Int J Softw Tools Technol Transfer (2010) 12:391–403
DOI 10.1007/s10009-010-0156-8

REGULAR PAPER

Oris: a tool for modeling, verification and evaluation of real-time systems

Giacomo Bucci · Laura Carnevali · Lorenzo Ridi · Enrico Vicario

Published online: 9 May 2010

Abstract Oris is a tool for qualitative verification and quantitative evaluation of reactive timed systems, which supports modeling and analysis of various classes of timed extensions of Petri Nets. As most characterizing features, Oris implements symbolic state space analysis of preemptive Time Petri Nets, which enable schedulability analysis of real-time systems running under priority preemptive scheduling; and stochastic Time Petri Nets, which enable an integrated approach to qualitative verification and quantitative evaluation. In this paper, we present the current version of the tool and we illustrate its application to two different case studies in the areas of qualitative verification and quantitative evaluation, respectively.

Keywords Qualitative verification · Quantitative evaluation · Symbolic state-space enumeration · Time Petri Nets · Preemptive Time Petri Nets · Stochastic Time Petri Nets · Difference Bounds Matrix · Real-time systems

1 Introduction

Time affects the development of a large class of reactive systems, for either explicit real-time constraints or sequencing limitations resulting from timed behavior. This calls for models which explicitly address these factors in order to achieve predictability, a requirement for critical applications. Formal methods such as Timed Automata (TA) [1–3] or Time Petri Nets (TPNs) [4–7] have been effectively adopted for modeling and validation of time-dependent systems. For all these methods, the semantics of the system is defined in terms of state transition rules driving the evolution of logical locations and of a set of quantitative clocks. While the former are discrete, the latter take values in dense domains. To obtain a discrete representation, the state-space is covered through equivalence classes, each characterized by a time domain collecting a dense variety of clock values [1–4,6,8–10]. A number of tools [2,3,8–13] have been developed to support the formal verification of real-time systems. On the one hand, in the context of TA, UPPAAL is a well-consolidated tool for validation and verification of real-time systems modeled as networks of timed automata extended with data types [2,14]. It includes a model checker providing a diagnostic trace that can be simulated to understand why a property is (or is not) satisfied. The application of TA to scheduling theory is enabled by the Times tool [11,15], which supports modeling, analysis, and synthesis of schedules and executable code. It allows the treatment of tasks with asynchronous and dense release times, running under the most practiced scheduling disciplines. As a limitation, the underlying analysis rules out nondeterministic computation times taking values within dense intervals. Kronos supports automatic scheduler generation for systems modeled as TA [8] and automatic generation of both non-preemptable and preemptable code for embedded Real-Time Operating Systems (RTOSs) [9,10]. On the other hand, in the context of TPNs, Tina [13], and Romeo [12] are well-established tools supporting the construction of various abstract state-space representations and the model-checking of reachability properties. As a

G. Bucci · L. Carnevali (✉) · L. Ridi · E. Vicario
Dipartimento di Sistemi e Informatica, Università di Firenze, Firenze, Italy
e-mail: carnevali@dsi.unifi.it

G. Bucci
e-mail: bucci@dsi.unifi.it

L. Ridi
e-mail: ridi@dsi.unifi.it

E. Vicario
e-mail: vicario@dsi.unifi.it

relevant trait, Romeo also supports both approximate and exact enumeration of the state space for an extension of TPNs (Scheduling-TPNs) modeling preemption.

In this paper, we provide a description of the Oris Tool, which integrates a rich set of modules for building, simulating, analyzing, and validating real-time systems described through various TPN formalisms. In particular, Oris supports symbolic state space enumeration of preemptive Time Petri Nets (pTPNs) [7,16], which extend the model of TPNs [4,5] with a mechanism of resource assignment which conditions the advancement of timers of enabled transitions. The resulting formalism allows the representation of complex and densely timed task-sets running under priority preemptive scheduling, enabling reachability analysis and evaluation of tight bounds on the time elapsed between events along critical execution sequences. As a characterizing trait, Oris also implements recently developed analysis techniques for models represented as stochastic Time Petri Nets (sTPNs) [17–22], which associate the static firing interval of each transition with a (dense) probability density function. This supports the integration of qualitative verification and quantitative evaluation, enabling the derivation of performance measures.

In this paper, we give an informal description of the tool focusing on examples and referring the reader to other papers for formal description of analysis methods. In particular, Sect. 2 provides a brief overview of Oris; Sects. 3 and 4 illustrate the application of the tool to qualitative verification of pTPN models and quantitative evaluation of sTPN models, respectively; conclusions are finally drawn in Sect. 5.

2 An overview of the Oris Tool

2.1 Oris basic features

Oris is a tool for modeling and analysis of reactive timed systems based on various classes of Petri Nets. As a salient core capability, it supports qualitative and quantitative analysis techniques based on symbolic state space enumeration. More specifically, Oris supports editing and analysis of the following timed extensions of Petri Nets:

- *Time Petri Nets (TPNs):* the classical model of [6], which associates a dense timer to each transition.
- *preemptive Time Petri Nets (pTPNs):* an extension of TPNs that supports the representation of suspension in a manner that can compare with stopwatch automata [23] and Petri Nets with hyper-arcs [24] and that can be applied to the representation of preemptive systems [7,16,25–27].
- *stochastic preemptive Time Petri Nets (spTPNs):* a discrete-time variant of TPNs based on a maximal step semantics of concurrency that supports the representation of preemptive behavior and associates quantitative probabilities with timers and switches [28].
- *stochastic Time Petri Nets (sTPNs):* a kind of non-Markovian Stochastic Petri Nets obtained by extending TPNs with a stochastic characterization of time distributions and choices [17,18,20,21].

The Oris Tool also includes various additional components with specific purposes:

- a module supporting timed animation with interactive token game: it was recently extended to support also stochastic simulation; however, the extension was instrumental to an internal experimentation and is presently limited to models with uniform and exponential distributions;
- a module that supports the derivation of all the execution traces between two events;
- a model checker for real-time temporal logic formulae (state and action based, with timing constraints on temporal operators) [14]: as special features, the model checker identifies all the witnesses that satisfy the formula and the formulae can be expressed using a visual formalism [29,30];
- a module for the visual representation and inspection of the dense variety of timings that is associated with any execution trace;
- a visual interface for the automated generation of timeline schemas of real-time task-sets; and
- a module for the automated translation of timeline schemas into pTPN models and into code running under RTAI [25,31,32].

2.2 Oris environment

The Oris Tool integrates a set of plugins that operate on timed extensions of Petri Nets into a comprehensive development environment, which supports the user along the steps of specification, verification, implementation, and testing. The environment is hierarchically arranged into *workspaces*, *projects*, and *plugin documents*: a workspace is made by one or more projects, each including a set of plugin documents. A plugin may produce one or more output documents and may require a set of input documents produced by other plugins.

