable discrete and compact enumeration of the state space. Properties of timed reachability among states are recovered through the analysis of timing constraints embedded within equivalence classes. In particular, algorithms are given to evaluate a tight profile for the set of feasible timings of any untimed run. Runtime refinement of static profiles supports a mixed static/dynamic strategy in the development of a failure avoidance mechanism for dynamic acceptance and a guarantee of hard real-time processes.

Index Terms—Time-dependent systems, hard real-time systems, timeliness predictability, enumerative static analysis, dynamic task guarantee, quantitative timing estimation, Time Petri Nets.

1 Introduction

Time affects the development of a large class of reactive systems, whether because of explicit real-time constraints [45], [16], or due to sequencing limitations which implicitly result from timed behavior [25], [18]. Engineering the *predictability* of both these factors is motivated by their inherent complexity, often exacerbated by the criticality of the application context [44], [36].

1.1 Timeliness Analysis in Real Time Systems

In the literature of real-time systems, a number of scheduling techniques have been proposed, which control the sequencing of execution and support some kind of feasibility tests to guarantee the satisfaction of hard deadlines. Most proposed and practiced methods rely on the assumption of a highly structured system organized as a set of tasks with limited mutual dependencies and scheduled under preemptive priority. In the theory of rate monotonic, a sufficient condition permits one to test the guarantee for a set of independent periodic tasks, each characterized by a period, a worst case computation time, and a deadline, and assigned with a fixed priority inverse to the period (shortest period first) [26]. The test can be extended to account for synchronization on exclusive resources [41], [23], [42], but not for precedence constraints, such as those deriving from intertask communication. Asynchronous repetitive tasks with a minimum interarrival time can also be encompassed in the analysis [36], but constraints on the interarrival times of different tasks cannot be accounted, thus, often leading to pessimistic and loose bounds. In [46], a set of one-shot tasks are decomposed as sequential chains of jobs, each associated with an assigned priority, a minimum, and a maximum execution time, with mutual resource synchronizations but without any precedence constraints between jobs in different chains. Under these conditions, if jobs are scheduled under any fixed priorities, completion time of tasks is approximately estimated in polynomial time.

Timeliness analysis for more complex tasking models generally results in NP-hard problems [7] that are managed differently depending on the objectives of the analysis.

In dynamic planning-based approaches [38], the analysis is finalized to decide the admission of dynamically arriving (optional) tasks. In this case, the complexity of planning is limited by accepting nonoptimal decisions which may result in the rejection of tasks that could be guaranteed. In the Spring kernel [43], the schedule for a set of nonpreemptable tasks with resource requirements and precedence constraints is derived using a branch and bound approach in which the complexity (and the optimality) of search is reduced to polynomial time by constraining the depth of backtrack [47], [40], [24]. In a different approach, in [22], backtrack is limited by a self-adjusting algorithm constraining the time available for planning.

In static table driven approaches [38], schedulability analysis is carried out before runtime to produce some kind of table that will drive dispatching during the execution. In this case, analysis may afford a higher complexity to derive a feasible schedule. In [48], a scheduling table is derived using a branch and bound search for the case of a set of one-shot tasks with exclusive resources, precedence constraints, and with explicit limitations on the preemptability of tasks. Periodic tasks can also be encompassed, but their single instances must be managed individually by extending the analysis scope to the hyperperiod. A similar search is employed in [35] for a set of periodic nonpreemptable tasks with precedence constrains. In this case, nonperiodic processes require periodic transformation based on the minimum interarrival time or on a periodic server. In [39], a feasible schedule for a set of periodic tasks with precedence, communication, and replication constraints is determined as a part of a distributed task allocation problem.

• *The author is with the Dipartimento Sistemi e Informatica, Università di Firenze 3, via santa Marta, 50139, Firenze, Italy.*
E-mail: vicario@dsi.unifi.it.

Manuscript received Spring 1997; revised 12 June 2000; accepted 28 June 2000.
Recommended for acceptance by D. Rosenblum.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 105046.

0098-5589/01/$10.00 © 2001 IEEE

A common trait of [48], [35], [39], and, more generally, of branch and bound techniques proposed for static table driven scheduling, is that timing analysis is based on worst case execution times, without considering the possible early completion of tasks. This provides no support for dynamic refinement of the schedule to safely reclaim resources on the basis of the actual time spent in the execution.

1.2 Enumerative Analysis of Densely Timed Models

Timing analysis for table driven scheduling can be extended to encompass complex tasking models and to manage nondeterministic execution times taking values within dense intervals, by resorting to enumerative techniques developed in the area of computer aided verification.

In the modeling stage, the tasking model can be represented using untimed operational formalisms augmented with timers constraining the intervals in which transitions are either prevented or forced to execute. Timed Automata and Time Petri Nets are notable examples layering these two kinds of *weak* and *strong* timing semantics on top of State Transition Systems and Petri Nets, respectively [2], [9], [8]. Such models naturally permit the representation of complex nonpreemptive tasking models, with arbitrary resources and precedences, undeterministic computation times (and durations) taking values within dense intervals, and periodic or asynchronous arrivals possibly including mutual constraints on the arrival times of different processes [31]. Preemptive behavior can also be represented using Rectangular or Hybrid Automata [4], which include timers increasing with variable speed.

In general, models can be directly expressed using an automaton such as a Time Petri Net or a Timed Automaton, or they can be derived automatically from other specifications, such as a process algebra [30], from the code of an ADA program [13], or from a conventional representation of a tasking set with real-time parameters.

In the verification stage, the (timed) state space of the model can be enumerated and analyzed to verify both logical sequencing (e.g., mutual exclusion or reachability) and timing properties (e.g., the satisfaction of a deadline), also accounting for the variability of undeterministic timings taking values within extended intervals. This can be achieved with different techniques which basically reflect the assumptions taken in the representation of timings.

In the straightforward approach, if timers take values within discrete and finite sets, the state space can be obtained by the explicit enumeration of the set of timers interpretations associated with each logical state [19], [33]. This largely eases evaluation of time spent along execution sequences and enables the use of consolidated model checking techniques in the verification of sequencing limitations [12]. However, the approach is likely to blow up enumeration complexity and cannot be applied to the common case in which timers take values within dense or continuous intervals.

To overcome both the limitations, different types of *equivalences* on timing spaces have been proposed, which encapsulate the native density of the state space so as to permit the enumeration of a *discrete* and compact reachability relation between equivalence state classes [3], [9], [8], [18], [4].

In the analysis of Timed Automata [3], *regions* collect states whose timers have equal integer parts and a common ordering of fractional parts. In addition, for each timer, two different evaluations are assumed to be equivalent as soon as both of them exceed the maximum threshold against which the timer can be compared. This type of equivalence is proven to let equivalent states be undistinguishable with respect to the expressive domain of an adequate real-time temporal logic [3]. This enables a real-time model checking approach to verification that can be used to decide whether a given logic or timing constraint is satisfied by every or by any of the possible behaviors of the model. This capability has been embodied in a number of experimental tools for computer aided verification of time-dependent systems [14], [17], [6], [4].

In [9], [8], [10], the timing space of a Time Petri Net is represented in terms of *firing domains*, also called *time zones*, each collecting the multiplicity of timings resulting from the variety of executions which follow a common speed-independent event sequence. The same approach is pursued in [18] for the analysis of Buchi Automata. With respect to the *region* abstraction, *firing domains* enable representation of timing constraints which more effectively fit the untimed sequencing of logical states. In fact, a single firing domain can encompass a complex set of states with timers ranging according to a system of linear inequalities, thus largely increasing enumeration compactness and extending the scope of potential applicability. However, this also weakens the semantics of state classes as different states collected within a single class may enable different sets of runs, differing both in the sequencing of events and in their feasible timing. While still evidencing the effects of timed behavior on the logical sequencing of the model, this weakened semantics obscures the relation between classes and dwelling times in class transitions. This prevents both real-time model checking and quantitative evaluation of constraints limiting the timing of feasible execution traces. In [18], the difficulty is partially circumvented by augmenting the model with fictitious events marking deadline expiration, so as to reduce the verification of real-time constraints to a decision problem on the the ordering of deadlines and process completion events. However, this decisional formulation does not permit the evaluation of an optimal bound on the actual time spent along an execution sequence. Moreover, in so doing, modeling of required behavioral characteristics is tangled with the modeling of the behavior itself. This not only reduces flexibility of verification, but also increases enumeration complexity, especially in the case of multiple deadlines constraining parallel processes with different time scales.

In the implementation stage, the operational semantics of specification models can be directly cast into code, to fully exploit the modeling effort and to preserve predictability achieved through the verification stage. Following [30] and [25], this can be obtained by translating the automata of the model into a network of cooperating modules, which enforce the sequencing of actions under the restriction of watchdogs and timeouts. In [13], the tasking model is derived from an ADA program, which can thus be used directly for the implementation. In both the cases, the

system runs along *any* of the possible behaviors of the model, letting all nondeterministic choices be determined by the environment and by the actual timing of actions. This requires that *every* behavior of the model is proven correct and timely, and does not permit the model include optional activities that can be safely guaranteed only under specific determinations of execution timing. Following an opposite approach, in [20], the analysis is oriented to find *any* correct behavior, which becomes the scheduling table for a cyclic executive. In this case, the model may also include noncorrect or untimely behaviors, but the system still lacks the flexibility to reclaim resources and to refine the scheduling on the basis of runtime information.

1.3 An Enumerative Approach to Static Analysis and Dynamic Steering

In this paper, we propose an enumerative approach to the analysis and of time-dependent tasking models, which partitions the achievement of predictability through the offline and the runtime stages. As a concrete testbed, the approach is applied to the enforcement of a failure avoidance mechanism in the dynamic acceptance and guarantee of nonpreemptive hard real-time tasks.

In the modeling stage, Time Petri Nets are used to represent a set of processes which generate tasks according to a minimum and a maximum task interarrival time and to any kind of precedence constraint among different arrivals and among the arrivals and the logical state of the system. This encompasses as particular cases one-shot, periodic, and repetitive asynchronous processes. Each task has a hard deadline constraining the minimum and the maximum allowable completion time and is comprised of a set of jobs characterized by a nondeterministic execution time taking values within a dense interval, by a set of nonpreemptable required resources and by any kind of precedence constraint. Processes are distinguished as either *mandatory* or *optional*. Tasks generated by mandatory processes must always be guaranteed. Whereas, tasks of optional processes can be accepted or rejected; however, if accepted, they must be serviced in time without causing any previously guaranteed task to fail.

During static verification, the timed state space of the model is enumerated and analyzed to classify acceptance decisions depending on their effect on task guarantee. This distinguishes safe, failing, and critical acceptances corresponding to the cases in which the acceptance certainly maintains correctness, it certainly leads to an untimely behavior, or it may lead to timely or untimely behavior depending on the specific timing determined during runtime. To support this analysis, the enumeration technique of [9], [8] is reformulated and extended so as to permit not only the verification of sequencing limitations deriving from timed behavior, but also the evaluation of quantitative tight bounds for the time spent along execution sequences.

During runtime, a *steering monitor* behaving as runtime dispatcher tracks the evolution of the system through a compact representation of the state space and of the information obtained in the classification of acceptance decisions. This permits one to decide critical acceptances by complementing static predictions with the actual knowledge of the execution timing.

The rest of the paper is organized in four sections. In Section 2, we introduce Time Petri Nets as used in the rest of the paper to model time-dependent systems. In Section 3, the analysis technique of [9], [8] is reformulated and extended to derive expected timing profiles of execution sequences. Static verification and dynamic steering of real-time systems are discussed in Section 4, and two case examples are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2 Time Petri Nets

A Time Petri Net (TPN) is a tuple

$$TPN =< P; T; A^-; A^+; M; A^\cdot; FI^s > . \tag{1}$$

- The first five members comprise the basic model of Petri Nets: P is a set of *places*, T a set of *transitions*, and A^- and A^+ are sets of preconditions and postconditions connecting places to transitions and viceversa, respectively:

$$\begin{aligned} A^- &\subseteq P \times T, \\ A^+ &\subseteq T \times P. \end{aligned} \tag{2}$$

  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 $< p, t > \in A^-$ or $< t, p > \in A^+$), respectively. M (the initial marking) associates each place with a nonnegative number of tokens:

$$M : P \rightarrow N \cup \{0\}. \tag{3}$$

  P, T, A^-, and A^+ comprise a bipartite graph, P and T being disjoint classes of nodes, and A^- and A^+ being relations between them. This graph is graphically represented by drawing places as circles, transitions as bars, and preconditions and postconditions as directed arcs; the tokens of the initial marking are represented as dots inside places (see for instance Fig. 9).

  $A^\cdot$ is a set of inhibitor arcs connecting places to transitions:

$$A^\cdot \subseteq P \times T \tag{4}$$

  inhibitor arcs are represented graphically as dot-terminated arcs (see transition t_{42} in Fig. 11).
- FI^s adds timing constraints to the net by associating each transition t with a *static firing interval* made up of an *earliest* and a *latest firing time*

$$\begin{aligned} FI^s &: T \rightarrow R^+ \times (R^+ \cup \{\infty\}), \\ FI^s(t) &= (EFT^s(t), LFT^s(t)). \end{aligned} \tag{5}$$

  In the graphic representation, static firing intervals are annotated close to their corresponding transitions (see Fig. 9).

TPNs are associated with an operational interpretation defined through a state and a state transition rule.

The state of a TPN is made up of a marking M and a dynamic firing interval FI associating each transition with an *earliest* and a *latest dynamic firing time*:

$$\begin{aligned} state = & < M, FI > \\ & M : P \rightarrow N \cup \{0\} \\ & FI : T \rightarrow R^+ \times (R^+ \cup \{\infty\}) \\ & FI(t) = (EFT(t), LFT(t)). \end{aligned} \tag{6}$$

The state dynamically evolves according to a transition rule made up of three clauses of *firability*, *progress*, and *firing*.

- *Firability.* A transition t_o is *enabled* if each of its input places contains one token at least and none of the places connected to it through an inhibitor arc contains any token.
- *Progress.* A transition t_o is *firable* with firing time τ_o if t_o is enabled and τ_o is neither lower than the earliest firing time $EFT(t_o)$ of t_o, or longer than the latest firing time $LFT(t)$ of any other enabled transition t:

$$EFT(t_o) \leq \tau_o \leq LFT(t) \qquad \forall t \; enabled. \tag{7}$$

- *Firing.* When transition t_o fires with firing time τ_o, the marking of the net and the values associated with the dynamic firing intervals of enabled transitions are changed through the atomic execution of the following three steps:

  1. A token is removed from each of the input places of t_o.
  2. A token is added to each of the output places of t_o.
  3. The firing intervals of all the transitions that are enabled after Step 2 are updated. This occurs in a different manner for *persistent* transitions, i.e., those transitions that are enabled before and after Step 1, and for *newly enabled* transitions, i.e., those transitions that are enabled after Step 2 but not after Step 1:

     a. For any persistent transition $t \neq t_o$, the firing interval is shifted left by the value τ_o:

$$\begin{aligned} EFT(t) &:= max\{0, EFT(t) - \tau_o\}, \\ LFT(t) &:= max\{0, LFT(t) - \tau_o\}. \end{aligned} \tag{8}$$

     b. For any newly enabled transition t, the firing interval is reset to the static value:

$$\begin{aligned} EFT(t) &:= EFT^s(t), \\ LFT(t) &:= LFT^s(t). \end{aligned} \tag{9}$$

     c. If transition t_o itself is still enabled after its own firing, it is regarded as newly enabled and its firing interval is reset to the static value.

