ORIS: a tool for state-space analysis of real-time preemptive systems

G.Bucci, L.Sassoli, E.Vicario

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

Abstract

Formal methods based on state-space enumeration, such as Timed Automata and Time Petri Nets (TPN), have been proposed for designing and validating reactive real-time systems. The great expressiveness of these methods is counterbalanced by the increased complexity of the analysis, which may grow exponentially. Furthermore, the enumerated state-space needs to be inspected to identify critical behaviors with respect to sequencing and timing requirements. This naturally leads to the implementation of tools supporting the different stages of the development process.

In this paper we present Oris, an environment for building, simulating, analyzing and validating complex real time systems specified in terms of an extended TPN formalism, named Preemptive Time Petri Nets. Oris includes not only the state-space enumeration engine, but also a number of modules which ease user interaction, and make it usable also by a designer with no specific experience in formal modelling.

1 Introduction

In the practice of real time systems, the problem of logical sequencing and quantitative timing of events is usually formulated under the assumption of a number of hypotheses on the structure of the tasking set. These assumptions simplify the complexity of sequencing and enable analytical techniques for estimating the worst-case response time in low-order polynomial complexity. The theory of Rate Monotonic scheduling is a notable example of the approach [13]. This theory can also deal with issues such as inter-task dependencies, due to mutual exclusion on shared resources, and to dataflow precedence relations [15]. However it does not consider non-deterministic computation times; sporadic tasks and/or tasks with mutual dependencies in the time of release; multiple processors.

Under these conditions, predictability in both sequencing and timing may become difficult enough to motivate the adoption of state-space analysis methods based on models such as Timed Automata [1] [5] [11], or Time Petri Nets [14] [6] [16] [8], all of which pay a price in terms of complexity for the greater expressiveness.

For all these models, 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. According to this, to obtain a discrete representation, the state-space must be covered through *equivalence classes* each characterized by a *time domain* collecting a dense variety of clock values [1] [5] [11] [6] [16].

Following these approaches, the validation process requires the enumeration of the state-space of the model. In practice, this is done through an analysis engine capable of generating reachability relations among state classes of a model. The enumerated state-space can be inspected through model checking techniques, in order to identify critical behaviors with respect to sequencing and timing requirements which, in turn, can be formulated as temporal logic formulae.

A number of tools [5] [11] have been proposed based on Timed Automata. In our perspective, the Times tool [3] has particular relevance, since it attempts to bridge scheduling theory and automata-theoretic approaches to system modelling and analysis, supporting the treatment of tasks with asynchronous and dense release times, running under the most practiced scheduling disciplines. However, the underlying analysis still rules out nondeterministic computation times taking values within dense intervals.

In [16], the Time Petri Net model (TPN) has been adopted as the formalism to deal with reactive systems. The basic model has been improved in [8], so as to encompass preemptive scheduling. The resulting model has been named Preemptive Time Petri Nets (PTPN). This has led to the development of an analysis engine named Oris, which, in the last years, has been supplemented with a number of tools supporting the user throughout the design process: from the initial stage of net specification to the final stage of validation. As a result, we now use the name Oris to

Proceedings of the First International Conference on the Quantitative Evaluation of Systems (QEST'04)
0-7695-2185-1/04 $ 20.00 IEEE

denote the entire environment. Oris allows representation of complex tasking models scheduled by priority on multiple processors, including periodic and sporadic tasks, with synchronization on exclusive resources and precedence constraints, with nondeterministic computation times. In this paper, we give a description of the Oris environment, referring to a case study which is thoroughly expounded through a complete design cycle.

The paper is organized as follows. Section 2 describes the functionalities of the various modules of Oris, as well as their interactions. Section 3 illustrates how these modules support the modelling and validation process referring to the case study of a real-time system which includes complexity factors that deny its treatment through analytical techniques. Section 4 presents the results of analysis. Conclusions are drawn in Section 5.

2 Description of the Toolset

ORIS comprises a rich set of tools for building, simulating, analyzing and validating Time Petri Net models.