Figure 1 depicts the interface of the Oris Tool. The panel *Workspace* on the left represents the composition of the current workspace in a tree structure, where the current project is highlighted in bold. Plugin documents are displayed in the main tabbed panel and they can be created through the

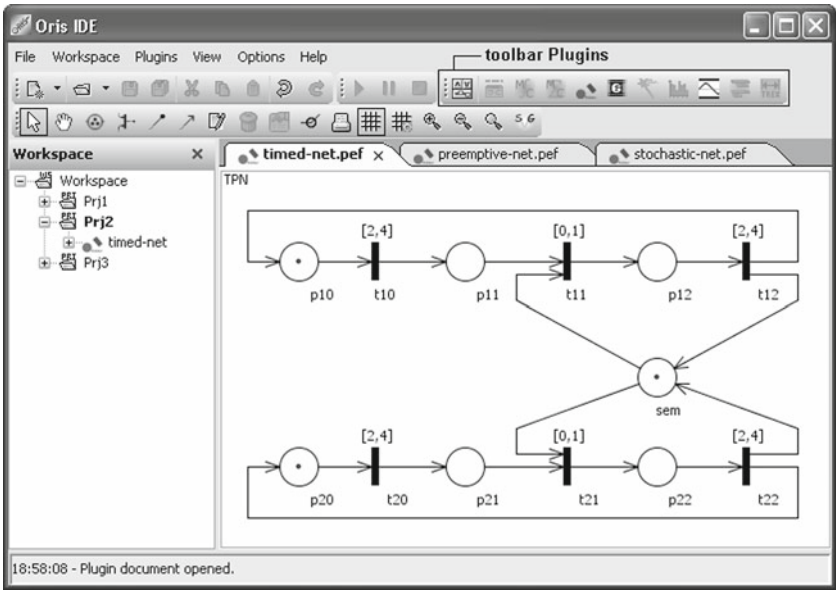

Fig. 1 A screenshot of the interface of the Oris Tool. The left panel *Workspace* illustrates the structure of the current workspace *Workspace*, which consists of projects *Prj1*, *Prj2*, and Prj3. The current project *Prj2* includes the document *timed-net*, which is produced by the plugin Petri Nets Editor and represents a TPN model. The toolbar *Plugins* enables the buttons of plugins Logic Formulae Editor and Timelines Editor, which do not require input documents, and those of plugins Petri Net Animator and Continuous Time Analyzer, which require a Petri Net model as input document. The input dependencies of all the other plugins are not satisfied within the current project (e.g., the plugin Trace Extractor requires the state space of a Petri Net model in input) and, thus, their buttons are not enabled

corresponding buttons of the toolbar *Plugins*. The user is allowed to add a plugin document to the current project only if the project contains all the documents that the plugin requires in input. According to this, the toolbar *Plugins* enables only the buttons of plugins whose documents can be added to the current project.

2.3 Oris current release

The Oris Tool is developed by the Software Technologies Laboratory of the University of Florence. To evaluate the applicability of formal methodology, it was used in various experiences of technology transfer targeting several steps of real practice development processes, including model generation from semi-formal specification, automated code generation, automated real-time test-case selection and sensitization, and Execution Time profiling. Nevertheless, we still think of it as a scientific workbench supporting experimentation and integration of theoretical results.

Oris can be downloaded from the home page http://www.stlab.dsi.unifi.it/oris/. The baseline release of the tool comprises the following modules: Petri Nets Editor, Timelines Editor, Petri Nets Animator, Continuous Time Analyzer, Trace Extractor, Logic Formulae Editor, Continuous Time Model Checker, and Trace Explorer. Additional components are provided on demand.

3 Schedulability verification through pTPNs

PTPNs extend the basic model of TPNs by associating each transition with the request of a resource set and with a priority level: an enabled transition is progressing and advances its clock if no other enabled transition requires any of its resources with a higher priority level; otherwise, it is suspended and maintains the value of its clock. The resulting formalism encompasses the representation of the suspension mechanism and thus of preemptive behavior, allowing natural description of complex real-time systems running under preemptive scheduling, with periodic and sporadic tasks, with nondeterministic Execution Times, with synchronization and precedence relations. Note that some temporal parameters of the model may be nondeterministic due to various practical factors: the arrival of asynchronous tasks is associated with a minimum but not a maximum inter-time; the Execution Time of a computation may be associated with a nondeterministic range of variation to make the model robust with respect to possible variations of the implementation and also to neglect dependencies among the Execution Times of different computations; temporal parameters of a re-engineered or reused component usually range within boundaries which over-approximate those of the implementation, to avoid difficulties in obtaining a precise estimate of Execution Times and release times.

Syntax and semantics of pTPNs are formally expounded in [7], together with related analysis techniques. Here, we

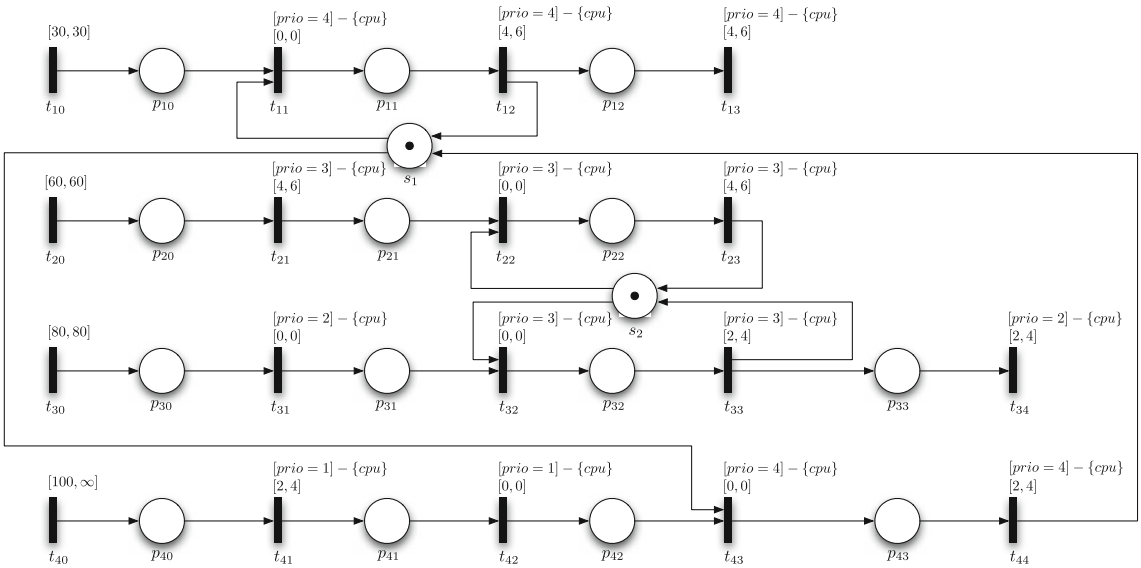

Fig. 2 The pTPN model for a task-set made by four tasks synchronized on two semaphores

discuss a case example to the purpose of illustrating how the Oris Tool supports the development of real-time preemptive systems.

3.1 Specification of real-time task-sets

We consider a system comprised of four tasks Tsk_1, Tsk_2, Tsk_3, and Tsk_4, each characterized by a minimum and maximum computation time and running on a single cpu. Tasks Tsk_1, Tsk_2, and Tsk_3 are periodic with period of 30, 60, and 80 time units, respectively, and they run at priority level 4, 3, and 2, respectively (in the Oris Tool, high priority numbers run first). Task Tsk_4 is sporadic, with minimum inter-arrival time of 100 time units, and runs at priority level 1. Tsk_1 and Tsk_2 have two computation steps with Execution Time comprised between 4 and 6 time units; Tsk_3 and Tsk_4 have two computation steps with Execution Time comprised between 2 and 4 time units. The first computation step of Tsk_1 and the second computation step of Tsk_4 require the binary semaphore s_1 to enter a critical section, while the second computation step of Tsk_2 and the first computation step of Tsk_3 require control over the binary semaphore s_2.