The proposed model differs with respect to the common formulation of Time Petri Nets [28] in two aspects:

- According to 3c, a transition which is still enabled after its own firing is always considered as newly enabled, so as to simplify the treatment of states in which a transition has sufficient tokens in its input places to permit multiple firings. This condition, usually referred to as *multiple enabledness*, requires multiple firing intervals be associated with a single transition, and it involves a number of semantic subtleties that are not relevant to the purposes of our discussion.
- Inhibitor arcs allow one to explicitly express priority relationships among transitions [37]. This capability is not essential for the purposes of our treatment, but it often eases the construction of models which prevent marginal execution sequences corresponding to noncorrect behaviors.

3 An Enumerative Approach for Timeliness Analysis of Time Petri Nets

In this section, an enumerative approach is presented which supports the evaluation of tight profiles bounding the time elapsed between the events of the execution traces of a Time Petri Net model. To this end, the enumeration technique of [9], [8] is originally reformulated to make explicit the semantics of the reachability relation between state classes and to provide original close form expressions for the enumeration algorithm. Close form expressions not only reduce the complexity of enumeration but also enable the analytical development that supports derivation of timing profiles for execution traces. These profiles provide tight bounds on the minimum and maximum time elapsing between any two events of the trace and support the analysis of implicit dependencies among transition sequencing and event timing.

3.1 States and State Classes

The execution rule of TPNs implies the notion of a reachability relation between states. In this relation, the firing of the same transition with different values of the firing time leads to states having the same marking but different firing conditions. Since the firing time can take values within a dense interval, the set of states reachable through the firing of any transition is, in general, a *dense* one, and it cannot be enumerated through the usual techniques for reachability and coverability analysis [29]. To overcome the hurdle, the reachability relation between states is replaced through a reachability relation between *state classes*, each collecting the dense multiplicity of states which result from a common sequence of untimed events.

State classes are defined as a formal generalization of the syntactic structure of the state of a Time Petri Net by referring firing times of transitions to a ground reference value $\tau(t_*)$ and by allowing the expression of constraints on the difference between firing times of different transitions:

Definition 3.1. *A state class is a pair $< m, D_m >$, where m is a marking and D_m is a firing domain, i.e., the set of solutions for a set of linear inequalities of the form:*

$$D_m = \begin{cases} \tau(t_i) - \tau(t_j) \leq b_{ij} \\ \forall t_i, t_j \in T(m) \cup \{t_*\} \qquad t_i \neq t_j, \end{cases} \tag{10}$$

for each $[i,j] \in [0, N-1] \times [0, N-1]$
  $b_{ij}^{N-1} = b_{ij}$
for each $k = N-1, \cdots, 0$
  for each $[i,j] \in [0, N-1] \times [0, N-1]$
    $b_{ij}^{k-1} = \min\{b_{ij}^{k}, b_{ik}^{k} + b_{kj}^{k}\}$

Fig. 1. Floyd-Warshall all-shortest-path algorithm. b_{ij}^{k} is the length of the shortest path going from node i to node j without visiting any internal node with index lower or equal to k. According to this, b_{ij}^{N-1} and b_{ij}^{-1} are the direct and the shortest path from node i to node j, respectively.

where $b_{ij} \in R \cup \{\infty\}$ represents assigned coefficients, $\tau(t_i)$ unknown values, and $\tau(t_*)$ stands for the reference time of the domain, i.e., the time instant at which the class is entered.

A set of inequalities in the form of (10) allows for multiple evaluations of coefficients b_{ij} which yield the same solution space. In [18], a *normal* representation is proposed which is characterized by the satisfaction of the triangular inequality:

$$b_{ij} \leq b_{ih} + b_{hj}. \tag{11}$$

This representation links the values of coefficients b_{ij} with the profile of the solution space: $-b_{ji}$ and b_{ij} are the minimum and maximum values of the difference $\tau(t_i) - \tau(t_j)$ which yield solutions for D_m, respectively. In particular, $-b_{*i}$ and b_{i*} are the minimum and the maximum values of $\tau(t_i)$ which yield solutions for system D_m.

The normal representation is proven to uniquely exist and its computation can be formulated as an *all shortest path* problem by representing each unknown value $\tau(t_i)$ as a node and each bound $\tau(t_i) - \tau(t_j) \leq b_{ij}$ as a directed edge of length b_{ij} between nodes $\tau(t_j)$ and $\tau(t_i)$. The problem can be solved with complexity $O(|T(m)|^3)$, where $|T(m)|$ denotes the number of transitions enabled by m. Fig. 1 reports a standard algorithm due to Floyd and Warshall [1].

3.2 Enumerating a Reachability Relation between State Classes

The reachability relation between state classes is defined so as to get rid of the dependency on the dense value of the firing time (see Fig. 2):

Definition 3.2. *A state class, S_{child}, is reachable from class S_{parent} through transition t_o if and only if S_{child} collects all and only the states that are reachable from some state collected in S_{parent} through some feasible firing of t_o.*

Definition 3.2 directly implies a homomorphic relation between the set of firing sequences of the net and the set of paths in the reachability graph of its state classes. In particular, a state s is reachable from some of the states included in a class S_o if and only if state s is collected in some class S reachable from S_o. Moreover, this implies that a transition sequence $\rho_n = t_1 \hat{} t_2 \hat{} \cdots \hat{} t_{n-1} \hat{} t_n$ can be executed from some reachable state of the net if and only if ρ_n is a path in the graph of reachable state classes.

In the following, two algorithmic clauses are given which permit one to identify the outgoing arcs of a generic class S_{parent} and to compute the successors of S_{parent} through each of its outgoing arcs. It is worth noticing that, since the reachability relation between state classes has been defined

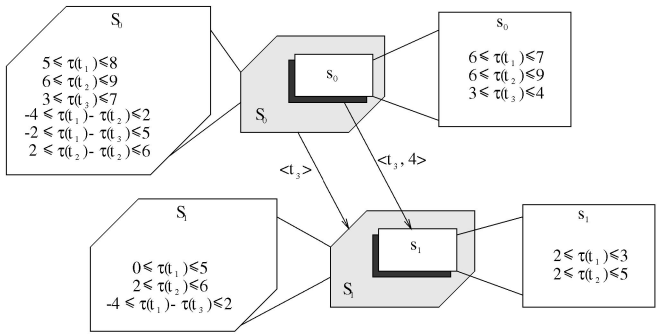


Fig. 2. A succession arc is a pair $< transition, firing_time >$ in the reachability relation between states, and a singleton $< transition >$ in the reachability relation between states classes.

in terms of the properties of classes that are associated by the relation, the two clauses should be supported by a proof of soundness, to demonstrate that they actually compute the relation of Definition 3.2. However, for the sake of conciseness, complete proofs are reported in [27], and only a sketch of their rationale is provided in the following.

Successor Existence: The set of outgoing arcs for a class S_{parent} is the set of transitions that are firable in some state collected within S_{parent} itself. According to the execution rule of transitions, this is a subset of the transitions that are enabled by the marking m_{parent} of S_{parent}. Within this set, a transition t_o is an outgoing arc iff the firing domain D_{parent} of S_{parent} accepts solutions in which the firing time $\tau(t_o)$ of transition t_o is not greater than that of any other enabled transition. This condition can be tested by direct inspection on the normal form of coefficients b_{io} of the firing domain D_{parent} which indicates the maximum acceptable delay of the firing of t_i after t_o:

Clause 3.1. *Given a state class $S_{parent} = < m_{parent}, D_{parent} >$, with D_{parent}represented in normal form as:*

$$D_{parent} = \begin{cases} \tau(t_i) - \tau(t_j) \leq b_{ij} \\ \forall t_i, t_j \in T(m_{parent}) \cup \{t_*\} \text{ with } t_i \neq t_j. \end{cases} \tag{12}$$

transition t_o is an outgoing arc for S_{parent} if and only if $t_o \in T(m_{parent})$ and $b_{io} \geq 0, \forall t_i \in T(m_{parent})$.

Successor Computation: The marking m_{child} of class S_{child} reached from S_{parent} through transition t_o is obtained by moving tokens of m_{parent} according to the execution rule of transitions. For the computation of the firing domain D_{child} of S_{child}, disabled transitions are excluded, and newly enabled transitions are included with firing intervals set to their static values. For each persistent transition t_i, the firing interval must be constrained within the minimum and the maximum delay that t_i can have after the firing of t_o under the constraint that t_o precedes t_i.

Specifically, if the time to fire in the parent and in the successor class are denoted by $\tau_p(.)$ and $\tau_s(.)$, respectively, the time to fire of a persistent transition t_i after the firing of t_o is subject to the constraint $\tau_s(t_i) - \tau_s(t_*) = \tau_p(t_i) - \tau_p(t_o)$. According to the properties of the normal representation of firing domains, the minimum and maximum acceptable value for the difference $\tau_s(t_i) - \tau_s(t_*)$ turns out to be equal to coefficients $-B_{oi}$ and B_{io} of the normal representation of

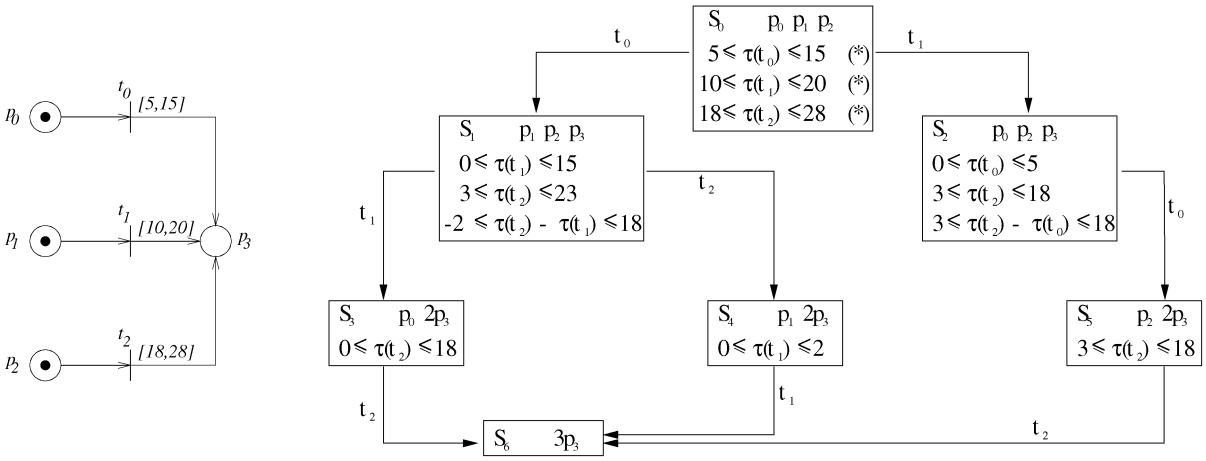


Fig. 3. A simple net and its graph of reachable state classes.

the restricted firing domain $D^{t_o}_{parent}$ which augments the firing domain of S_{parent} with a set of additional constraints imposing $\tau(t_o)$ to be not longer than any transition $\tau(t_i)$ enabled in S_{parent}:

Clause 3.2. *Let t_o be a firable transition in class S_{Parent}, with firing domain D_{parent}*

$$D_{parent} = \begin{cases} \tau(t_i) - \tau(t_j) \leq b_{ij} \\ \forall t_i \in T(m_{parent}) \cup \{t_*\} \qquad t_i \neq t_j. \end{cases} \tag{13}$$

Let $D^{t_o}_{parent}$ be the restricted firing domain which augments the firing domain D_{parent} with a set of additional constraints imposing $\tau(t_o)$ to be not longer than any transition $\tau(t_i)$ enabled in S_{parent}:

$$D^{t_o}_{parent} = \begin{cases} \tau(t_i) - \tau(t_j) \leq b_{ij} \\ \tau(t_o) - \tau(t_j) \leq \min\{0, b_{oj}\} \\ \\ \forall t_i, t_j \in T(m_{parent}) \cup \{t_*\} \quad t_i \neq t_j. \end{cases} \tag{14}$$

And, let $D^{t_o}_{parent}$ be represented in normal form as:

$$D^{t_o}_{parent} = \begin{cases} \tau(t_i) - \tau(t_j) \leq B_{ij} \\ \forall t_i, t_j \in T(m_{parent}) \cup \{t_*\} \; with \; t_i \neq t_j. \end{cases} \tag{15}$$

The normal form of the firing domain of the class S_{child} reached from S_{parent} through the firing of t_o is:

$$D_{child} = \begin{cases} \tau(t_i) - \tau(t_*) & \leq & B_{io} \\ \tau(t_*) - \tau(t_i) & \leq & B_{oi} \\ \tau(t_i) - \tau(t_j) & \leq & B_{ij} \\ \tau(t_k) - \tau(t_*) & \leq & LFT^s(t_k) \\ \tau(t_*) - \tau(t_k) & \leq & -EFT^s(t_k) \\ \tau(t_k) - \tau(t_i) & \leq & LFT^s(t_k) + B_{oi} \\ \tau(t_i) - \tau(t_k) & \leq & B_{io} - EFT^s(t_k) \\ \tau(t_k) - \tau(t_h) & \leq & LFT^s(t_k) - EFT^s(t_h) \\ \tau(t_h) - \tau(t_k) & \leq & LFT^s(t_h) - EFT^s(t_k) \\ \\ \forall t_i, t_j \in T(pers_{child}) \\ \forall t_k, t_h \in T(new_{child}), \end{cases} \tag{16}$$

where $T(pers_{child})$ and $T(new_{child})$ denote the sets of transitions that are persistent and newly enabled in S_{child}, respectively.

With respect to the treatment of [9], [8], Clause 3.1 permits one to check the existence of an outgoing arc for t_o without resolving the set of difference constraints of the firing domain. Besides, Clause 3.2 expresses the normal representation of the child class D_{child} as a close form of the normal representation of the restricted firing domain $D^{t_o}_{parent}$. This enables the reduction of enumeration complexity expounded in the next section and opens the way to the analytical treatment of timing profiles derived in Section 3.4.

Clauses 3.1 and 3.2 can be embedded within a conventional graph enumeration algorithm to construct the reachability relation between the state classes of a TPN model. The enumeration algorithm recursively applies Clause 3.1 to verify the existence of a child class, builds the normal form of the restricted parent class, and computes the normal form of the child class using Clause 3.2. Fig. 3 reports a simple example which helps the understanding of the two clauses.