Figure 1 schematizes the validation process in the style of a dataflow diagram. Each bubble represents a phase of the process and reports the name of the module that supports it. The figure also shows the data structures that are generated at different stages. It does not present the input/output interactions with the user, since these are intrinsically associated with the modules that accept user data and/or present results.

A typical design cycle starts with the user specifying the tasking set. This is done with the Timelines Editor; alternatively, the user can directly specify a TPN model through the TPN Editor. Once the model has been built, the analyst can either simulate or analyze it. Simulation is in two forms: the first consists of animating the model, the second in performing a conventional batch simulation to produce a file of statistic data. The analysis is carried out through the TPN Analyzer engine, which generates a graph representing reachability relations among state classes. Correctness verification is carried out through a model checker which finds out all the symbolic traces that are compliant with both logical sequencing and quantitative timings specified through temporal logic formulae. These, in turn, can be laid down through the interactive graphic editor named Formed.

We now give a more detail description of the modules contained in the Oris environment. In the following discussion, we use the acronym TPN to denote both the basic model and the extended one.

Timelines Editor: A graphical interface permitting intuitive description of complex tasking sets in the timeline formalism. It automatically generates TPN models encoded in XML format. A relevant feature of this module is the automatic generation of the Petri net corresponding to the timeline-style description of the tasking set. This permits use of formal methods even to those designers that have no specific experience in using them.

TPN Editor: As an alternative to the previous module, the TPN Editor supports direct model construction in the TPN formalism. The specified models are encoded in the same format used by the Timelines Editor. As a result, a TPN model can be shared between Timelines Editor and TPN Editor, thus permitting the *round trip* mechanism between the two modules (Figure 1).

The TPN Editor always accepts a TPN model generated by the Timelines Editor, but the opposite is not always true, since a TPN model can not always be translated into a timeline description.

TPN Animator: Supports the simulation of TPN models by playing the token-game, i.e. moving tokens on the firing of transitions. It provides a representation of the overall state of the net by highlighting the state of every transition.
Animation can proceed in continuous mode or step-by-step. In the first case, transition firing times are chosen randomly by the TPN Animator within the firing interval. In the second case the user has control over transition execution times.
It is possible to save the log of all transition firing events and related token distribution.

TPN Simulator: Performs simulation in batch mode. The result of simulation is a file of statistic data such as: token distribution, transition firing sequences, and other measures of interest.

TPN Analyzer: The core of the entire system. It has the duty of building the reachability graph of TPN models. Since TPNs have a dense time semantics, in order to obtain a discrete enumeration of the state-space, individual states are collected into state classes each characterized by a logical location (i.e. a marking) and by a dense variety of clock assignments [7]. Dense clock domains are efficiently encoded into a set of linear inequalities each composed of two terms. This kind of encoding, usually referred to as *Difference Bounds Matrix* (DBM) [12], permits representation of classes with space complexity O(N^2) (where N stands for the number of active clocks, i.e. the number of enabled transitions).

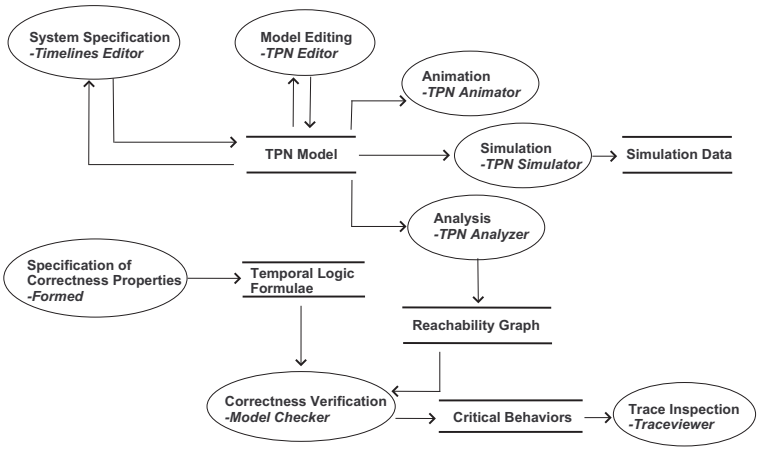

Figure 1. The different phases of the specification-validation process represented in the style of a dataflow diagram. Each phase is labelled with the name of the Oris module that supports it.