Figure 2 shows the pTPN model that corresponds to the above specification. Repetitive task releases are represented by always-enabled transitions t_{10}, t_{20}, t_{30}, and t_{40}. Computation steps are represented by transitions t_{12}, t_{13}, t_{21}, t_{23}, t_{33}, t_{34}, t_{41}, and t_{44}, having firing intervals equal to the corresponding min-max range of Execution Time. Wait operations on semaphore s_1 (represented by place s_1) are modeled by transitions t_{11} and t_{43}, while signal operations are represented by transitions t_{12} and t_{44}, which also account for the completion of computation steps. In analogous manner, transitions t_{22} and t_{32} account for wait operations on semaphore s_2 (modeled by place s_2) and the corresponding signal operations are represented by transitions t_{23} and t_{33}. In order to avoid priority inversion, the priority ceiling emulation protocol [33] is assumed, according to which the priority of a task that attempts to acquire a semaphore is raised to the highest priority of any task that ever uses that semaphore. According to this, the priority of tasks Tsk_3 and Tsk_4 is raised to level 3 and 4, respectively, in the sections where they access a critical section. Priority boost operations are modeled by transitions t_{31} and t_{42}, while the corresponding deboost operations are represented by transitions t_{33} and t_{44} which also account for the completion of computation steps.

The Oris Tool also supports modeling of real-time task-sets through timeline schemas, a semi-formal specification based on an intuitive graphical notation which provides modeling convenience and facilitates industrial acceptance. Figure 3 shows the timeline schema that corresponds to the pTPN model of Fig. 2: tasks are characterized by their release interval and their deadline; computations are represented by segments annotated with min-max interval of Execution Time, with resource reclaiming and static priorities; semaphore operations are represented by circles. The Oris Tool supports the automated translation of timeline schemas into pTPN models through the Net&Code Generator.

3.2 Verification through simulation and analysis

A first understanding of dynamic behavior of a model can be obtained through the Petri Nets Animator which executes the model through a token-game (see Fig. 4). Animation can proceed in continuous mode or step-by-step: in the first case, firing times of transitions are randomly chosen by the Animator

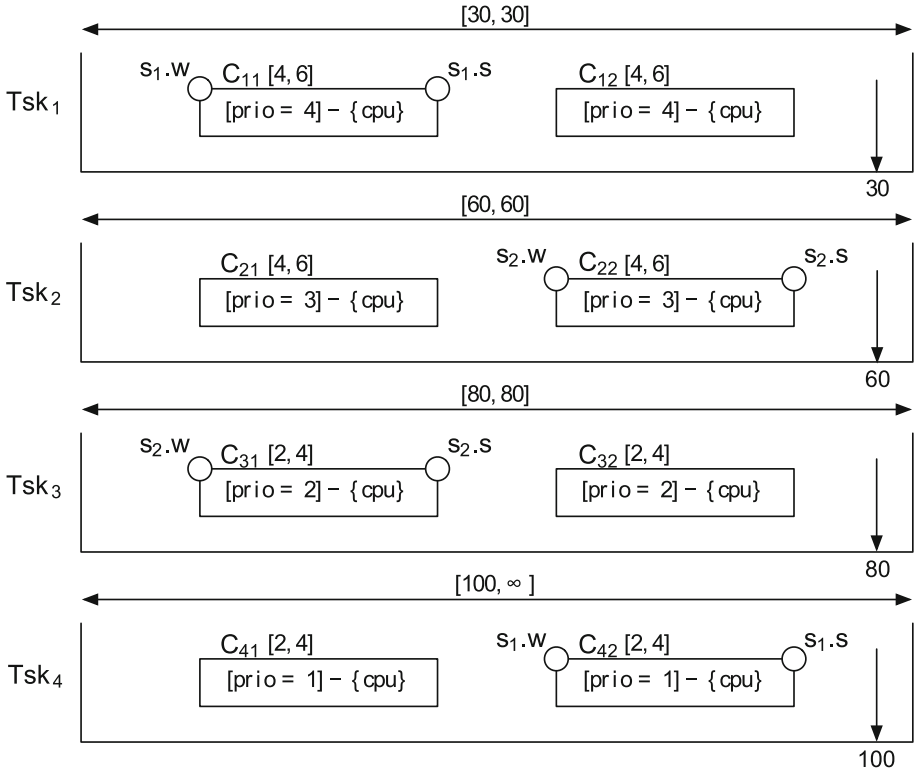

Fig. 3 The timeline schema that corresponds to the pTPN model of Fig. 2

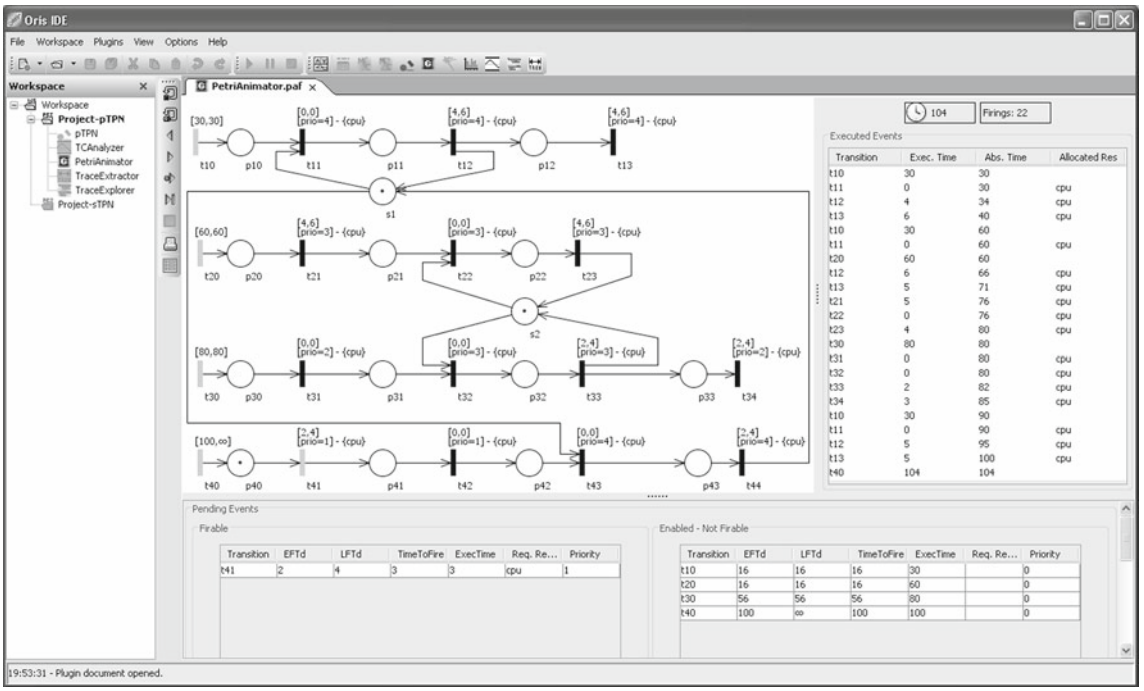

Fig. 4 Animation of the pTPN model of Fig. 2 through the Petri Nets Animator. The *panel on the right* enlists fired transitions, specifying their Execution Time and the corresponding absolute time since the start of the simulation

within their respective firing intervals; in the second case, the user has control over transition Execution Times. The Animator also supports the evaluation of the probability of each reached marking (in a Petri Net, the marking identifies the logical location of the system and identifies the set of enabled transitions) and the mean Execution Time of paths specified in an input document.