Termination of the algorithm is generally not decidable. In [8], it is shown that termination can be ensured by the finiteness of the *untimed* graph which enumerates reachable markings without considering timing constraints. But, this sufficient condition is not applicable when finiteness of the reachability set is ensured by timing constraints themselves. This is indeed the majority of significant cases. In [10], a modular approach is proposed in which the boundedness of the graph of a composed model is reduced to the boundedness of its components, thus restraining termination undecidability to low-level (smaller) modules. In a different approach, a finite and limited number of reachable state classes can be ensured by restricting the analysis to a *local* neighborhood of the starting state class, i.e., by considering only those classes that are reachable within a predefined maximum time interval [25]. In this case, since the number of successors for each state class is limited by the number of transitions in the net, termination is ensured provided that firing intervals prevent zero-time loops.

3.3 Reducing Enumeration Complexity

In case of finite enumeration, the dominating computational complexity stems from the normalization of restricted firing domains. Using the standard Floyd and Warshall algorithm, this results in a total complexity $O(|G||T|^3)$, where $|G|$ is the final number of nodes in the graph and $|T|$ is the maximum number of transitions that can be enabled by any reachable marking.

This complexity can be reduced by exploiting the peculiar conditions in which normalization is performed during the enumeration process. According to (14), $D^{t_o}_{parent}$ is derived from the normal form of D_{parent} by perturbating only the coefficients related to $\tau(t_o)$; this permits to reduce the complexity of Floyd-Warshall normalization by skipping the first $N-1$ iterations of the main loop on index k and by directly computing B_{ij} in close form:

Lemma 3.1. *If D_m is in normal form, in the execution of the Floyd-Warshall algorithm on $D^{t_o}_m$, for any $k > 0$*

$$b^{k-1}_{ij} = \begin{cases} b_{ij} & \text{if } i \neq 0 \\ \min\limits_{l \geq k}\{b_{0j}, 0, b_{lj}\} & \text{else.} \end{cases}$$

Proof. Demonstration runs by induction on index k.

For $k = N$, the statement is a direct consequence of (14) and of the initialization step of the Floyd-Warshall algorithm, which yields

$$b^{N-1}_{ij} = \begin{cases} b_{ij} & \text{if } i \neq 0 \\ \min\{b_{0j}, 0\} & \text{else.} \end{cases}$$

If the statement is true for $k = h$, the Floyd-Warshall algorithm computes b^{h-1}_{ij} as

$$b^{h-1}_{ij} = \min\{b^h_{ij}, b^h_{ih} + b^h_{hj}\}.$$

Two different cases are possible depending on index i: if $i \neq 0$, then b^{h-1}_{ij} can be rewritten as

$$b^{h-1}_{ij} = \min\{b_{ij}, b_{ih} + b_{hj}\},$$

which, using the normalization condition of (11), yields

$$b^{h-1}_{ij} = b_{ij}$$

if $i = 0$, then b^{h-1}_{0j} can be rewritten as

$$\begin{aligned} b^{h-1}_{0j} &= \min_{l \geq h+1}\{b_{0j}, 0, b_{lj}, \min_{m \geq h+1}\{b_{0h}, 0, b_{mh}\} + b_{hj}\} \\ &= \min_{l,m \geq h+1}\{b_{0j}, 0, b_{lj}, b_{0h} + b_{hj}, b_{hj}, b_{mh} + b_{hj}\}, \end{aligned}$$

which, using the normalization condition and collecting similar terms, yields

$$b^{h-1}_{0j} = \min_{l \geq h}\{b_{0j}, 0, b_{lj}\}. \qquad \square$$

Lemma 3.1 permits one to derive the normal representation of $D^{t_o}_{parent}$ from the normal representation of D_{parent} with $O(|T(m)|^2)$ computations. This improves the current practice of enumerative techniques for time-dependent computer aided verification by reducing the dominating complexity of enumeration from $O(|G||T|^3)$ to $O(|G||T|^2)$. It also permits to maintain a close form relation between coefficients b_{ij} of the normal form of D_{parent} and coefficients B_{ij} of the normal form of D^{to}_{parent}:

$$B_{ij} = \begin{cases} \min\{b_{ij}, b_{io}, b_{io} + \min\limits_{l \neq 0}\{b_{lj}\}\} & \text{if } i \neq 0 \\ \min\limits_{\forall l}\{b_{lj}, 0\} & \text{else.} \end{cases} \tag{17}$$

This enables the analytical derivations in reachability and timeliness analysis of state classes that will be expounded in Section 3.4.

3.4 Estimating Timing Profiles

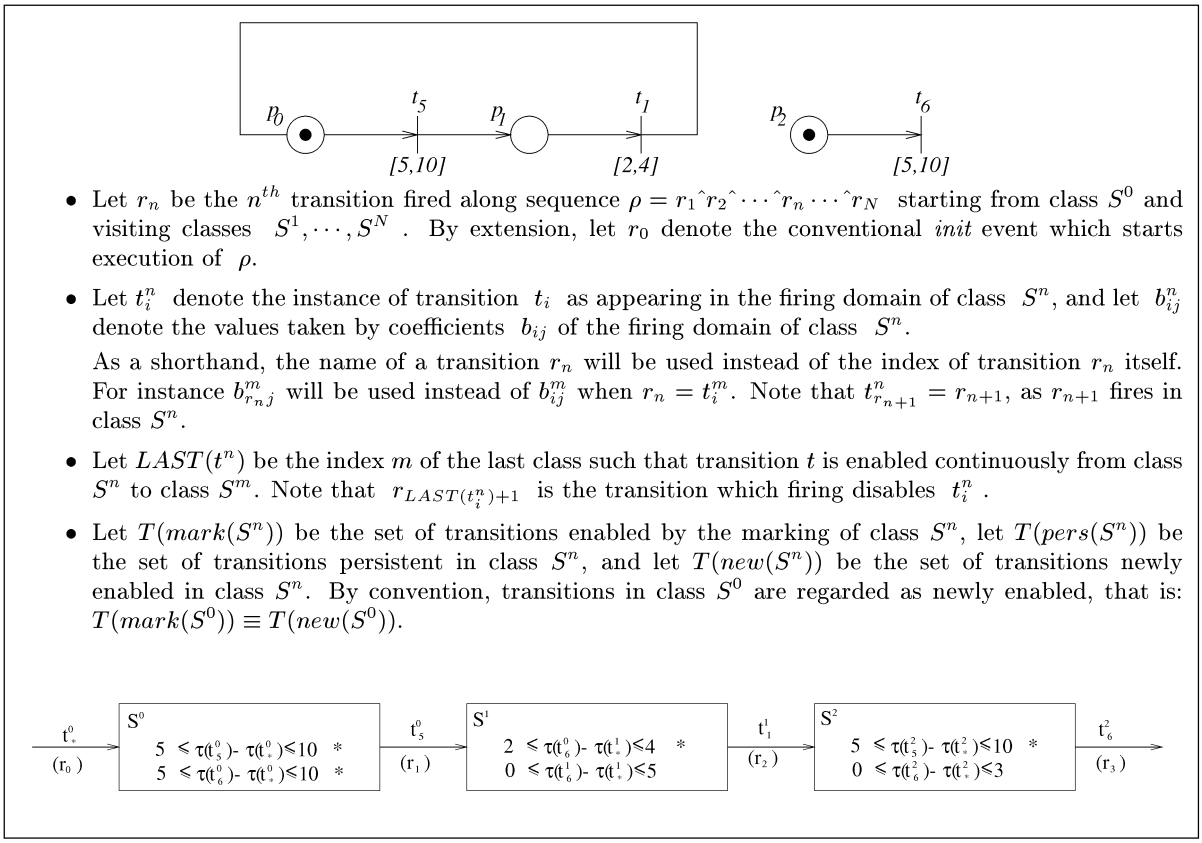
The graph of reachable state classes supports the conventional analysis of speed-independent properties of the model, such as reachability, deadlock-freedom, state liveness, and qualitative sequencing of execution traces. This analysis may account for sequencing limitations deriving from timed behavior, but it is not suited to evaluate timing properties of execution traces through the classes of the state graph. Due to the dense nature of timing spaces collected within classes, each such trace may be executed with a dense variety of timings. In the rest of this section, we introduce an original estimation algorithm which computes a profile for this variety. By developing on the close form of Clauses 3.1 and 3.2, the algorithm determines the firing times of all the transitions fired along a given sequential trace as unknown values of a *global* set of linear inequalities collecting constraints appearing in the classes visited by the trace itself. Since the global set turns out to have the same form as that of firing domains, the evaluation of the timing profile is further reduced to the normalization algorithm mentioned in Section 3.1.

Formulation of the estimation algorithm requires preliminary introduction of some notational conventions to distinguish instances of the same transition enabled in different classes. To this end, S^n denotes the nth class visited by the trace and transitions and coefficients, referred to S^n, are superscripted by n.; in addition, r_n denotes the index of the transition which fires to enter class S^n. The complete description of notational conventions is reported in Fig. 4.

The global set G_ρ, constraining the timing profile of a trace ρ, can be constructed by collecting all the inequalities appearing in the firing domains visited along the trace itself, together with additional gluing constraints capturing the sequencing by which the domains are encountered:

$$G_\rho = \begin{cases} D_n & (a) \\ \tau(t^n_{r_{n+1}}) - \tau(t^n_i) \leq 0 & (b) \\ \tau(t^n_{r_{n+1}}) = \tau(t^{n+1}_*) & (c) \\ \\ \forall n \in [0, N-1] \\ \forall t^n_i \in T(mark(S^n)), \end{cases} \tag{18}$$

where (18a) are the firing constraints of class S^n, (18b) is the sequencing constraint imposing $t^n_{r_{n+1}}$ be the outcoming transition of S^n, and (18c) is the synchronization imposing the outcoming time of S^n be equal to the entering time of S^{n+1} (i.e., the time $\tau(t^{n+1}_*)$ of the ground reference of class S^{n+1}).



- Let r_n be the n^{th} transition fired along sequence $\rho = r_1 \hat{} r_2 \hat{} \cdots \hat{} r_n \cdots \hat{} r_N$ starting from class S^0 and visiting classes $S^1, \cdots, S^N$. By extension, let r_0 denote the conventional *init* event which starts execution of ρ.
- Let t_i^n denote the instance of transition t_i as appearing in the firing domain of class S^n, and let b_{ij}^n denote the values taken by coefficients b_{ij} of the firing domain of class S^n.

  As a shorthand, the name of a transition r_n will be used instead of the index of transition r_n itself. For instance $b_{r_n j}^m$ will be used instead of b_{ij}^m when $r_n = t_i^m$. Note that $t_{r_{n+1}}^n = r_{n+1}$, as r_{n+1} fires in class S^n.
- Let $LAST(t^n)$ be the index m of the last class such that transition t is enabled continuously from class S^n to class S^m. Note that $r_{LAST(t_i^n)+1}$ is the transition which firing disables t_i^n.
- Let $T(mark(S^n))$ be the set of transitions enabled by the marking of class S^n, let $T(pers(S^n))$ be the set of transitions persistent in class S^n, and let $T(new(S^n))$ be the set of transitions newly enabled in class S^n. By convention, transitions in class S^0 are regarded as newly enabled, that is: $T(mark(S^0)) \equiv T(new(S^0))$.

Fig. 4. Notational conventions for the algorithm estimating the profile of an execution trace. The bottom picture exemplifies conventions in the case of a trace of the net depicted in the upper picture: The trace runs through three classes that are denoted as S^o, S^1, and S^2. The fictitious init event t_*^0 entering class S^0, is denoted as r_o; in a similar manner, transitions t_5^0, t_1^1, and t_6^2 enter classes S^1, S^2, and S^3 and, thus, are denoted as r_1, r_2, and r_3, respectively. Transitions are superscripted by the index of the class in which they are enabled. Persistent transitions appearing throughout subsequent classes are referred to with multiple superscripts: t_6^2 and t_6^1 denote the same instance of transition as t_6^0, but they are referred to different classes, so that $\tau(t_6^0)$ and $\tau(t_6^1)$ denote the time to fire of t_6 as observed in classes S^0 and S^1, respectively. Transition t_6 is enabled with continuity from class S^0 to class S^2, and it will be disabled after the exit of S^2: according to this, $LAST(t_6^0) = LAST(t_6^1) = LAST(t_6^2) = 2$. The transition that disables t_6^0 is $r_{LAST(t_6^0)+1} = r_3$. Constraints involving newly enabled transitions are marked by an asterisk.

3.5 Managing Complexity

In principle, since G_ρ has the same form as that of firing domains, the timing profile of the trace can be made explicit by reducing G_ρ in normal form. However, before tackling this computation, the complexity for constructing and normalizing the system can be reduced by eliminating constraints and unknown values that are dependent or not relevant for the understanding of the trace profile.

The number of inequalities appearing in system G_ρ can be reduced by eliminating constraints which refer to persistent transitions appearing throughout subsequent classes:

- If transition t_i^n is persistent in class S^n ($n > 0$), then all the constraints which involve t_i^n in the firing domain D_n are implied by the constraints in D_{n-1} and by the sequencing constraints $\tau(t_{r_n}^{n-1}) - \tau(t_i^{n-1}) \leq 0$.
- If transition t_i^n is newly enabled in class S^n and persistent until class $S^{LAST(t_i^n)}$, then all the sequencing constraints of type (18b) which involve t_i^m with $m = n \cdots LAST(t_i^n)$ are implied by the constraint which imposes that t_i^n does not fire before leaving class $S^{LAST(t_i^n)}$ (which happens at the firing of $r_{LAST(t_i^n)+1}$):

$$\tau(r_{LAST(t_i^n)+1}) - \tau(t_i^n) \leq 0.$$

Note that, if t_i^n is eventually fired as the jth event of the trace, i.e., if $t_i^n = r_j$, then $r_{LAST(t_i^n)+1} = r_{j-1}$ and the equation becomes:

$$\tau(r_{j-1}) - \tau(r_j) \leq 0.$$

Demonstration, which is reported in [27], relies on the close form of Clause 3.2.

By inductive composition of the two simplifications and by eliminating the unknown value $\tau(t_*^{n+1})$, the global set $G_\rho$can be rewritten as:

$$G_\rho = \begin{cases} \tau(t_i^0) - \tau(t_j^0) & \leq & b_{ij}^0 & (i) \\ \tau(t_h^n) - \tau(r_n) & \leq & b_{h*}^n & (ii) \\ \tau(r_n) - \tau(t_h^n) & \leq & b_{*h}^n & (iii) \\ \tau(r_{LAST(t_h^n)+1}) - \tau(t_h^n) & \leq & 0 & (iv) \\ \\ \forall n \in [0, N-1] & & & \\ \forall t_i^0, t_j^0 \in T(mark(S^0)) \quad \forall t_h^n \in T(new(S^n)). & & & \end{cases} \tag{19}$$

Note that (19) contains an unknown value for each newly enabled transition found along the trace. This could result in a complexity of normalization $O((|T|N)^3)$, N being the length of the trace and $|T|$ being the maximum number of

newly enabled transitions within a class. This number can be reduced by eliminating the firing time of any transition which is enabled along the trace but gets disabled before firing, thus having no physical counterpart in the model execution.