Exhaustive verification of the model is obtained only through the enumeration of the whole state space, which requires a symbolic approach. In fact, the state of timed

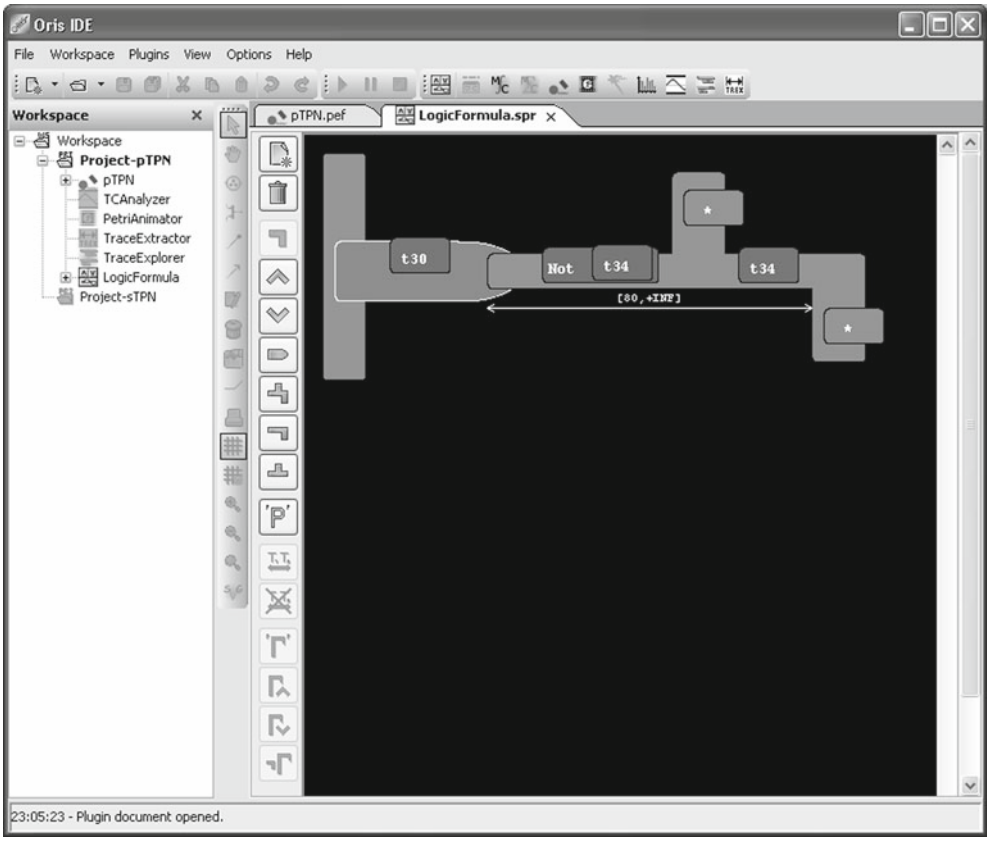

Fig. 5 The Logic Formulae Editor transposes the formal textual syntax of temporal logic formulae into an intuitive graphic representation. With reference to the pTPN model of Fig. 2, the formula identifies the execution traces of task Tsk_3 (i.e., the execution traces that start with a firing of t_{30} and end with the firing of t_{34} without any intermediate firing of t_{34}) with completion time higher than 80 time units

extensions of Petri Nets depends not only on the marking but also on timers associated with transitions. While markings are discrete, timers take values in a dense space. To obtain a discretely enumerable reachability relation, the state space must be partitioned into equivalence classes, each collecting a dense variety of states. This is obtained by collecting the states that are reached through the same firing sequence but with different times; the state space is thus covered using state classes, each represented as a tuple $S = \langle M, D \rangle$ where M is a marking and D is a firing domain which identifies a (dense) set of values for the timers associated with enabled transitions. While the marking M is represented in a straightforward manner, the firing domain D must be encoded as the space of solutions for the set of constraints limiting the timers of enabled transitions. A state class S_0 is reachable from class S through transition t_0 if and only if S_0 contains all and only the states that are reachable from some state collected in S through some feasible firing of t_0. Enumeration of the reachability relation among state classes leads to the construction of the so-called state class graph (SCG) [6,34].

In the analysis of TPN models, firing domains are efficiently encoded as Difference Bounds Matrixes (DBMs) [34,35]. State-space analysis of pTPN models, instead, involves clocks that can advance with non-uniform rate across different logical locations, so as to represent a computation that can be suspended and resumed in subsequent stages of the execution. This requires the enumeration of complex time domains, which can no longer be encoded as DBMs, changing the nature of time and space complexity. In order to avoid exponential complexity in the derivation and representation of state classes, [7] derives the tightest conservative-approximate DBM representation of the state-space using a refinement of the approximation proposed in [23], for which reachability has been proven to be not decidable. The approximate representation of the state-space is still sufficient to identify all the traces that can be critical with respect to requirements pertaining to the logical sequencing or to the quantitative timing of events. Moreover, the constraints encoded in the classes visited by any critical trace are sufficient to re-construct the exact set of timings that are feasible for the trace itself, thus opening the way to the clean-up of false behaviors [7].

The Continuous Time Analyzer performs the analysis of both TPN and pTPN models, implementing the analysis techniques reported in [6,7]. It produces as output the graph of reachable markings and the SCG, both in text and GraphML

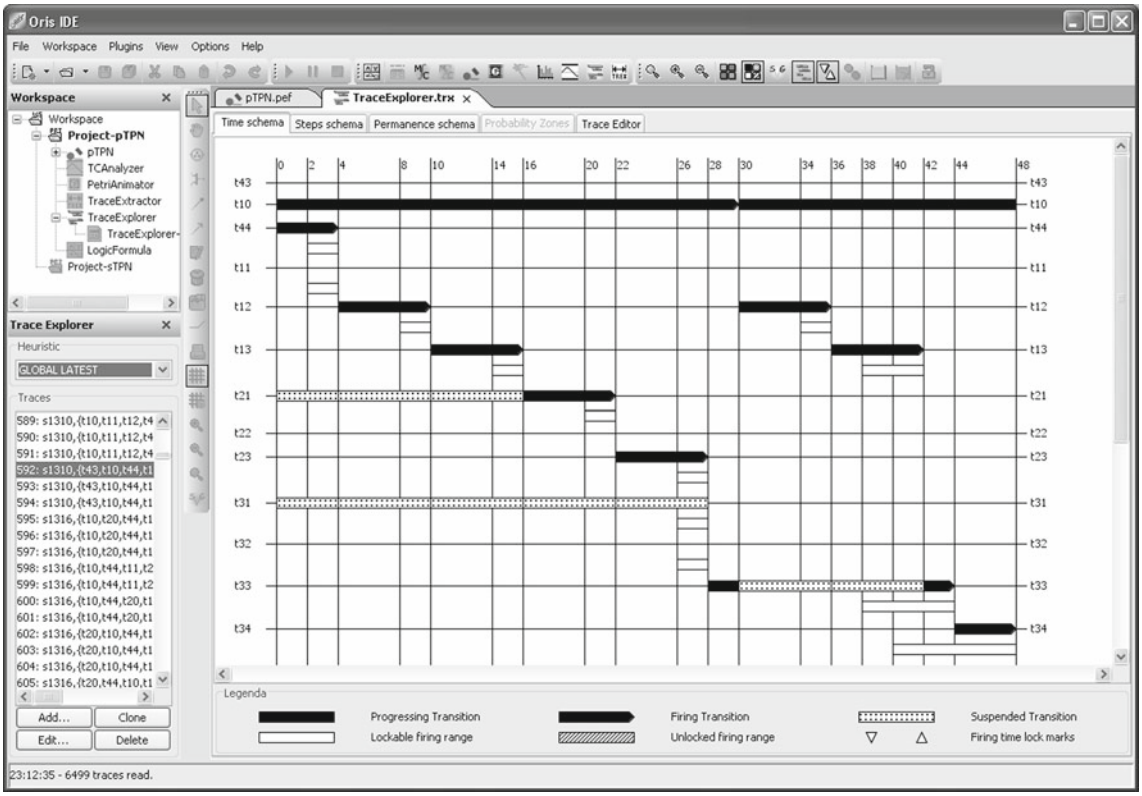

Fig. 6 The Trace Explorer implements different methods to select a profile within the dense set of timings associated with an execution sequence

format (GraphML is an XML-based file format for graphs, which is supported by various graph editors).