Elimination of such unknown values can be formulated as a *projection* problem [18]:

Definition 3.3. *Given a constraint set D_N among unknown values $< \tau_1, \cdots, \tau_{N-1}, \tau_N >$*

$$D_N = \begin{cases} \tau_i - \tau_j \leq b_{ij} \\ \forall i, j \in [1, N], i \neq j, \end{cases}$$

the projection of D_N on the unknown values $< \tau_1, \cdots, \tau_{N-1} >$ is any system D_{N-1} such that: 1) if $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1}, \bar{\tau}_N >$ is a solution for D_N, then $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1} >$ is a solution for D_{N-1}; and 2) if $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1} >$ is a solution for D_{N-1}, then there exists $\bar{\tau}_N$ such that $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1}, \bar{\tau_N} >$ is a solution for D_N.

The following lemma gives a close form to compute the projection in the simplification of G_ρ:

Lemma 3.2. *The set of constraints among the unknown values $< \tau_1, \cdots, \tau_{N-1} >$*

$$D_{N-1} = \begin{cases} \tau_i - \tau_j \leq \min\{b_{ij}, b_{iN} + b_{Nj}\} \\ \forall i, j \in [1, N-1], i \neq j. \end{cases}$$

is a projection for system D_N of Definition 3.3.

Proof. 1) If $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1}, \bar{\tau}_N >$ is a solution for D_N, then

$$\begin{cases} \bar{\tau}_i - \bar{\tau}_j & \leq & b_{ij} \\ \bar{\tau}_i - \bar{\tau}_N + \bar{\tau}_N - \bar{\tau}_j & \leq & b_{iN} + b_{Nj} \\ \forall i, j \in [1, N-1], i \neq j. \end{cases} \quad (20)$$

which implies that $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1} >$ is a solution for D_{N-1}.

2) If $< \bar{\tau}_1, \cdots, \bar{\tau}_{N-1} >$ is a solution for D_{N-1}, then the substitution of $\bar{\tau}_i$ in D_N yields a set of inequalities where τ_N is the only unknown value left:

$$\begin{cases} \bar{\tau}_i - \bar{\tau}_j & \leq & b_{ij} & (i) \\ \tau_N - \bar{\tau}_i & \leq & b_{Ni} & (ii) \\ \bar{\tau}_i - \tau_N & \leq & b_{iN} & (iii) \\ \forall i, j \in [1, N-1], i \neq j. \end{cases} \quad (21)$$

Inequality (21i) is satisfied by hypothesis, while (21ii) and (21iii) can be rewritten as

$$-b_{iN} \leq \tau_N - \bar{\tau}_i \leq b_{Ni}, \quad (22)$$

which accepts a solution $\bar{\tau}_N$ iff

$$\max_{i \in [1, N-1]} \{-b_{iN} + \bar{\tau}_i\} \leq \min_{i \in [1, N-1]} \{b_{Ni} + \bar{\tau}_i\}. \quad (23)$$

Ab absurdo, if there exist i and j such that

$$-b_{iN} + \bar{\tau}_i > b_{Nj} + \bar{\tau}_j, \quad (24)$$

then

$$\bar{\tau}_i - \bar{\tau}_j > b_{Nj} + b_{iN} \quad (25)$$

which contradicts the hypothesis. □

Note that D_{N-1} is derived from D_N by eliminating all the inequalities involving τ_N and by adding the constraint $\tau_i - \tau_j \leq b_{iN} + b_{Nj}$ which represents the *propagation* of the constraints on τ_N to the other unknown values $\tau_1 \cdots \tau_{N-1}$.

Lemma 3.2 enables elimination of every unknown value corresponding to a transition which is enabled but not fired along the sequence. In this elimination, t_w^n will denote any transition newly enabled in S^n but not fired, and t_x^0 will denote any transition enabled in S^0 but not fired:

Theorem 3.1. *The projection of G_ρ on the unknown values representing the firing time of the transition actually fired along sequence ρ can be derived from G_ρ by eliminating every constraint which involves any $\tau(t_w^n)$ or $\tau(t_x^0)$, and by adding the propagation constraints*

$$\begin{cases} \tau(r_{LAST(t_w^n)+1}) - \tau(r_n) & \leq & b_{w*}^n \\ \tau(r_{LAST(t_x^0)+1}) - \tau(r_i) & \leq & b_{xr_i}^0 \\ \forall n \in [0, N-1] \\ \forall r_i \in T(mark(S^0)) \\ \forall t_x^0 \in T(mark(S^0)) \text{ such that } t_x^0 \text{ is not fired in the trace} \\ \forall t_w^n \in T(new(S^n)) \text{ such that } t_w^n \text{ is not fired in the trace.} \end{cases}$$

Proof. According to Lemma 3.2, $\tau(t_w^n)$ can be eliminated from G_s provided that the constraint relating to it are propagated to the other unknown values. From (19), the only finite bounds that can be propagated by $\tau(t_w^n)$ are

$$\begin{cases} \tau(t_w^n) - \tau(r_n) & \leq & b_{w*}^n \\ \tau(r_n) - \tau(t_w^n) & \leq & b_{*w}^n \\ \tau(r_{LAST(t_w^n)+1}) - \tau(t_w^n) & \leq & 0, \end{cases} \quad (26)$$

so the only unknown values in the projection that can be constrained by $\tau(t_w^n)$ are $\tau(r_n)$ and $\tau(r_{LAST(t_w^n)+1})$. The bound between these unknown values can be written as

$$\tau(r_{LAST(t_w^n)+1}) - \tau(r_n) = \tau(r_{LAST(t_w^n)+1}) - \tau(t_w^n) + \tau(t_w^n) - \tau(r_n), \quad (27)$$

which, from (26), yields

$$-b_{*w}^n \leq \tau(r_{LAST(t_w^n)+1}) - \tau(r_n) \leq 0 + b_{w*}^n. \quad (28)$$

Bounds propagated by constraints involving a transition $\tau(t_x^0)$ are computed in a similar manner. □

After the elimination of transitions that are not fired, the total number of unknown values appearing in the system is equal to the number N of transitions which are actually fired along the sequence, and the set G_ρ of (19) is reduced to the final form:

$$G_\rho = \begin{cases} \tau(r_i) - \tau(r_j) & \leq & b_{r_i r_j}^0 & (i) \\ \tau(r_h) - \tau(r_n) & \leq & b_{r_h *}^n & (ii) \\ \tau(r_n) - \tau(r_h) & \leq & b_{*r_h}^n & (iii) \\ \tau(r_{h-1}) - \tau(r_h) & \leq & 0 & (iv) \\ \tau(r_{LAST(t_w^n)+1}) - \tau(r_n) & \leq & b_{w*}^n & (v) \\ \tau(r_{LAST(t_x^0)+1}) - \tau(r_i) & \leq & b_{xr_i}^0 & (vi) \\ \forall n = 0, N-1 \\ \forall r_i, r_j \in T(mark(S^0)) \qquad \forall r_h \in T(new(S^n)) \\ \forall t_x^0 \in T(mark(S^0)) \text{ such that } t_x^0 \text{ is not fired in the trace } \rho \\ \forall t_w^n \in T(new(S^n)) \text{ such that } t_w^n \text{ is not fired in the trace } \rho. \end{cases} \quad (29)$$

Since G_ρ has the same form as that of firing domains, it can be solved by reducing it in normal form as described in Section 3.1. According to this, a tight timing profile on the firing times of a trace with length N is computed with $O(N^3)$ operations.

3.6 An Example

For the sake of comprehension, the steps for the construction of the global system expounded in the previous section are exemplified with reference to the case of the trace ρ of Fig. 4.

Equation (18) is obtained by collecting the inequalities appearing in the firing domains of classes S^0, S^1, S^2, along with the gluing constraints expressing the sequencing of the trace:

$$G_\rho = \begin{cases} 5 \leq & \tau(t_5^0) - \tau(t_*^0) & \leq 10 & (a) \\ 5 \leq & \tau(t_6^0) - \tau(t_*^0) & \leq 10 & (a) \\ 2 \leq & \tau(t_1^1) - \tau(t_*^1) & \leq 4 & (a) \\ 0 \leq & \tau(t_6^1) - \tau(t_*^1) & \leq 5 & (a) \\ 5 \leq & \tau(t_5^2) - \tau(t_*^2) & \leq 10 & (a) \\ 0 \leq & \tau(t_6^2) - \tau(t_*^2) & \leq 3 & (a) \\ & & & \\ & \tau(t_{r_1}^0) - \tau(t_5^0) & \leq 0 & (b) \\ & \tau(t_{r_1}^0) - \tau(t_6^0) & \leq 0 & (b) \\ & \tau(t_{r_2}^1) - \tau(t_1^1) & \leq 0 & (b) \\ & \tau(t_{r_2}^1) - \tau(t_6^1) & \leq 0 & (b) \\ & \tau(t_{r_3}^2) - \tau(t_5^2) & \leq 0 & (b) \\ & \tau(t_{r_3}^2) - \tau(t_6^2) & \leq 0 & (b) \\ & & & \\ & \tau(t_{r_1}^0) = \tau(t_*^1) & & (c) \\ & \tau(t_{r_2}^1) = \tau(t_*^2) & & (c) \\ & \tau(t_{r_3}^2) = \tau(t_*^3) & & (c). \end{cases}$$

Equation (19) is obtained by replacing the unknown values $\tau(t_*^n)$ and by eliminating the constraints referring to the persistent transitions t_6^1 and t_6^2:

$$G_\rho = \begin{cases} 5 \leq & \tau(t_5^0) - \tau(r_0) & \leq 10 \\ 5 \leq & \tau(t_6^0) - \tau(r_0) & \leq 10 \\ 2 \leq & \tau(t_1^1) - \tau(r_1) & \leq 4 \\ 5 \leq & \tau(t_5^2) - \tau(r_2) & \leq 10 \\ 5 \leq & \tau(t_5^2) - \tau(r_2) & \leq 10 \\ & & \\ & \tau(r_3) - \tau(t_5^2) & \leq 0 \\ & \tau(r_0) - \tau(t_5^0) & \leq 0 \\ & \tau(r_2) - \tau(t_6^0) & \leq 0 \\ & \tau(r_1) - \tau(t_1^1) & \leq 0 \\ & \tau(r_3) - \tau(t_5^2) & \leq 0. \end{cases}$$

The final representation of (29) is obtained by eliminating the unknown value $\tau(t_5^2)$associated with transition t_5^2 which is not fired along ρ:

$$G_\rho = \begin{cases} 5 \leq & \tau(r_1) - \tau(r_0) & \leq 10 \\ 5 \leq & \tau(r_3) - \tau(r_0) & \leq 10 \\ 2 \leq & \tau(r_2) - \tau(r_1) & \leq 4 \\ & & \\ & \tau(r_0) - \tau(r_1) & \leq 0 \\ & \tau(r_2) - \tau(r_3) & \leq 0 \\ & \tau(r_1) - \tau(r_2) & \leq 0 \\ & \tau(r_3) - \tau(r_2) & \leq 10. \end{cases}$$

The final normal representation of the profile is:

$$G_\rho = \begin{cases} 5 & \leq & \tau(r_1) - \tau(r_0) & \leq & 8 \\ 7 & \leq & \tau(r_2) - \tau(r_0) & \leq & 10 \\ 7 & \leq & \tau(r_3) - \tau(r_0) & \leq & 10 \\ 2 & \leq & \tau(r_2) - \tau(r_1) & \leq & 4 \\ 0 & \leq & \tau(r_3) - \tau(r_1) & \leq & 5 \\ 0 & \leq & \tau(r_3) - \tau(r_2) & \leq & 3. \end{cases}$$

which comprises a tight execution profile making explicit the extreme values for the set of feasible timings of the trace ρ:

- The extreme values constraining the firing time of the last transition of the sequence represent the minimum and maximum possible durations for the execution of the trace; in our example, the last transition is $r_3 = t_6^2$ and, thus, the inequality $7 \leq \tau(r_3) - \tau(r_0) \leq 10$ shows that the trace has a minimum and maximum possible duration equal to 7 and 10, respectively.
- The earliest and latest possible execution time for all the transitions along the sequence can be determined as the extreme values constraining their firing times. For instance, the inequality $5 \leq \tau(r_1) - \tau(r_0) \leq 8$ shows that transition $r_1 = t_5^0$ is executed in a time which is in the interval $[5, 8]$. Note that this means that, while any time to fire in the interval $[5, 10]$ complies with the constraints in the firing domain of state class S^0, only a time to fire in the interval $[5, 8]$ makes the execution of trace ρ possible. In this case, the timing profile makes explicit the dependency among transition sequencing and firing times.
- The minimum and maximum delay between the execution of distinct (not necessarily subsequent) transitions in the trace can be determined by inspection of the extreme values of the difference between their corresponding firing times. For instance, the inequality $0 \leq \tau(r_3) - \tau(r_2) \leq 3$ shows that transition $r_3 = t_6^2$ is executed within three time units after the execution of $r_2 = t_1$.

Once more, it is worth recalling that all these estimates are tight in that: 1) no feasible executions of the trace may exceed them; 2) any weaker estimate is exceeded by some feasible execution of the trace.

3.7 Trace Composition

Sections 3.4 and 3.5 provide means to compute a tight time profile for a *single* execution trace in a system model. The complexity involved in the repetition of this analysis on multiple traces (e.g., in deriving a bound for the time spent along any possible trace between a pair of states) can be managed by decomposing traces into sequences of events, that we will call *tracelets*, which may appear within multiple traces. The profiles of individual tracelets can be computed in isolation and reused to compute the execution profile of each composed trace in which they belong.

To expound the details, let $\rho = \bar{\rho} \rightarrow \hat{\rho}$ be a trace made up of two tracelets $\bar{\rho}$ and $\hat{\rho}$, linked through a common class S. Let $\bar{r}_n$ and $\hat{r}_n$ denote the nth transition fired along $\bar{\rho}$ and $\hat{\rho}$, respectively, and let $\bar{N}$ and $\hat{N}$ denote the number of transitions in the two tracelets. Finally, let $\bar{b}_{ij}^n$ denote the

coefficient appearing in the inequality $\tau(t_i^n) - \tau(t_j^n) \leq \bar{b}_{ij}^n$ included in the nth class $\bar{S}^n$ of $\bar{\rho}$.

The comparison of the global systems G_ρ, $G_{\bar{\rho}}$, and $G_{\hat{\rho}}$ (written according to (29) for the three traces ρ, $\bar{\rho}$ and $\hat{\rho}$) shows that G_ρ can be expressed as the conjunction of $G_{\bar{\rho}}$ and $G_{\hat{\rho}}$ with the addition of a link constraint imposing the last event of $\bar{\rho}$ to be the initial event of $\hat{\rho}$, and with the addition of a set of inequalities $\Gamma_{\bar{\rho},\hat{\rho}}$:

$$G_\rho = \begin{cases} G_{\bar{\rho}} \\ G_{\hat{\rho}} \\ \tau(\bar{r}_{\bar{N}}) = \tau(\hat{r}_0) \\ \Gamma_{\bar{\rho},\hat{\rho}}. \end{cases} \tag{30}$$

The additional set $\Gamma_{\bar{\rho},\hat{\rho}}$stems from those transitions that are persistent at the firing of the last transition of $\bar{\rho}$ (i.e., transitions that are persistent in the linking class S). These yield different constraints whether they are enabled since the first class $\bar{S}^0$ of $\bar{\rho}$ or since a generic class $\bar{S}^n$ within $\bar{\rho}$ itself and whether they are fired or not in the execution of $\hat{\rho}$:

- For any transition t_i, which is enabled in $\bar{S}^0$ and fired as $\hat{r}_I$ in $\hat{\rho}$, and for any transition $\bar{r}_j$ enabled in $\bar{S}^0$ and fired in $\bar{\rho}$, the set $\Gamma_{\bar{\rho},\hat{\rho}}$ includes the constraint:
$$\tau(\hat{r}_I) - \tau(\bar{r}_j) \leq \bar{b}^0_{i\bar{r}_j}.$$
- For any transition t_x enabled in $\bar{S}^0$ and disabled in $\hat{\rho}$ by the firing of transition $\hat{r}_K$ and for any transition $\bar{r}_j$ enabled in $\bar{S}^0$ and fired in $\bar{\rho}$, the set $\Gamma_{\bar{\rho},\hat{\rho}}$includes the constraint:
$$\tau(\hat{r}_K) - \tau(\bar{r}_j) \leq \bar{b}^0_{x\bar{r}_j}.$$
- For any transition t_h newly enabled in $\bar{S}^n$, persistent until the end of $\bar{\rho}$, and fired as $\hat{r}_H$ in $\hat{\rho}$, the set $\Gamma_{\bar{\rho},\hat{\rho}}$ includes the constraints:
$$\tau(\hat{r}_H) - \tau(\bar{r}_n) \leq \bar{b}^n_{h*},$$
$$\tau(\bar{r}_n) - \tau(\hat{r}_H) \leq \bar{b}^n_{*h}.$$
- For any transition t_w newly enabled in $\bar{S}^n$ and disabled in $\hat{\rho}$ by the firing of transition $\hat{r}_K$, the set $\Gamma_{\bar{\rho},\hat{\rho}}$ includes the constraint:
$$\tau(\hat{r}_K) - \tau(\bar{r}_n) \leq \bar{b}^n_{w*}.$$

In conclusion, $\Gamma_{\bar{\rho},\hat{\rho}}$ includes a maximum of $2 \cdot |T|$, where $|T|$ is the number of transitions that are persistent through the firing of the last transition of the first tracelet $\bar{\rho}$.

If $G_{\bar{\rho}}$ and $G_{\hat{\rho}}$ are in normal form since they share a single unknown value ($\tau(\bar{r}_{\bar{N}}) \equiv \tau(\hat{r}_0)$), their conjunction is still a normal form. To exploit this initial condition, the normal form of G_ρ of (30) can be incrementally derived by adding, one by one, the additional constraints of $\Gamma_{\bar{\rho},\hat{\rho}}$ and by recovering the normal form after each perturbation due to the addition of a constraint. As suggested in [1], renormalization after such perturbation can be accomplished with complexity $O((\bar{N}+\bar{N})^2)$. The total complexity to add all the $2|T|$ additional constraints is thus equal to $O(2|T|(\bar{N}+\bar{N})^2)$.

In conclusion, if $|T|$ is the number of transitions that are persistent through the firing of the last transition of $\bar{\rho}$, the overall complexity to derive the profile for $G_{\bar{\rho}\to\hat{\rho}}$ is equal to $O(\bar{N}^2) + O(\hat{N}^2)$ necessary to separately derive the normal form for $G_{\bar{\rho}}$ and $G_{\hat{\rho}}$, plus the additional complexity $O(2|T|(\bar{N}+\bar{N})^2)$ needed to compose them. If $\bar{N}$ and $\hat{N}$ are not small numbers (i.e., if $\bar{\rho}$ and $\hat{\rho}$ are long traces), this reduces the complexity with respect to a direct application of the Floyd-Warshall algorithm to the overall set of inequalities (which would result in a complexity $O(2|T|(\bar{N}+\bar{N})^2)$). In addition, if $\bar{\rho}$ and $\hat{\rho}$ appear in multiple traces, their profile can be computed only once.

4 Static Verification and Dynamic Steering of Real-Time Systems

The methods described in the previous section provide the basis for a mixed static/dynamic approach to timeliness predictability. The approach is expounded in this section with reference to the problem of dynamic acceptance and guarantee within a hard real-time environment.

4.1 System Model

We address the case of a set of processes which generate tasks comprised of a sequence of computations, each characterized by an upper and a lower bound on the computation time and by a set of nonpreemptable required resources. Processes are constrained by a hard deadline made up of an earliest and latest completion time. Both periodic and aperiodic processes are considered. In the latter case, task generation is limited by a minimum intertime between the arrival of subsequent tasks of the same process. Processes are distinguished as either *mandatory* or *optional*. Tasks generated by mandatory processes must always be serviced. Whereas, tasks of optional processes can be accepted or rejected at their generation; however, if accepted, they must be serviced in time without causing any previously guaranteed task to fail.

For one such system model, correctness is faced with a twofold safeness and liveness requirement: The system must never miss the deadline of any accepted task, but it must also eventually accept as many as possible optional tasks. Joint satisfaction of both requirements must be achieved through the appropriate steering of acceptance/rejection decisions of optional tasks. These decisions are taken either statically or at runtime according to the principle *as static as possible, as dynamic as necessary*.

4.2 Static Analysis

Before runtime, the system is modeled as a Time Petri Net and its state space is enumerated according to the methods of Section 3.2. The state space is then analyzed to identify the set of *critical steering classes* in which the acceptance of an optional task *can* lead to a deadline miss.

To this end, traces of each process p are identified as the paths in the state space leading from the arrival event to the completion event of a task of p.

The timing profile of each such trace ϱ^p is then compared against the deadline of p itself. Each trace ϱ^p is classified as either *safe*, *unsafe*, or *failing* whether the untimely completion of process p along ϱ^p itself is impossible, possible, or necessary, respectively.

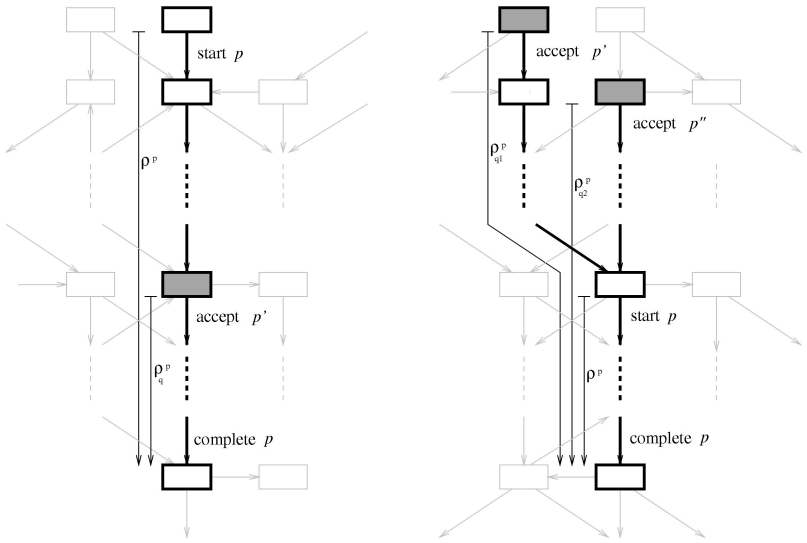


Fig. 5. Identification of the steering classes (shadowed) enabling avoidance of an execution trace ϱ^p. (a) If ϱ^p contains the acceptance of any process p', ϱ^p can be avoided by forcing the rejection of p'. (b) If ϱ^p does not contain any accept event, multiple steering classes are found through backward reachability analysis from the starting class of trace ϱ^p.

Operatively, the classification is derived by comparing the deadline interval $[\alpha^p, \beta^p]$ of process p against the set of solutions of the global set G_{ϱ^p} computed according to (29) for trace ϱ^p. If G_{ϱ^p} is represented in normal form as:

$$G_{\varrho^p} = \begin{cases} \tau(r_i) - \tau(r_j) \leq b_{ij} \\ \forall i, j \in [0, N] \;\; with\ N = length(\varrho^p)\ and\ i \neq j, \end{cases} \tag{31}$$

then ϱ^p is: *safe* iff $[-b_{0N}, b_{N0}] \subseteq [\alpha^p, \beta^p]$, *failing* iff $[-b_{0N}, b_{N0}] \cap [\alpha^p, \beta^p] = \emptyset$, and *unsafe* in the remaining case.

Each failing or unsafe trace is mapped to one or more critical steering classes representing the last control points where a rejection can avoid the execution of the trace. Specifically, the steering classes of a trace ϱ^p are identified by backward reachability analysis, as the classes where the last acceptance decision is taken before coming to the completion of p along some execution which covers ϱ^p itself (see Fig. 5).

The mapping of critical traces ϱ^p permits the association each steering class S with the set of unsafe or failing traces ρ_q^p originating in S itself and terminating with the completion of p along the tail of a failing or unsafe trace ϱ^p (see again Fig. 5). For each such trace ρ_q^p, the steering class S is annotated with the global set $G_{\rho_q^p}$ associated to ρ_q^p according to (29). This set, that we call *trace signature*, encodes the future behaviors, together with their enabling conditions, which may lead to a failure from some states included in the class S.

The above treatment does not explicitly addresses traces which may contain loops. In fact, in the specific application discussed in this section, static analysis requires to manage traces delimited within the start and completion event of a task which is comprised of a linear sequence of computations. Under these conditions, if λ would be a loop included within a trace ρ^p of a process p, then λ could not include any event of process p, i.e., a transition t_p belonging to process p should be persistent throughout the entire loop λ. In addition, for λ to be a loop in the graph of state classes, the coefficient b_{p*} in the constraint $\tau(t_p) - \tau(t_*) \leq b_{p*}$ should be the same at the beginning and the end of λ. However, according to Clause 3.2, this requires either that $b_{p*} = \infty$ or that the maximum time spent along the sequence of events in λ is equal to zero. The first case is not possible as all the computations within a task have a finite duration. The latter case is possible, but reflects an erroneous modeling abstraction yielding a so-called Zeno behavior in which time is not allowed to progress beyond a finite limit while an infinite number of events happen [4]. According to this, during the enumeration of traces of any process p, loops are detected (using conventional network flow algorithms [1]) and notified as errors to the designer, but the analysis is limited to traces which do not contain any loop.

Out of the specific scope of the application discussed in this paper, loops could occur in a more general tasking model that can be expressed using Time Petri Nets, or in the analysis of more complex traces delimited by any pair of initial and terminal events (not necessarily corresponding to the start and completion of a linear process). For instance, this might happen in the analysis of a protocol transaction in which a packet transmission is repeated until reception of an acknowledgment. In the analysis of such a trace, while loops can be easily detected, the enumeration of traces containing any possible combination of loops is not finite and enumeration of traces containing a maximum nonnull number of loops (e.g., completion time of the transaction in case of k retransmissions) is NP-hard. However, while deriving the individual profile of all such traces is nonviable, a bound on the maximum and minimum completion time for the overall set of traces can be obtained by separately deriving the profile of the basic trace without loops and of the individual loops that can be added to the trace: The minimum completion time for the overall set of traces is the minimum completion time for the basic trace

without loops; besides, if any loop in the trace has a maximum completion time higher than zero, then the maximum completion time is nonfinite, while it is equal to the maximum completion time of the basic trace without loops if all the loops have a maximum completion time equal to zero. Again, this latter case usually reflects an erroneous modeling abstraction.

4.3 Dynamic Steering

During runtime, a *steering monitor* tracks the evolution of the system through the annotated graph of state classes evaluating a *current dynamic state* which complements static predictions with the knowledge of the actual execution timing. On each event, the monitor moves the current state along the edges of the state class graph and shifts firing intervals of persistent transitions according to the actual time elapsed since the last event. In addition, the monitor also maintains the time spent from the beginning of the execution of each pending task.

When the system comes to an acceptance/rejection point, the steering monitor derives a decision from the *trace signatures* annotated before runtime on the *current class* to which the current state is mapped: Acceptance is forced if the current class is not associated with any failing or unsafe trace; conversely, rejection is forced if the current class is associated with at least one failing trace; in the remaining case in which the class is associated only with unsafe traces, the monitor must also resort to the runtime knowledge of the system state and check whether any unsafe trace ρ_q^p associated with the current class before runtime is actually feasible under the current determination of timing.

To expound the details of this check, let us consider the case that process p is pending in S^0 (i.e., the case of Fig. 5a). The case in which p is expected to start within ρ_q^p (i.e., the case of Fig. 5b) is dealt in a similar manner.

The knowledge of the actual time τ^p elapsed since the start of the last task of process p permits to exactly evaluate the time interval $\bar{I}^p$ within which p must complete:

$$\bar{I}^p = [\alpha^p - \tau^p, \beta^p - \tau^p]. \tag{32}$$

Besides, the knowledge of the current dynamic state also permits to refine the expected timing profile of trace ρ_q^p. In detail, if $G_{\varrho_q^p}$ is the trace signature annotated before runtime, then the runtime refined profile $\bar{G}_{\varrho_q^p}$ of ρ_q^p is obtained by restricting $G_{\varrho_q^p}$ with the additional set of dynamic constraints $\Gamma_{\varrho_q^p}$:

$$\bar{G}_{\varrho_q^p} = \begin{cases} G_{\varrho_q^p} \\ \\ \Gamma_{\varrho_q^p} \ \ where \ \ \Gamma_{r_q}^p = \begin{cases} \tau(r_i) - \tau(r_0) & \leq & \bar{b}_i \\ \tau(r_0) - \tau(r_i) & \leq & -\bar{a}_i \\ \tau(r_{LAST(t_w)+1}) - \tau(r_0) & \leq & \bar{b}_w, \end{cases} \end{cases} \tag{33}$$

where, according to the notation of (29): r_0 denotes the starting event of ρ_q^p, r_i stands for any transition which is enabled in the current state and is expected to fire along ρ_q^p, t_w is any transition which is enabled in class S^0 but will not fire along ρ_q^p (being diabled by the firing of transition $r_{LAST(t_w)}$), and $[\bar{a}_i, \bar{b}_i]$ is the dynamic firing interval of transition t_i when the current state is entered.

Normalization of $G_{\varrho_q^p}$ provides a tight bound for the expected time to the completion of the pending task of process p if the system runs through trace ρ_q^p. The comparison of this time against the dynamic deadline of (32) permits one to *exactly* decide whether trace ρ_q^p may lead to the untimely completion of process p. This supports optimal decision on the acceptance or rejection of the optional task.