3.3 Properties verification through model checking

The SCG obtained by the Continuous Time Analyzer can be used as an input for the procedure of correctness verification through model checking. In the Oris Tool, the Continuous Time Model Checker has the capability of generating all the symbolic paths in the SCG that match a given specification. This, in turn, is formalized using a branching-time temporal logic, which extends the existential subset of Computation Tree Logic (CTL) [36], enabling the expression of conditions on states (markings) and actions (transitions) [37]. In addition, it supports the expression of temporal constraints on operators *Until*, *Eventually* and *Always*, in the style of *Real-Time Temporal Logic* (RTTL) [38].

With reference to the example of Fig. 2, we want to check if it is possible that a new instance of task Tsk_3 is released while a previous instance is still pending. Referring to the pTPN model in Fig. 2, we need to check if the SCG contains any symbolic execution sequence that (i) starts with a firing of t_{30}, (ii) ends after more than 80 time units with the firing of t_{34}, and (iii) does not include an intermediate firing of t_{34}. The following formula identifies such sequence:

$$\mathbf{X}_{t_{30}}((true)_{\neg t_{34}}\mathbf{U}^{[80;+\infty]}_{t_{34}}(true))$$

In the example, the *Next* operator (**X**) is used to find all the state classes that admit the firing of transition t_{30}, which models releases of task Tsk_3. The *Until* operator (**U**) requires that the task instance corresponding to a given release ends no earlier than 80 time units. Note that the formula is expressed on actions, since state-based conditions are set to true. The Continuous Time Model Checker provides all the traces that satisfy the given formula and computes the corresponding minimum and maximum Execution Time. The composition of the logic formula is aided by the Logic Formulae Editor, a visual tool which transposes the formal textual syntax into intuitive visual representation [27], thus reducing usage effort and error frequency (see Fig. 5).

The identification of traces that start with a task release and end with its completion is also supported by the Trace Extractor, which enables the derivation of execution traces that start and end with the firing of two specified transitions. Each trace is associated with minimum and maximum Execution Time, thus permitting to verify whether the task meets its deadline. In the example of Fig. 2, tasks Tsk_1, Tsk_2, Tsk_3, and Tsk_4 have a Worst Case Completion Time (WCCT) of 16, 28, 48, and 52 time units, respectively, and they thus meet their deadlines with minimum laxity of 14, 32, 32, and 48 time units, respectively.

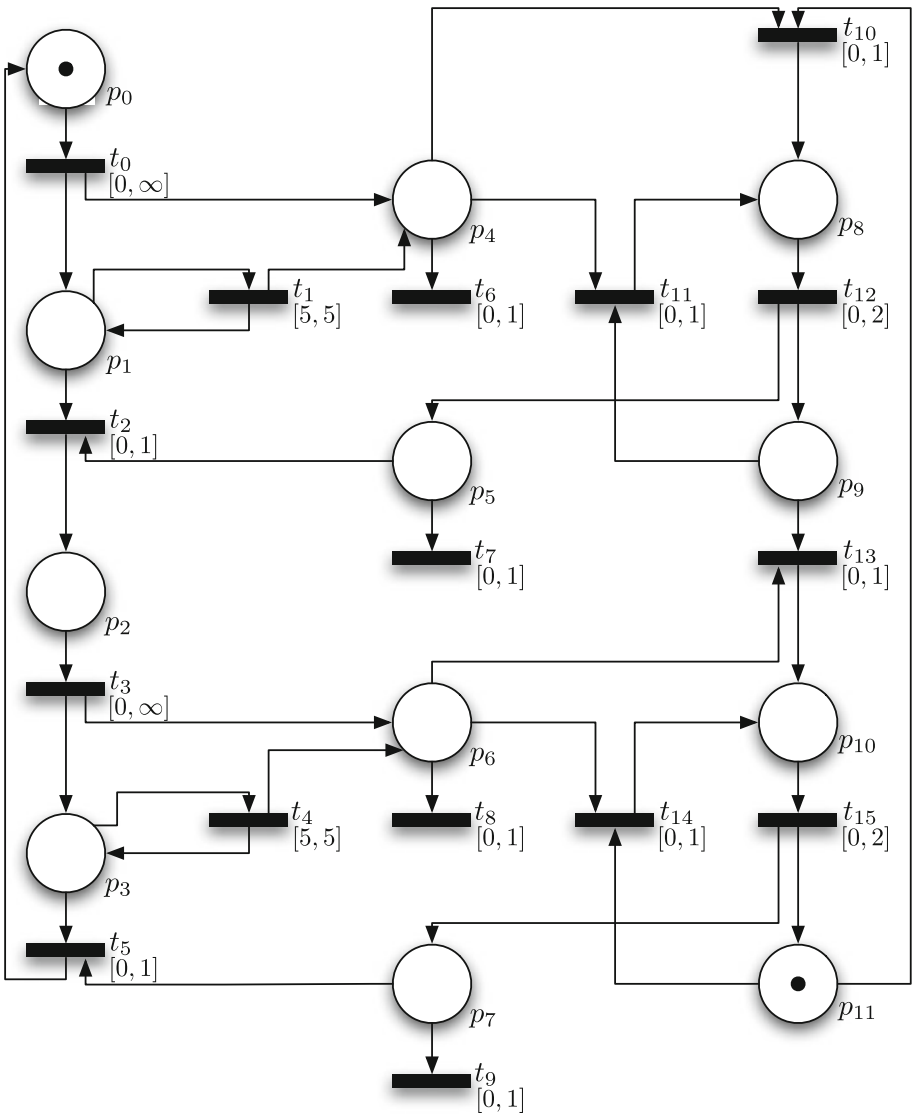

Fig. 7 The sTPN model of the alternating bit protocol: the static firing interval of each transition is associated with a probability density function

3.4 Timeliness analysis of model traces

The dense variety of timings that can be applied to an execution sequence results from the combination of a number of non-deterministic variables, representing the sojourn times in the classes along the trace itself [7]. Each of these variables ranges within a minimum and a maximum value and they may depend on each other. The Trace Explorer enables interactive exploration of these dependencies. In particular, the user can apply four different heuristics to select a timing profile for a trace: *local latest* heuristic constructs the timing profile by always selecting the maximum sojourn time in each visited class; *global latest* selects a timing profile which maximizes the Execution Time of the last event in the trace; *local earliest* and *global earliest* work in similar manner.

With reference to the example of Fig. 2, we consider a symbolic run identified through the Trace Extractor as a case in which task Tsk_3 may attain its WCCT of 48 time units: the trace starts from state class S_{1310}, which is entered through the firing of transition t_{30}, fires the sequence $t_{43}, t_{10}, t_{44}, t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23}, t_{31}, t_{32}, t_{10}, t_{11}, t_{12}, t_{13}, t_{33}, t_{34}$, and finally reaches state class S_{4279}. Figure 6 shows a screenshot of the Trace Explorer illustrating the application of the global latest heuristic to the selected trace. The diagram partitions the temporal axis with vertical lines corresponding to transition firings and, for each transition in the run, shows a timeline which makes evident the periods in which the associated clock has been progressing (solid line) or suspended (dashed line). In the example of Fig. 6, task Tsk_3 undergoes preemption by the other three tasks, and it is first suspended from the start of the trace until time 28 and then from time 30 until time 42. The diagram also visualizes the range of variability of each clock above the horizontal time axis: this range can be repeatedly restricted through the introduction of