4.4 Minimizing Dynamic Effort

Online deployment of the steering monitor requires that the acceptance test be optimized with respect to its runtime effort. The dominating component of this effort is the normalization of the global set of (33), which requires $O(N^3)$ assignments and comparisons, N standing for the number of transitions fired along trace ρ_q^p. This complexity can be largely reduced by anticipating before runtime the processing of (33) which does not require online knowledge.

During runtime, (33) provides a restriction of the trace signature $G_{\varrho_q^p}$ which permits one to derive a bound for the expected timing of *all* the events of the trace, though the checking procedure only relies on the bound on the expected time of the completion event. In so doing, we compute online a restriction and then a projection of the signature of the trace. In the following lemma, we show that the order of projection and restriction can be commuted. This permits one to compute the restriction on the projection, which can be anticipated before runtime. In doing so, runtime complexity is reduced by managing a projected, and thus smaller, set of unknown values.

Lemma 4.1. *Let D_N be a set of constraints among unknown values $< \tau_1, \cdots, \tau_{N-1}, \tau_N >$ and let D_{N-1} be the projection of D_N on the unknown values $< \tau_1, \cdots, \tau_{N-1} >$. The restriction of D_{N-1} with the additional constraint $\tau_2 - \tau_1 \leq b$ is equal to the projection of the restriction of D_N on the unknown values $< \tau_1, \cdots, \tau_{N-1} >$.*

Proof. Let D_N be

$$D_N = \begin{cases} \tau_i - \tau_j \leq b_{ij} \\ \forall i, j \in [1, N], i \neq j. \end{cases} \tag{34}$$

According to Lemma 3.2, D_{N-1} can be written as

$$D_{N-1} = \begin{cases} \tau_i - \tau_j \leq \min\{b_{ij}, b_{iN} + b_{Nj}\} \\ \forall i, j \in [1, N-1], i \neq j, \end{cases} \tag{35}$$

which, restricted with $\tau_2 - \tau_1 \leq b$, yields

$$D_{N-1}{}' = \begin{cases} \tau_i - \tau_j \leq \min\{b_{ij}, b_{iN} + b_{Nj}\} \\ \tau_2 - \tau_1 \leq \min\{b_{21}, b_{2N} + b_{N1}, b\} \\ \\ \forall i, \in [1, N-1], i \neq j, i \neq 2, j \neq 1. \end{cases} \tag{36}$$

On the other hand, the restriction of D_N can be written as

$$D_N{}' = \begin{cases} \tau_i - \tau_j \leq b_{ij} \\ \tau_2 - \tau_1 \leq \min\{b_{21}, b\} \\ \\ \forall i, j \in [1, N-1], i \neq j, i \neq 2, j \neq 1, \end{cases} \tag{37}$$

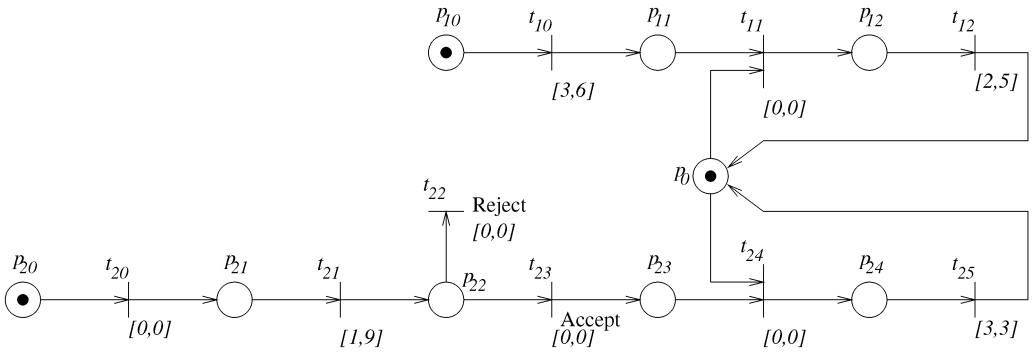


Fig. 6. A simple net modeling two one-shot processes.

which, projected on $< \tau_1, \cdots, \tau_{N-1} >$, yields

$$D_N'' = \begin{cases} \tau_i - \tau_j \leq \min\{b_{ij}, b_{iN} + b_{Nj}\} \\ \tau_2 - \tau_1 \leq \min\{b_{21}, b, b_{2N} + b_{N1}\} \\ \\ \forall i, j \in [1, N-1], i \neq j, i \neq 2, j \neq 1. \end{cases} \quad (38)$$

As $D_N'' = D_{N-1}'$, the lemma is proved. □

By recursive application of Lemma 4.1, the trace signature $G_{\varrho_q^p}$ in the right hand side of (33) can be replaced by the projection $\hat{G}_{\varrho_q^p}$ on the set of unknown values that are explicitly referenced in runtime restrictions or in the evaluation of the duration of the trace, i.e.,

- The starting time $\tau(r_0)$ of the trace ρ_q^p,
- The completion time $\tau(r_N)$ of the trace ρ_q^p,
- The firing time $\tau(r_i)$ of any transition r_i which is enabled in S^0 and expected to fire along ρ_q^p, and
- The firing time $\tau(r_{LAST(t_w)+1})$ of any transition $r_{LAST(t_w)+1}$ which is expected to disable any of the transitions t_w enabled in S^0 but not expected to fire along ρ_q^p.

The total number of unknown values appearing in (33) is, thus, reduced to a number which is not higher than $|M| + 2$, where $|M|$ stands for the number of transitions enabled in S^0.

The projection of $G_{\varrho_q^p}$ onto $\hat{G}_{\varrho_q^p}$ still requires $O(N^3)$ computations, but it can be entirely performed before runtime. During runtime, it is sufficient to restrict the projected signature $\hat{G}_{\varrho_q^p}$ with the additional constraints of (33) and normalize the resulting system, which can be done with $O((|M|+2)^3)$ computations. This complexity can be further reduced to $O((|M|+2)^2)$ by avoiding complete normalization and computing only the two coefficients which limit the difference between the starting and completion events of the trace.

4.5 An Example

The case of the system in Fig. 6 will help the comprehension. The system is comprised of two processes, that we name 10 and 20, modeled by the sequences $t_{10} \rightarrow t_{11} \rightarrow t_{12}$ and $t_{20} \rightarrow t_{21} \rightarrow (t_{22} \vee \rightarrow t_{23} \rightarrow t_{24} \rightarrow t_{25})$, respectively. Both the processes generate a single task, whose release is modeled by transitions t_{10} and t_{20}, respectively. The phasing is arranged so as to make t_{10} be fired after t_{20}, with a delay ranging between three and six time units. Both tasks include a computation, with worst case execution times taking values in the intervals $[2, 5]$ and $[3, 3]$, which require the exclusive resource modeled by place P_0. The task of process 10 is mandatory and we assume that it must be completed within seven time units after the firing of t_{10}. The task generated by process 20 is optional and can be accepted or rejected by choice on the firing of t_{23} or t_{22}. We assume that, if accepted, the task must be completed within 13 time units measured since the firing of t_{20}.

Fig. 7 reports a fragment of the graph of state classes, which includes three steering classes, S_3, S_7, and S_{13} (drawn in rounded rectangles), in which the system can control the execution by deciding on the acceptance (transition t_{23}) or rejection (transition t_{22}) of the optional task 20.

Following the treatment of Section 4.2, the timing analysis of the traces that initiate with t_{10} and terminate with t_{12} indicates that process 10 can miss its deadline only along the trace $\rho^{10} = t_{25} \rightarrow t_{11} \rightarrow t_{12}$, which originates in the class S_{15} and which may last for a maximum duration of eight time units. In a similar manner, process 20 can miss its deadline only along the trace $\rho^{20} = t_{10} \rightarrow t_{11} \rightarrow t_{21} \rightarrow t_{23} \rightarrow t_{12} \rightarrow t_{24} \rightarrow t_{25}$, which originates in the class S_1, and which may last for a maximum duration of 14 time units. The two traces are highlighted in Fig. 7 using bold lines.

Trace ρ^{20} visits the steering class S_7, where a possible failure can be avoided by forcing the firing of transition t_{22} (i.e., rejecting the optional task) instead of t_{23} (this is the case of Fig. 5a). This decision cannot be efficiently taken before runtime, as trace ρ^{20} either catches or misses the deadline depending on the actual determination of timings during runtime. Trace ρ^{10} does not visit any steering class, so that its avoidance requires a decision be taken prior to its beginning (this is the case of Fig. 5b). Backward reachability analysis from class S_{15} indicates that the trace ρ^{10} can be avoided by forcing the rejection (i.e., the firing of transition t_{22}) when the system is in the class S_3. Also, in this case, the optimal decision requires the knowledge of the actual timings determined during the execution prior to the arrival in the steering class.

According to this analysis, steering classes S_3 and S_7 are annotated with a signature, which will be employed during runtime to decide on the acceptance or rejection of the optional task. Whereas, the steering class S_{13} is not visited by any failing or unsafe trace and its decision can be taken before runtime: In class S_{13}, the optional task

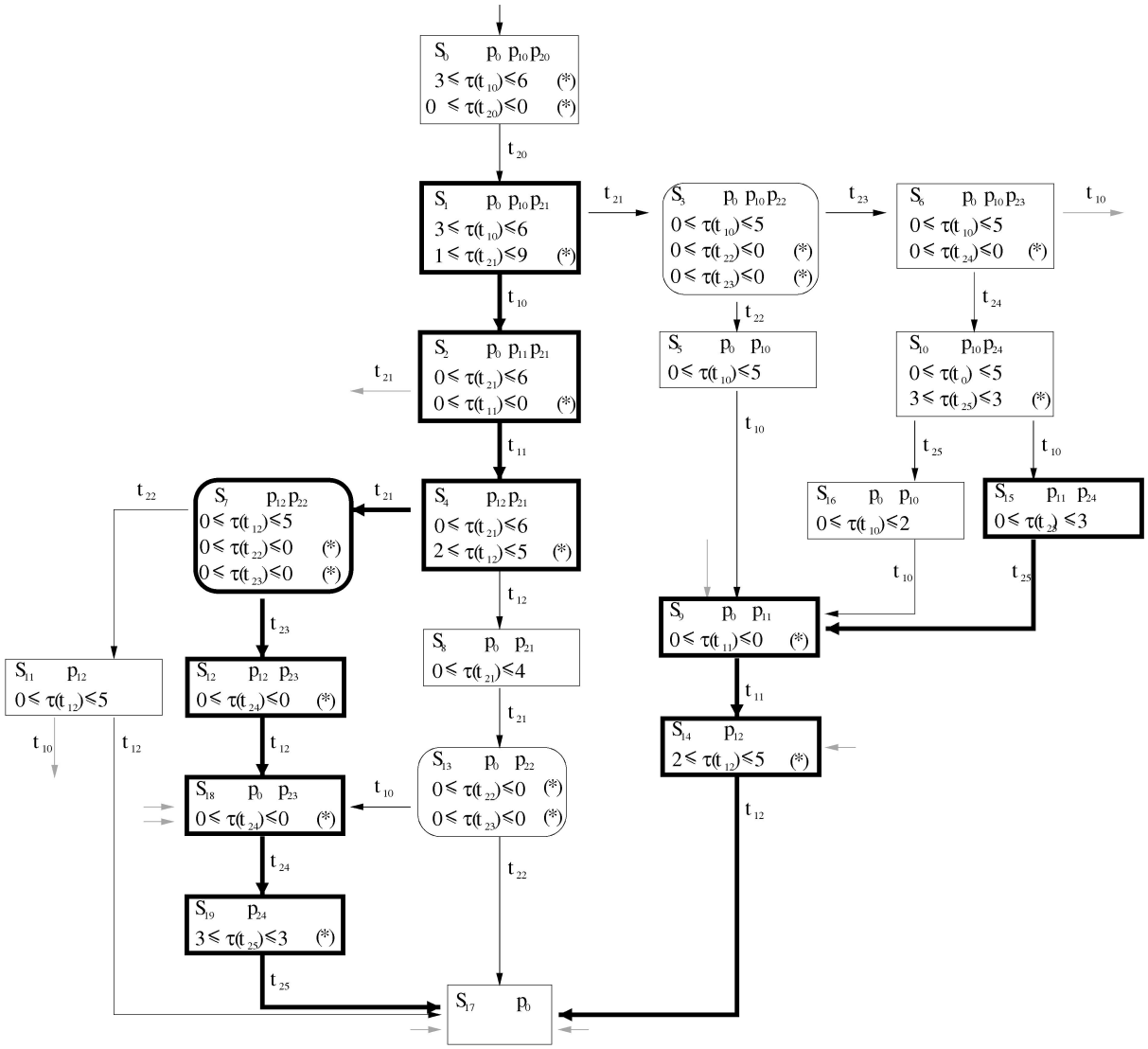


Fig. 7. A fragment of the reachability graph of the net of Fig. 6, including three steering classes (rounded rectangles) and two critical execution traces (bold lines).

will always be accepted without requiring runtime analysis of any signature.

In particular, let us consider the case of class S_7. According to Sections 4.3 and 4.4, the signature associated to the class before runtime is obtained by projecting the global system of trace ρ^{20} on: the initial event t_* corresponding to transition t_{21} which enters the steering class S_7, transitions t_{12} and t_{23} that are enabled in S_7 and then fired along the trace, transition t_{23} whose firing will disable transition t_{22}, and transition t_{25} which completes the trace. This yields the following signature:

$$\hat{G}_{\rho^{20}} = \begin{cases} 0 \le \tau(t_{23}) - \tau(t_*) \le 0 \\ 0 \le \tau(t_{12}) - \tau(t_*) \le 3 \\ 3 \le \tau(t_{25}) - \tau(t_{12}) \le 3. \end{cases} \quad (39)$$

During runtime, when the system comes to class S_7, the signature is restricted according to the dynamic determination of the firing times of enabled transitions:

$$\Gamma_{\rho^{20}} = \begin{cases} \alpha_{23,*} \le \tau(t_{23}) - \tau(t_*) \le \beta_{23,*} \\ \alpha_{12,*} \le \tau(t_{12}) - \tau(t_*) \le \beta_{12,*} \\ \tau(t_{23}) - \tau(t_*) \le \beta_{22,*}, \end{cases} \quad (40)$$

where α_{ij} and β_{ij} are the *dynamic* lower and upper bounds on $\tau(t_i) - \tau(t_j)$ at the time in which class S_7 is entered. The first two constraints are yielded by transitions t_{12} and t_{23}, which are enabled in S_7 and which will fire along the trace. The last constraint is due to transition t_{22}, which is enabled in class S_7, but which will be disabled by the firing of t_{23}.

The conjunction of $\hat{G}_{\rho^{20}}$ with $\Gamma_{\rho^{20}}$ encompasses all the possible constraints that can be used to look-ahead of the timing profile of ρ^{20} at the time when a decision must be taken on the acceptance or rejection of the optional task in class S_7. In particular, the maximum acceptable value for $\tau(t_{25})$ is a tight estimate on the time that can elapse before completing process 20. The comparison of this value against the time elapsed since the beginning of process 20 permits one to verify whether trace ρ^{20} can actually lead to the miss of a deadline. Depending on the result of this check, the

optional task is either accepted or rejected by forcing the firing of t_{23} or t_{22}, respectively.

For the concreteness of the example, let us consider the case with specific numeric values. Let τ^{20} denote the time elapsed since the start of process 20 and let τ_{12} denote the time elapsed since the beginning of the computation t_{12}. The analysis of the graph of classes indicates that, when S_7 is entered, $\alpha_{23,*} = \beta_{23,*} = \beta_{22,*} = 0$, while $\alpha_{12,*} = Max\{3 - \tau_{12}, 0\}$ and $\beta_{12,*} = 5 - \tau_{12}$, respectively. The conjunction of $\hat{G}_{\rho^{20}}$ with $\Gamma_{\rho^{20}}$ yields:

$$\begin{cases} \hat{G}_{\rho^{20}} \\ \Gamma_{\rho^{20}} \end{cases} = \begin{cases} 0 \le \tau(t_{23}) - \tau(t_*) \le 0 \\ Max\{5 - \tau_{12}, 0\} \le \tau(t_{12}) - \tau(t_*) \le Min\{5 - \tau_{12}, 3\} \\ 3 \le \tau(t_{25}) - \tau(t_{12}) \le 3. \end{cases} \tag{41}$$

If $\tau_{12} < 2$, the system does not accept any solution as $Max\{5 - \tau_{12}, 0\} \ge 3 \ge Min\{5 - \tau_{12}, 3\}$. This means that the unsafe trace ρ_{20} is not feasible under the actual current timing, so that the optional task can be safely accepted. Whereas, if $\tau_{12} \ge 2$, the system accepts solutions and the maximum acceptable value for $\tau(t_{25})$ is equal to $3 + Min\{5 - \tau_{12}, 3\}$, which, in turn, is equal to six. If six plus the time elapsed since the beginning of process 20 is less than the process deadline (i.e., if $6 + \tau^{20} \le 13$), then the optional task can be safely accepted by firing transition t_{23}. Otherwise, transition t_{22} must be fired and the task rejected to avoid entering an unsafe state from which the system could be yielded to miss a deadline without having further chances to steer the execution.

4.6 Managing Space Complexity

The acceptance test expounded in Section 4.4 relies on a precomputed representation of the topology of the entire graph of state classes to track the runtime state of the system to the steering classes identified offline. In the case of complex systems, this topology may result in a space complexity which is not affordable within the limits of the target platform.

Space complexity may be largely reduced by relating static analysis to dynamic steering through markings rather than through state classes: Before runtime, (projected) trace signatures are indexed by the marking of the steering class to which they are associated; during runtime, the evolution of the system is tracked in the graph of markings rather than in the graph of state classes; whenever the system comes to an acceptance/rejection point, the failure check is applied to all the signatures associated with the current marking.

Steering through state classes or markings basically follow the same principle: Both of them compare the runtime state against sets of critical states identified during the static analysis. In this perspective, a marking comprises the states of different state classes. Since each steering marking is annotated with the signatures of *all* the state classes with that same marking, the acceptance test checks every signature belonging to the specific class which collects the current state. This ensures that steering through the marking graph is safe, i.e., that any failing trace is detected. Besides, when signatures are restricted by actual timing constraints from the current state, any nonempty solution corresponds to a feasible evolution for the system. Any such evolution exceeding the time left to the expiration of the deadline corresponds to a feasible failure. This ensures that steering in the marking graph is also strict, i.e., that it does not reject tasks that would have been accepted in the steering based on state classes.

With respect to space complexity, the graph of markings is by far smaller than the graph of state classes, and, in any case, can be dealt with efficient hierarchical representations [32], [11]. Since multiple signatures are mapped to a single steering marking, the acceptance test must be performed on more signatures, which increases the time complexity with respect to the case of state class steering. However, computational experience shows that, in most cases, different traces produce the same projected signature and that the same signature is annotated in multiple state classes. This results in the number of unique signatures annotated on the totality of markings being far less than the number of signatures annotated on the totality of state classes. For the same reason, the maximum number of signatures annotated on the same steering marking does not significantly increases when steering on classes is replaced by steering on markings.

5 Computational Experience

Support for the proposed strategy has been embedded within a CASE tool, named ORIS, for the automated verification of time-dependent reactive systems [10]. In the rest of this section, computational experience is reported on the application of the tool to two sample cases.

5.1 A Simple System

The first example addresses the case of a system in which three processes P^1, P^2, and P^3 contend for three non-preemptable resources R_0, R_1, and R_2. Processes P^1 and P^2 are assumed to be mandatory, whereas tasks of process P^3 are optional.

A schema of the three processes is shown in Fig. 8. P^1 is periodic with the period equal to nine time units; in each period, P^1 releases a task with a maximum jittering delay of one time unit; the task requires resource R_0 for a first computation lasting a time within two and three time units, and both resources R_0 and R_1 for a second computation lasting a time within one and two time units; P^1 is constrained to complete within eight time units. P^2 is an aperiodic process, with minimum interarrival time of 10 units; each instance has a computation time within one and two units, needs resources R_0 and R_2, and must complete within eight time units. P^3 is an aperiodic process with a minimum intertime of six units, requiring resources R_1 and R_2 for a computation time within one and two time units, and having a deadline of six time units. An analysis of process laxities and computation times shows that process P^1 can miss the deadline if it is blocked on resource locks by both P^2 and P^3. The same happens when P^2 is blocked by P^1 and P^3. Both these conditions may occur under any scheduling policy which is not able to withhold or reject tasks generated by P^3 under critical phasings.

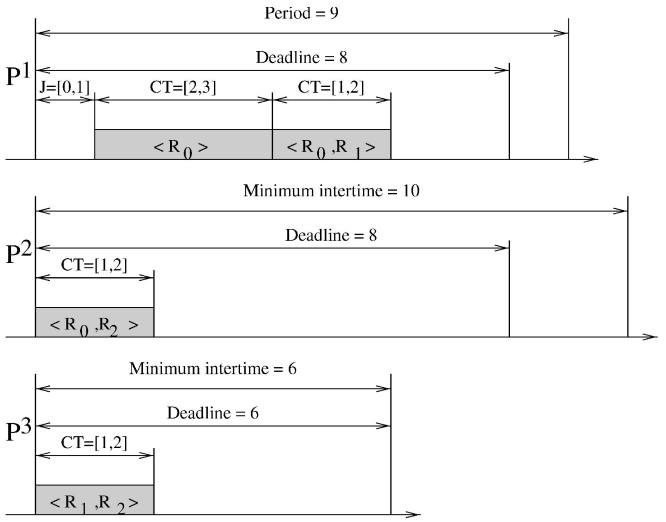


Fig. 8. One periodic and two aperiodic process contending for resources R_0, R_1, and R_2.

In the following, the scheduling problem is solved using the mixed static/dynamic approach to acceptance and guarantee expounded in Section 4. To this end, the system is modeled as a TPN in Fig. 9. Places P_0, P_1, and P_2 model the three shared resources. The three *source* transitions t_{10}, t_{20}, and t_{30} account for the arrival of task service requests of the three processes. Transition t_{10} fires periodically every nine time units, while transitions t_{20} and t_{30} fire repetitively with a minimum intertime equal to 10 and six, respectively. Transition t_{11} accounts for the jittering delay of process P^1. The acquisition of a resource (transitions t_{12}, t_{14}, t_{21}, and t_{33}) is associated with firing interval $[0, 0]$ so as to model an instantaneous action. Other instantaneous actions are the acceptance or the rejection of process P^3, modeled with transitions t_{32} and t_{31}. Resource usage (transitions t_{13}, t_{15}, t_{22}, and t_{34}) is modeled with a nonnull firing time that accounts for the best and the worst case execution time.

5.1.1 Static Analysis

Static enumeration of reachable state classes of system results in a graph of 5870 classes. Execution traces corresponding to tasks of each process are identified as the traces leading from the start to the completion transition of each process: Processes P^1, P^2, and P^3 start with transitions t_{10}, t_{20}, and t_{32}, and they complete with transitions t_{15}, t_{22}, and t_{36}, respectively.

The analysis of timing profiles of execution traces shows that both the mandatory processes can miss the deadline if no adequate steering is performed:

Process	Execution traces	Possible failures
P^1	3594	103
P^2	1871	352
P^3	1327	0

85 classes (out of the 408 classes in which an accept/reject is decided) are identified as critical steering points requiring dynamic check for acceptance and they are annotated with a total of 171 signatures with a mean size of 5.2 events per signature. The maximum number of signatures annotated on a steering class is five. Using the graph of markings to reduce space complexity, the overall set of 5,870 state classes is mapped to 52 markings, and the 85 steering classes are mapped to only five markings. Moreover, the total set of signatures annotated on the graph is reduced from 171 to 17, while the maximum number of signatures to be checked in the same steering point is still equal to five.

5.1.2 Dynamic Steering

The performance of dynamic steering was measured by simulating monitor execution. The effectiveness of the steering policy in accepting optional processes in critical state classes has been quantified through the *reclaiming* ratio, i.e., the ratio of positive responses produced by the steering algorithm with respect to the number of invocations of the algorithm itself.

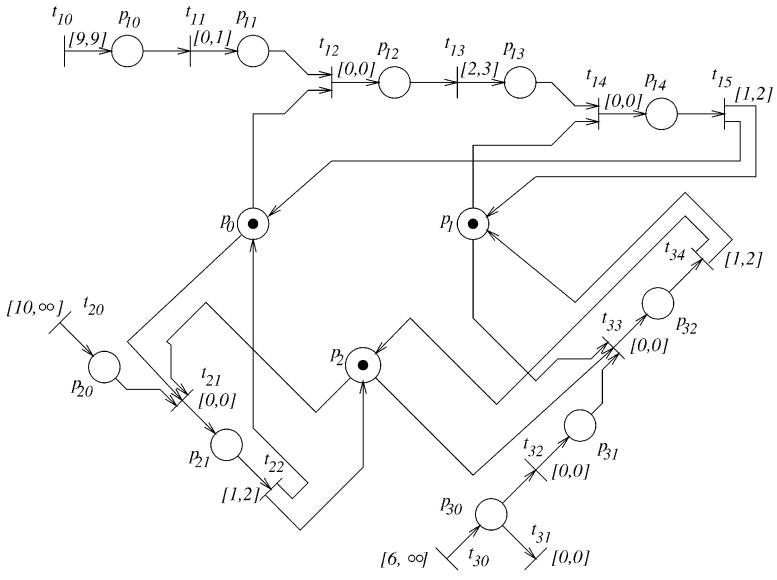


Fig. 9. A Time Petri Net modeling the concurrent composition of the three processes of Fig. 8.

According to the common practice of testing, a criterion of *coverage* was assumed to provide a heuristic evaluation of the trustworthiness of simulation results. Since static analysis results in an exhaustive reachability graph among state classes, coverage was evaluated as the ratio of state classes visited by the system during the simulation with respect to the total number of state classes enumerated in the static analysis.

During the simulation, the time to fire of each enabled transition t_0 was evaluated as a random variable. Uniform distribution has been assumed for finite time intervals, while exponential distribution has been used for unbounded intervals. Probability distributions of different transitions were made mutually independent.

The system was tested under various steering policies, i.e., using various algorithms to decide the acceptance/rejection of an optional task in a steering class:

- *Pessimistic.* The optional task is always rejected, regardless of the eventual nonfeasibility of any unsafe trace. This algorithm is correct but uneffective: It never leads to a failure, but has 0 percent reclaiming.
- *Optimistic.* The optional task is always accepted, regardless of the feasibility of some unsafe traces. This behavior is maximally effective, but, as expected from static analysis, causes some tasks to fail. More precisely, 868 out of 10^8 executed tasks miss their deadline.
- *Static.* The dynamic checking of the feasibility of unsafe traces is only partially implemented: The expectation about the time available before the deadline expiration is refined according to (32), but, the expected duration of traces as computed before runtime is not refined (i.e., the system of (33) is not recomputed). This guarantees that no task can ever fail, and the system obtains a 29 percent reclaiming with no significant runtime effort.
- *Dynamic.* The dynamic checking of the feasibility of unsafe traces is fully implemented according to the algorithm of Section 4.4. Now, the system obtains a 78 percent reclaiming and still guarantees that no tasks can fail.

Results of simulation through 10^8 transition firings are summarized in Table 1.

Simulation results show that the reachability graph is at most half covered. Fig. 10 illustrates that the coverage grows very slowly with the number of simulation steps executed; data collected during several simulations can be used to roughly estimate that at least 10^{13} simulation steps would be necessary to obtain a full coverage. In this case, exhaustive verification seems to be by far less expensive than simulation.

5.2 A Robotic Arm Controller

The second example aims at stressing the problem of space complexity in the proposed approach to dynamic steering. To this end, the example addresses the case of the control software for an industrial robotic arm, derived from a real application developed and implemented at the University of Parma [5].

TABLE 1
Results of 10^8 Simulation Steps for the Model of the Three Processes of Fig. 8

Policy	Coverage	Reclaiming	Failures
Pessimistic	45%	0%	0
Optimistic	50%	100%	868
Static	48%	29%	0
Dynamic	50%	78%	0

The arm is programmed to lift the objects carried by a conveyor belt and put them into a basket, using an intermediate shelf as a temporary buffer. The control software of the arm is comprised of five interacting tasks.

- A Trajectory Control (TC) is spawned every 16 ms to issue set-points to the low-level arm controller, reading elementary commands from a shared buffer. This task can terminate immediately if there are no commands to process, otherwise it has a WCT between 5 ms and 6 ms. The TC has a deadline equal to its period, i.e., 16 ms.
- Two Motion Executors (ME) are invoked whenever the system must lift an item from the belt or put it into the basket; each ME generates a set of commands in the command buffer. The first ME (*lifter*) is activated whenever the system senses the presence of an object on the conveyor belt, with a minimum intertime of 40 ms between each activation. The lifter issues to the TC the commands to get the object and put it in the buffer, then activates the second ME (*putter*). The putter sends to the TC the commands to take the object from the intermediate buffer and put it into the final basket. Each ME produces its commands within 4 to 8 ms.
- A Sensor Reader (SR) reads several different sensors every 24 ms. The readings of this tasks are used by the TC. The SR has a WCT of 1~ms, and a deadline of 24 ms.
- A Motion Planner (MP) computes alternative motion plans for the MEs to use. The MP uses an incremental approach, i.e., it refines the motion plan each time it can be run without compromising the safeness of the remaining tasks. The MP tries to run once every 80 ms and, if accepted, produces a refined plan within 10 ms.