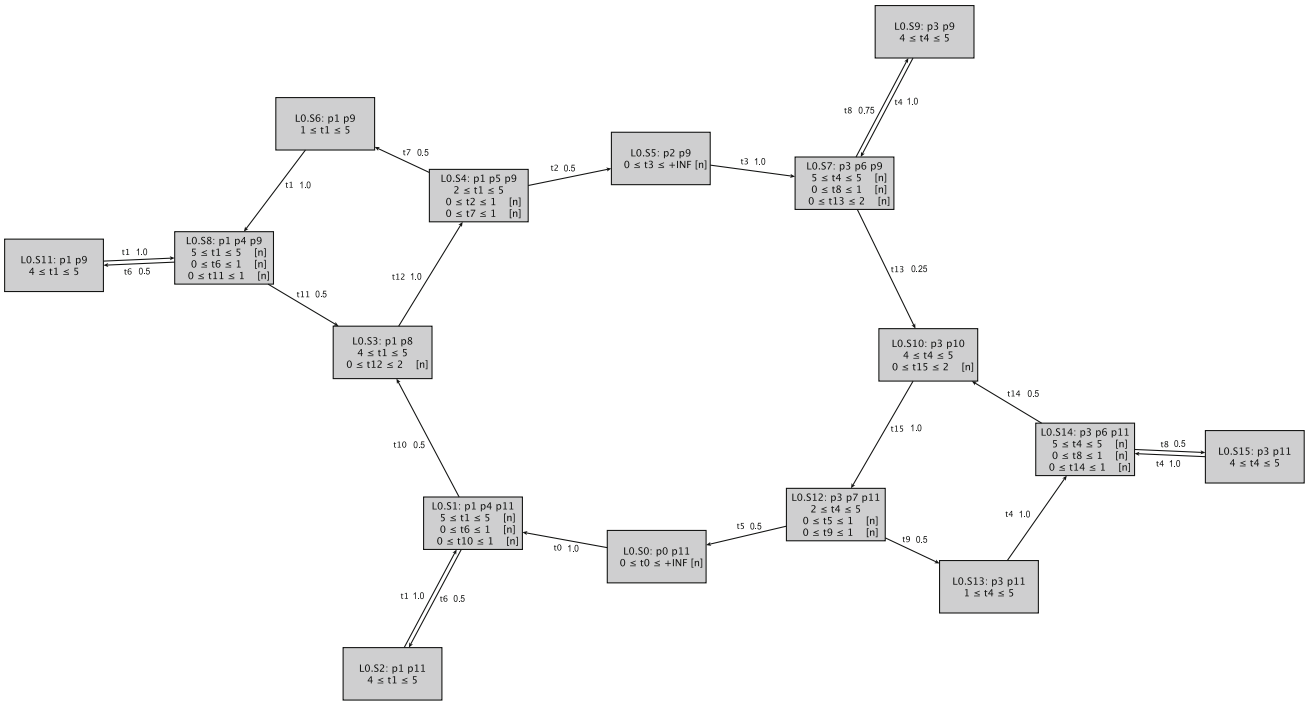

Fig. 8 The SSCG of the sTPN model of Fig. 7

a lock which forces any clock to take a specific value within its acceptable interval or to take values within a subset of its acceptable interval, thus reducing the range of variability of the profile caused by dependencies among clocks. In the example of Fig. 6, transition t_{34} can fire within [40, 48] time units since the start of the trace; however, the firing of transition t_{33} at time 44 restricts the range of variability of the Execution Time of t_{34} to [46, 48] time units.

3.5 Code generation and Execution Time profiling

A pTPN model can be implemented on top of the primitives of an RTOS. The Net&Code Generator supports the automated translation of the timeline schema of a real-time task-set into C-code running under RTAI [32]. As characterizing features, the architecture of the code has a readable structure and preserves semantic properties of the corresponding pTPN model. This leaves the developer full control over the code and makes the model an oracle against which code executions can be compared.

The Net&Code Generator also supports code instrumentation, in order to obtain a time-stamped log of events that correspond to transition firings in the pTPN model. The Petri Nets Simulator supports the reconstruction of the Execution Time of each computation step through a measurement-based approach in interrupted mode [39], performs off-line evaluation of execution logs, and provides a measure of attained coverage in the state-space, thus enabling conformance verification with respect to sequencing and timing requirements.

Finally, the Net&Code Generator supports the implementation of a busy-sleep function, which is employed to emulate computation steps in the first stages of development, when entry-point methods that implement functional behavior are not yet available. As the development process advances, entry-points become available and they are separately plugged into the emulated architecture. This supports unit testing of individual computation steps and enables their incremental integration testing.

4 Performance evaluation through sTPNs

TPNs do not allow the characterization of feasible behaviors with a measure of probability, an essential step towards dependability and performance evaluation. To overcome this limitation, Vicario et al. [17] defines the formalism of sTPNs by extending TPNs through the association of the static firing interval of each transition with a (dense) probability density function. Syntax, semantics, and steady-state analysis of sTPNs are formally treated in [17,18]. Here, we report an informal description which is instrumental to illustrate the application of the Oris Tool to performance evaluation of systems.

4.1 Specification of sTPN models

As an example, we consider the case of the Alternating bit protocol, which has been largely addressed in the literature of TPNs. In particular, we present the sTPN extension of the

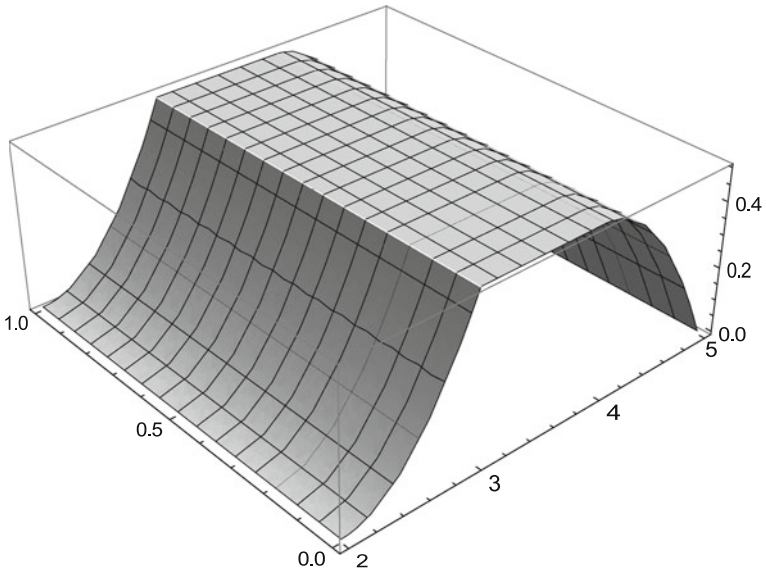

Fig. 9 The state density function $f_4(\tau_1, \tau_2, \tau_7)$ of stochastic state class S_4 of the sTPN model of Fig. 7, drawn under the assumption that τ_7 is equal to 0

TPN model presented in [5], in which the protocol is verified through an enumerative approach.

The protocol delivers messages and acknowledgments between two entries using a non-reliable transmission channel. Recovery of both messages and acknowledgments are ensured by a timeout-and-retransmitting mechanism. The acceptance of duplicate messages is avoided by numbering messages with modulo 2 sequence and marking the acknowledgment with the same number of the packet received. The model is shown in Fig. 7. Transitions t_0, t_3, t_{12}, t_{15} are responsible for sending messages and acknowledgments; transitions t_{10}, t_{11} and t_{13}, t_{14} model the rejection/release of a received message, respectively; transitions t_2 and t_5 represent the receipt of acknowledgments; finally, transitions t_6 through t_9 model message/acknowledgment loss. We extend the original TPN model by assuming exponential distributions for transitions modeling message releases and uniform distributions for any other transition. The Petri Nets Editor supports the specification of sTPN models, allowing the selection of a probability density function for each transition.

4.2 Performance evaluation through stochastic analysis

The analysis of sTPN models requires the derivation of a density function which characterizes the probability of individual timings comprised within the boundaries of a DBM time domain. This leads to the introduction of *stochastic state classes*, which extend state classes and their reachability relation by enriching firing domains with a state probability density function [17,20,21]. According to this, algorithms for detection and computation of class successors are combined with the evaluation of successor probability and with the symbolic derivation of state density functions: a closed-form calculus supports efficient analysis for the class of models with timers that are immediate, deterministic, and expolynomially distributed. The resulting stochastic state class graph (SSCG) is thus an extension of the SCG of the underlying TPN model, that maps each state class on one or more stochastic state classes.