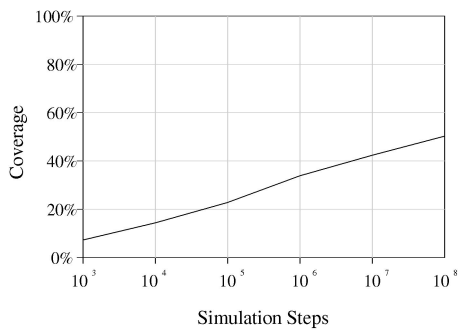


Fig. 10. Coverage of the reachability graph obtained with increasing numbers of simulation steps.

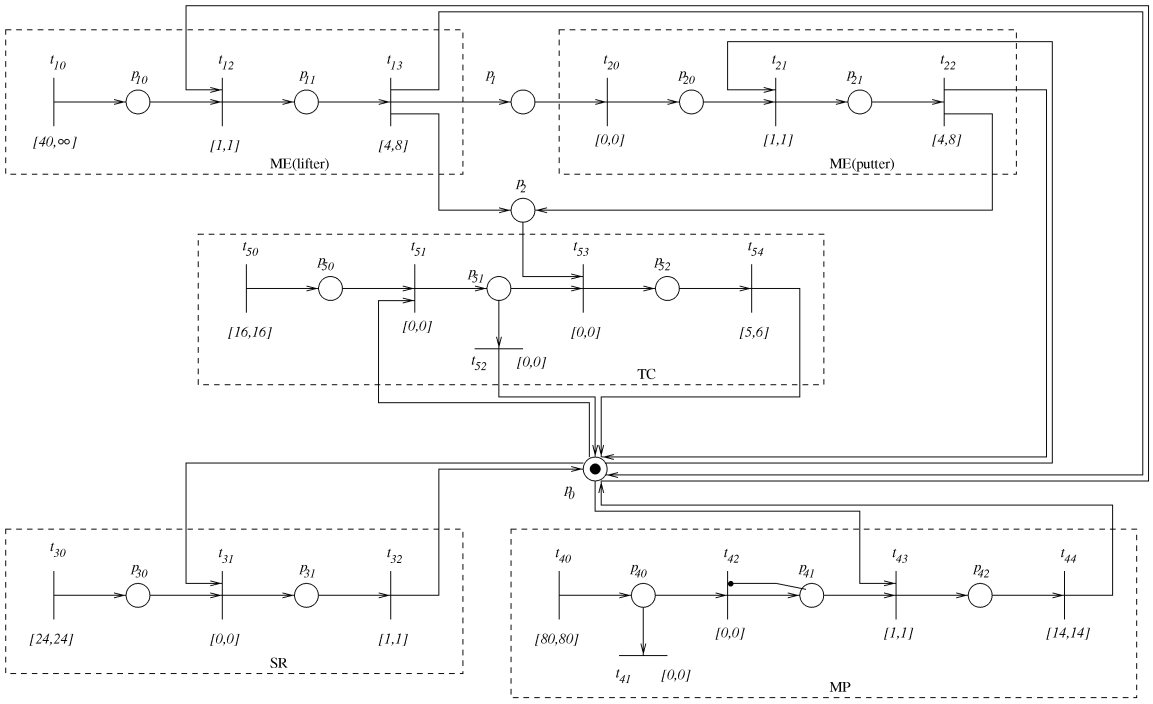


Fig. 11. A Time Petri Net modeling the robotic arm controller.

The system is modeled as a TPN in Fig. 11. Place P_0 models the CPU as a resource shared by each task, place P_1 models the intermediate buffer shelf and place P_2 models the command buffer.

5.2.1 Static Analysis

Static enumeration of reachable state classes for the arm controller results in a graph with 49,544 classes. The analysis of timing profiles of execution traces is reported in the following table:

Process	Execution traces	Possible failures
TC	22344	1095
SR	17598	16
ME_1 (lifter)	17076	0
ME_2 (putter)	18615	0
MP	15670	0

878 state classes are annotated with a total of 2,533 signatures with a mean size of 6.6 events per signature. The maximum number of signatures annotated on a steering class is eight. Using the graph of markings to reduce space complexity, 49,544 state classes are replaced by 321 markings, with a total of 520 signatures annotated on 54 different markings. The maximum number of signatures on the same steering point increases from eight to nine.

5.2.2 Dynamic Steering

Simulating the arm controller through 10^8 transition firings produces the results summarized in Table 2.

6 Conclusions

We presented an enumerative approach to the analysis of concurrent and distributed reactive systems modeled as automata with the timing semantics of Time Petri nets. The method develops on the analysis technique proposed in [8], [9] which creates a discrete partition in the dense timed state space using equivalence classes in the form of firing domains (time zones). The formulation of [8] was revised and extended, to make explicit the actual semantics of the reachability relation among state classes and to provide close form expressions for their enumeration. This not only reduces the complexity of enumeration but also opens the way to analytical development supporting an original technique for the evaluation of a comprehensive timing profile for any execution trace of the model. In particular, this supports the evaluation of a tight bound on the time elapsing between any two events within an execution sequence.

The method can be applied to support timing analysis in a variety of application contexts, including real-time systems and communication protocols. In particular, in this paper, the analysis technique was finalized to support dynamic resource reclaiming within a static table driven approach to the scheduling of a hard real-time system with mandatory and optional tasks represented as Time Petri Nets. This supports the representation of complex tasking sets, with any kind of periodicity constraints, with individual tasks characterized by computation times and deadlines, requirements on nonpreemptive exclusive resources, and precedence and communication constraints.

The proposed approach is basically static, as it requires preruntime determination of the structure and timing parameters of tasks that can be generated during runtime. However, before runtime, the analysis is carried out on nondeterministic timings represented by a dense time interval (possibly not finite), rather than by a worst case. This supports a mixed static/dynamic strategy in which decisions on the acceptance of optional tasks can be taken by

TABLE 2
Results of 10^8 Simulation Steps for the Robotic Arm Control Software Model

Policy	Coverage	Reclaiming	Failures
Pessimistic	21%	0%	0
Optimistic	37%	100%	110124
Static	22%	0%	0
Dynamic	27%	12%	0

complementing static predictions with the knowledge of the actual execution timing and of the actual tasks generated during runtime. In so doing, firing domains employed in the representation of the timed state space comprise a suitable abstraction for static construction of decision regions that can be efficiently refined during runtime.

Our ongoing research and experimentation on the subject is now aimed at extending the proposed method to represent and analyze systems which employ preemptive protocols in the access to mutual exclusive resources.

ACKNOWLEDGMENTS

The author would like to thank Marco Lusini for his invaluable contribution in the design and implementation of the ORIS tool. Also, Gacomo Bucci, for the generous and long-termed support that he always gave to this research. This work was supported by the Italian Ministry of Scientific and Technological Research (MURST) as a part of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications" (MOSAICO) and "Reactive and Reliable Systems for Industrial Appliations" (ISIDE).

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows.* Prentice Hall, 1993.

[2] R. Alur and D.L. Dill, "Automata for Modeling Real-Time Systems," *Proc. 17th Int'l Colloquium on Automata, Languages, and Programming,* 1990.

[3] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking for Real-Time Systems," *Proc. Fifth Symp. Logic in Computer Science,* June 1990.

[4] R. Alur, T.A. Henzinger, and P.-H. Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Trans. Software Eng.,* vol. 22, no. 3, Mar. 1996.

[5] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "A Real-Time Library for the Design of Hybrid Robot Control Architectures," *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems, IROS '98,* Oct. 1998.

[6] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: A Tool-Suite for Automatic Verification of Real-Time Systems," *Hybrid Systems III,* R. Alur, T.A. Henzinger, and E.D. Sontag, eds., 1996.

[7] J.A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer,* vol. 28, no. 6, June 1995.

[8] B. Berthomieu and M. Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 17, no. 3, Mar. 1991.

[9] B. Berthomieu and M. Menasche, "An Enumerative Approach for Analyzing Time Petri Nets," *Proc. Int'l Federation for Information Processing Congress,* Sept. 1983.

[10] G. Bucci and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 21, no. 12, Dec. 1995.

[11] S. Christensen and L. Petrucci, "Modular State Space Analysis of Coloured Petri Nets," *Proc. 16th Int'l Conf. Application and Theory of Petri Nets,* June 1995.

[12] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automated Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Programming Languages and Systems,* vol. 8, no. 2, Apr. 1986.

[13] J.C. Corbett, "Timing Analysis of Ada Tasking Programs," *IEEE Trans. Software Eng.,* vol. 22, no. 7, pp. 461-483, July 1996.

[14] C. Corcoubetis, D. Dill, M. Chatzaki, and P. Tzounakis, "Verification with Real-Time COSPAN," *Proc. Int'l Workshop on Computer Aided Verification,* 1992.

[15] C. Corcoubetis and M. Yannakakis, "Minimum and Maximum Delay Problems in Real-Time Systems," *Proc. Int'l Workshop Computer Aided Verification,* 1991.

[16] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods for Validating Them," *IEEE Trans. Software Eng.,* vol. 11, no. 1, Jan. 1985.

[17] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool KRONOS," *Hybrid Systems III,* R. Alur, T.A. Henzinger, and E.D. Sontag, eds., 1996.

[18] D. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. Workshop Computer Aided Verification Methods for Finite State Systems,* 1989.

[19] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan, "Quantitative Temporal Reasoning," *Proc. Workshop Automatic Verification Methods for Finite State Systems,* 1989.

[20] A. Fehnker, "Scheduling a Steel Plant with Timed Automata," *Proc. Sixth Int'l Conf. Real-Time Computing Systems and Applications,* Dec. 1999.

[21] N. Gehani and K. Ramamrithan, "Real-Time Concurrent C: A Language for Programming Dynamic Real-Time Systems," *J. Real Time Systems,* no. 3, 1991.

[22] B. Hamidzadeh and Y. Atif, "Dynamic Scheduling of Real Time Tasks, by Sssignment," *IEEE Concurrency,* pp. 14-25, Oct.-Dec. 1998.

[23] M.G. Harbour, M.H. Klein, and J.P. Lehoczky, "Timing Analysis for Fixed Priority Scheduling of Hard Real Time Systems," *IEEE Trans. Software Eng.,* vol. 20, no. 1, pp. 13-28, Jan. 1994.

[24] M. Humphrey and J.A. Stankovic, "Predictable Threads for Dynamic Hard Real Time Environments," *IEEE Trans. Software Eng.,* vol. 23, no. 12, pp. 745-758, Dec. 1997.

[25] N. Leveson, "Safety Analysis Using Petri Nets," *IEEE Trans. Software Eng.,* vol. 13, no. 3, Mar. 1987.

[26] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, 1973.

[27] M. Lusini and E. Vicario, "Static Analysis and Dynamic Steering of Time-Dependent Systems Using Time Petri Nets," Technical Report 28.98, Dip. Sistemi e Informatica, Univ. of Florence, 1998.

[28] P. Merlin and D.J. Farber, "Recoverability of Communication Protocols," *IEEE Trans. Comm.,* vol. 24, no. 9, Sept. 1976.

[29] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, Apr. 1989.

[30] X. Nicollin, J. Sifakis, and S. Yovine, "Compiling Real-Time Specifications into Extended Automata," *IEEE Trans. Software Eng.,* vol. 18, no. 9, Sept. 1992.

[31] C. Norstrom, A. Wall, and W. Yi, "Timed Automata as Task Models for Event Driven Systems," *Proc. Sixth Int'l Conf. Real-Time Computing Systems and Applications,* Dec. 1999.

[32] M. Notomi and T. Murata, "Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent Software Analysis," *IEEE Trans. Software Eng.,* vol. 19, no. 1, Jan. 1993.

[33] J.S. Ostroff, "Deciding properties of Timed Transition Models," *IEEE Trans. Paralel and Distributed Systems,* vol. 1, no. 2, Apr. 1990.

[34] Y.A. Papelis and T.L. Casavant, "Specification of Parallel/Distributed Software and Systems by Petri Nets with Transition Enabling Functions," *IEEE Trans. Software Eng.,* vol. 18, no. 3, Mar. 1992.

[35] D.-T. Peng, K.G. Shin, and T.F. Abdelzaher, "Assignment and Scheduling Communicating Periodic tasks in Distributed Real Time Systems," *IEEE Trans. Software Eng.,* vol. 23, no. 12, pp. 745-758, Dec. 1997.

[36] J. Xu and D.L. Parnas, "On Satisfying Timing Constraints in Hard Real Time Systems," *IEEE Trans. Software Eng.*, vol. 19, no. 1, Jan. 1993.
[37] J.L. Peterson, "Petri Nets," *ACM Computing Surveys*, vol. 9, no. 3, Sept. 1977.
[38] K. Ramamritham and J.A. Stankovic, "Scheduling Algorithms and Operating System Support for Real Time Systems," *Proc. IEEE*, vol. 82, no. 1, Jan. 1994.
[39] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 4, pp. 412-420, Apr. 1995.
[40] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *Trans. Computers*, vol. 38, no. 8, Aug. 1989.
[41] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real Time Synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1175-1185, Sept. 1990.
[42] L. Sha, R. Rajkumar, and S.S. Sathaye, "Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real Time Systems," *Proc. IEEE*, vol. 82, no. 1, Jan. 1994.
[43] J.A. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real Time systems," *IEEE Software*, vol. 8, no. 3, pp. 62-72, May 1991.
[44] J.A. Stankovic and K. Ramamritham, "What is Predictability for Real Time Systems," *J. Real Time Systems*, vol. 2, Dec. 1990.
[45] J.A. Stankovic, "Misconceptions About Real Time Computing," *IEEE Computer*, vol. 21, no. 10, Oct. 1988.
[46] J. Sun, M.K. Gardner, and J.W.S. Liu, "Bounding Completion Times of Jobs with Arbitrary Release Times, Variable Execution Times, and Resource Sharing," *IEEE Trans. Software Eng.*, vol. 23, no. 10, Oct. 1997.
[47] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real Time Systems," *IEEE Trans. Software Eng.*, vol. 13, no. 5, pp. 564-577, May 1987.
[48] J. Xu and D.L. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations," *IEEE Trans. Software Eng.*, vol. 16, no. 3, pp. 360-369, Mar. 1990.

Enrico Vicario received the Doctoral degree in electronics engineering (Laurea in Ingegneria Elettronica) and the PhD degree in information and communications technology (Dottorato in Ingegneria Informatica e delle Telecomunicazioni) from the University of Florence, in 1990 and 1994, respectively. From 1994 to 1998 he was an assistant professor (Ricercatore) at the Engineering School of the University of Florence. Since 1998, he has been an associate professor of information engineering, first at the Engineering School of the University of Ancona and now at the Engineering School of the University of Florence. His research activity addresses both software engineering and visual information technologies. Specifically, he has been working on formal specification and validation of time dependent systems, protocol engineering, content modeling and retrieval for image and video, visual languages and formalisms, and usability engineering for interactive graphic and multimedia applications. He is an associate editor of *IEEE Multimedia*. He is a member of the IEEE Computer Society.

▷ **For further information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.