The SSCG is instrumental not only to the verification of correctness properties pertaining to the logical sequencing and the qualitative timing of events, but also to the derivation of performance measures. In fact, the theory of stochastic state classes enables the evaluation of the probability of selected execution traces and the symbolic derivation of the distribution of their feasible timings, and supports steady-state analysis based on Markov Renewal Theory.

An approximate analysis technique is developed for models where all timers with infinite support have exponential distribution [18]. Under this assumption, multivariate distributions associated with stochastic state classes are approximated through Bernstein Polynomials [40,41], enabling the analysis of models where the enumeration of the state space is unfeasible due to practical factors of explosion or to the presence of cycles with overlapping activity of non-Markovian transitions.

The Continuous Time Analyzer implements the analysis techniques of [17,18], relying on a C++ library that implements the symbolic calculus of integral functions under the assumption that static probability density-functions of non-deterministic transitions are expolynomial functions. Both textual and GraphML representations of the SCG and the SSCG are provided, facilitating the identification of stochastic state classes having the same marking and temporal domain but different state density functions. In the textual

representation of the SSCG, state density functions are represented according to the format used by Mathematica [42].

In the example of Fig. 7, the analysis method described in [17] leads to the enumeration of 16 stochastic state classes. Figure 8 shows the SSCG of the model. As a case example, the state density function of stochastic state class S_4 is illustrated in Fig. 9.

The DBM time domain D_4 of class S_4 is

$$D_4 = \begin{cases} 2 \le \tau_1 \le 5 \\ 0 \le \tau_2 \le 1 \\ 0 \le \tau_7 \le 1 \end{cases}$$

where τ_1, τ_2, and τ_7 represent the times to fire of transitions t_1, t_2, and t_7, respectively. The state density function $f_4(\tau_1, \tau_2, \tau_7)$ has piece-wise representation over DBM subdomains $D_{4.1}$, $D_{4.2}$, and $D_{4.3}$ of D_4:

$$f_4(\tau_1, \tau_2, \tau_7) = \begin{cases} 2 - 2\,\tau_1 + 0.5\,\tau_1^2 & \text{if} \quad (\tau_1, \tau_2, \tau_7) \in D_{4.1} \\ 0.5 & \text{if} \quad (\tau_1, \tau_2, \tau_7) \in D_{4.2} \\ -7.5 + 4\,\tau_1 - 0.5\,\tau_1^2 & \text{if} \quad (\tau_1, \tau_2, \tau_7) \in D_{4.3} \end{cases}$$

where

$$D_{4.1} = \begin{cases} 2 \le \tau_1 < 3 \\ 0 \le \tau_2 \le 1 \\ 0 \le \tau_7 \le 1 \end{cases} \quad D_{4.2} = \begin{cases} 3 \le \tau_1 < 4 \\ 0 \le \tau_2 \le 1 \\ 0 \le \tau_7 \le 1 \end{cases}$$

$$D_{4.3} = \begin{cases} 4 \le \tau_1 \le 5 \\ 0 \le \tau_2 \le 1 \\ 0 \le \tau_7 \le 1 \end{cases}$$

On the one hand, the SSCG is employed in the verification of correctness properties. For instance, we can verify that the timeout is correctly set so as to guarantee that only one message/acknowledgment is pending at a time and that no duplicate message is sent. This is easily proven by checking that, in any marking, all places hold at most one token.

On the other hand, the SSCG supports the evaluation of performance measures on the model under consideration. Since each edge is associated with a firing probability, it is possible to evaluate the probability of any execution trace of the model. For instance, the execution sequence $\rho = S_1 \xrightarrow{t_{10}} S_3 \xrightarrow{t_{12}} S_4 \xrightarrow{t_2} S_5$, which corresponds to the successful delivery of a message and the corresponding acknowledgment, has a probability of 0.25.

The Continuous Time Analyzer also provides the probability density function (in Mathematica format) of the completion time of any path between any two regeneration classes. Figure 10 illustrates the probability density function

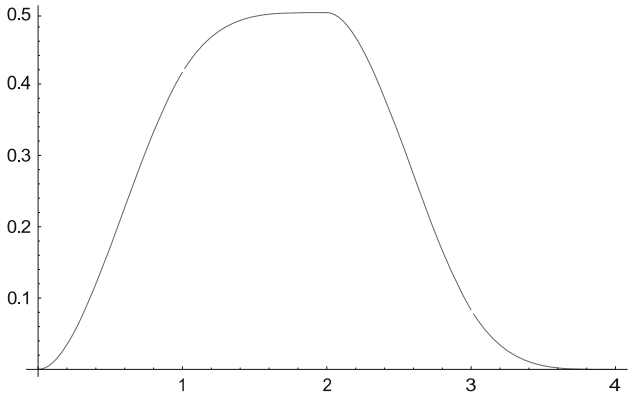

Fig. 10 The probability density function $f(x)$ of the completion time of the trace $S_1 \xrightarrow{t_{10}} S_3 \xrightarrow{t_{12}} S_4 \xrightarrow{t_2} S_5$ in the SSCG of the sTPN model of Fig. 7

$f(x)$ of the completion time of the trace ρ:

$$f(x) = \begin{cases} x^2 - 0.667x^3 + 0.083x^4 \\ \quad \text{if } x \in [0, 1) \\ -0.833 + 2.667x - 2x^2 + 0.667x^3 - 0.0833x^4 \\ \quad \text{if } x \in [1, 2) \\ -10.167 + 14.667x - 7x^2 + 1.333x^3 - 0.0833x^4 \\ \quad \text{if } x \in [2, 3) \\ 21.333 - 21.333x + 8.0x^2 - 1.333x^3 + 0.0833x^4 \\ \quad \text{if } x \in [3, 4] \end{cases}$$

5 Conclusions

The Oris Tool provides a comprehensive environment that integrates a rich set of modules supporting specification and analysis of various classes of reactive timed systems. As a characterizing trait, it manages both preemptive and stochastic models, enabling the integration of qualitative verification and quantitative evaluation. Oris is mainly a research tool, intended to comprise a framework where theoretical developments are put to the test of experimentation. In the years, this has addressed many different scenarios, and it has been applied to solve various classes of problems and to support several activities of real practice processes.

Theoretical results pertaining to the analysis of preemptive models [6,7,16] are largely consolidated now and by far ready for application. In the development of real-time preemptive systems, the Oris Tool supports in organic manner

verification, implementation, and testing activities [25–27]. These include modeling based on a semi-formal specification and automated translation into the corresponding pTPN model, schedulability analysis, automated generation of real-time code running under RTAI, a simple yet effective measurement based approach to Execution Time profiling, test-case selection and execution, evaluation of executed tests and coverage measure. These development practices have been brought to application and concretely experimented in the industrial context of Galileo Avionica-Firenze, as a part of the SW Initiative of the Finmeccanica Group under the research line "Design and Verification Methods for Concurrent Real Time SW". Recently, they have also been applied to the construction of real-time hierarchical scheduling systems, which provide partitioning and reduction of complexity and thus open the way to cope with state space explosion in the analysis of complex systems.

Stochastic analysis [17,18,20,21] has a more recent history and the development effort has mainly focused on issues and techniques to overcome practical limitations. We are presently working on various directions that aim at supporting probabilistic test-case selection and sensitization, derivation of quantitative reward measures, monitoring of real-time systems [43,44].

Acknowledgments The Oris Tool is the result of a joint effort of best practices of SW design and theory of qualitative verification and quantitative evaluation at the Software Technologies Laboratory of the University of Florence. Among these we shall mention at least Luigi Sassoli, Francesco Poli, Irene Bicchierai, Jacopo Torrini, and Marco Lusini.

References

1. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: 17th ICALP (1990)
2. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL: a tool-suite for automatic verification of real-time systems. In: Hybrid Systems III: LNCS, 1066 (1996)
3. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Hybrid Systems III, 1066. Springer (1995)
4. Merlin, P., Farber, D.: Recoverability of communication protocols. IEEE Trans. Commun. **24**(9), 1036–1043 (1976)
5. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. IEEE Trans. Softw. Eng. **17**(3), 259–273 (1991)
6. Vicario, E.: Static analysis and dynamic steering of time dependent systems using Time Petri Nets. IEEE Trans. Softw. Eng. **27**(1), 728–748 (2001)
7. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Timed state space analysis of real time preemptive systems. IEEE Trans. Softw. Eng. **30**(2), 97–111 (2004)
8. Altisen, K., Gossler, G., Pnueli, A., Sifakis, J., Tripakis, S., Yovine, S.: A framework for scheduler synthesis. In: RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium, Washington, DC, USA, 154 pp. IEEE Computer Society (1999)
9. Bertin, V., Closse, E., Poize, M., Pulou, J., Sifakis, J., Venier, P., Weil, D., Yovine, S.: Taxys = Esterel + Kronos. A tool for verifying real-time properties of embedded systems. In: CDC '01: Proceedings of the 40th IEEE Conference on Decision and Control (2001)
10. Kloukinas, C., Yovine, S.: Synthesis of safe, QoS extendible, application specific schedulers for heterogeneous real-time systems. In: In ECRTS 03: Euromicro Conference on Real-Time Systems, pp. 287–294. IEEE Computer Society Press (2003)
11. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: TIMES: a tool for schedulability analysis and code generation of real-time systems. In: Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems (2003)
12. Gardey, G., Lime, D., Magnin, M., Roux, O.: Roméo: a tool for analyzing time petri nets. In: 17th International Conference on Computer Aided Verification (CAV́c605). Lecture Notes in Computer Science (2005)
13. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA—construction of abstract state spaces for petri nets and time petri nets. Int. J. Prod. Res. **42**(14), 2741–2756 (2004)
14. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer-Verlag (2004)
15. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Times—a tool for modelling and implementation of embedded systems. LNCS **2280**, 460–464 (2002)
16. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Modeling flexible real time systems with preemptive Time Petri Nets. In: Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS03) (2003)
17. Vicario, E., Sassoli, L., Carnevali, L.: Using stochastic state classes in quantitative evaluation of dense-time reactive systems. IEEE Trans. Softw. Eng. **35**(5), 703–719 (2009)
18. Carnevali, L., Grassi, L., Vicario, E.: State-density functions over DBM domains in the analysis of non-Markovian models. IEEE Trans. Softw. Eng. **35**(2), 178–194 (2009)
19. Horváth, A., Vicario, E.: Aggregated stochastic state classes in quantitative evaluation of non-markovian Stochastic Petri Nets. In: Proceedings of the 6th Int. Conf. on Quant. Evaluation of Sys. (QEST '09) (2009)
20. Sassoli, L., Vicario, E.: Close form derivation of state-density functions over DBM domains in the analysis of non-Markovian models. In: Proc. of the 4th Int. Conf. on the Quant. Evaluation of Sys. (QEST) (2007)
21. Bucci, G., Piovosi, R., Sassoli, L., Vicario, E.: Introducing probability within state class analysis of dense time dependent systems. In: Proc. of the 2nd Int. Conf. on Quant. Evaluation of Sys. (QEST '05) (2005)
22. Carnevali, L., Ridi, L., Vicario, E.: Partial stochastic characterization of timed runs over DBM domains. In: Proc. of the 9th International Workshop on Performability Modeling of Computer and Communication Systems (2009)
23. Cassez, F., Larsen, K.G.: The Impressive Power of Stopwatches. LNCS, vol. 1877 (August, 2000)
24. Roux, O.H., Lime, D.: Time petri nets with inhibitor hyperarcs: formal semantics and state-space computation. In: 25th Int. Conf. on Theory and Application of Petri nets, vol. 3099, pp. 371–390 (2004)
25. Carnevali, L., Grassi, L., Vicario, E.: A tailored V-Model exploiting the theory of preemptive Time Petri Nets. In: Ada-Europe '08: Proc. of the Ada-Europe Int. Conf. on Reliable Software Technologies, pp. 87–100. Springer-Verlag, Berlin (2008)
26. Carnevali, L., Sassoli, L., Vicario, E.: Sensitization of symbolic runs in real-time testing using the ORIS tool. In: Proc. of the IEEE Conf. on Emerging Technologies and Factory Automation (ETFA) (2007)

27. Carnevali, L., Sassoli, L., Vicario, E.: Casting preemptive Time Petri Nets in the development life cycle of real-time software. In: Proc. of the Euromicro Conference on Real-Time Systems (2007)
28. Bucci, G., Sassoli, L., Vicario, E.: Correctness verification and performance analysis of real time systems using stochastic preemptive Time Petri Nets. IEEE Trans. Softw. Eng. **31**(11), 913–927 (2005)
29. Vicario, E.: Engineering the usability of a visual formalism for real-time temporal logic. J. Vis. Lang. Comput. **12**(6), 573–599 (2001)
30. Lusini, M., Vicario, E.: Design and evaluation of a visual formalism for real time logics. In: ACoS '98/VISUAL '98, AIN '97: Selected Papers on Services and Visualization: Towards User-Friendly Design, pp. 158–173. Springer-Verlag, London (1998)
31. Carnevali, L., D'Amico, D., Ridi, L., Vicario, E.: Automatic code generation from real-time systems specifications. In: Proceedings of the IEEE/IFIP International Symposium on Rapid System Prototyping (RSP) (2009)
32. Dept. of Aerospace Eng. of the Polytechnic of Milan: RTAI: Real Time Application Interface for Linux. https://www.rtai.org
33. Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans. Comput. **39**(9), 1175–1185 (1990)
34. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time Petri nets. In Mason, R.E.A. (ed.) Information Processing: Proceedings of the IFIP Congress 1983, vol. 9, pp. 41–46. Elsevier (1983)
35. Dill, D.: Timing assumptions and verification of finite-state concurrent systems. In: Proceedings of Workshop on Computer Aided Verification Methods for Finite State Systems (1989)
36. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
37. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G.: An action-based framework for verifying logical and behavioural properties of concurrent systems. Comput. Netw. ISDN Syst. **25**(7), 761–778 (1993)
38. Alur, R., Henzinger, T.A.: Logics and models of real-time: a survey. In: Real Time: Theory in Practice, vol. 600, pp. 74–106. Springer-Verlag (1991)
39. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Statshulat, J., Stenstroem, P.: Priority inheritance protocols: the worst case execution-time problem: overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 1–53 (2008)
40. Lorentz, G.: Bernstein Polynomials. University of Toronto Press, Toronto (1953)
41. Sauer, T.: Multivariate Bernstein polynomials and convexity. Comput. Aided Geom. Des. **8**(6), 465–478 (1991)
42. Wolfram Research: Mathematica 5.2. www.wolfram.com
43. Carnevali, L., Ridi, L., Vicario, E.: Stochastic fault trees for cross-layer power management of wsn monitoring systems. In: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA) (2009)
44. Bucci, G., Carnevali, L., Vicario, E.: A tool supporting evaluation of non-markovian fault trees. In: Proceedings of the 5th Int. Conf. on Quant. Evaluation of Sys. (QEST '05) (2008)