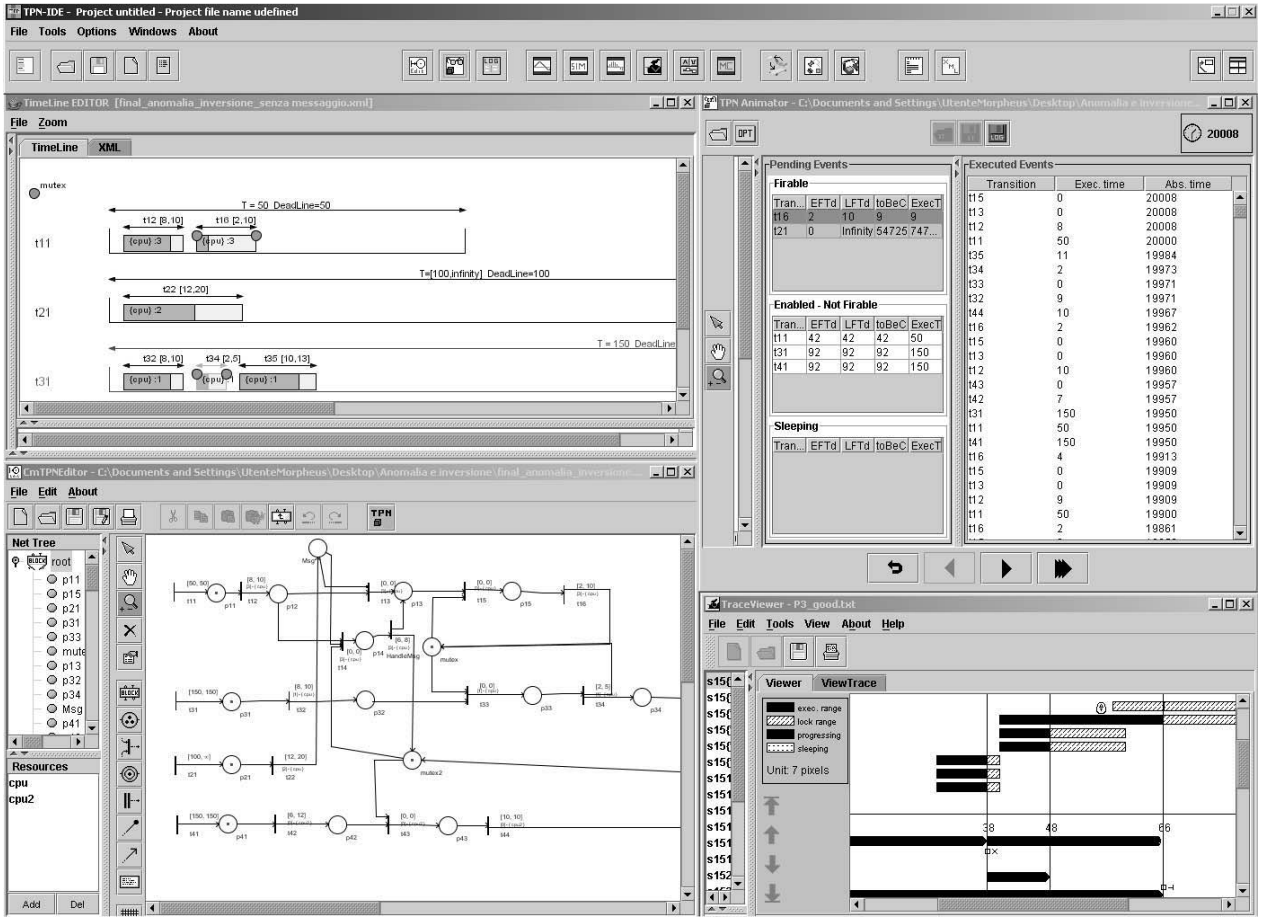

Figure 2. A screenshot of the ORIS environment.

Their manipulation amounts to an all shortest path which is efficiently solved in time O(N^3) [12], or even O(N^2) when it is repeatedly applied within an enumeration algorithm[16].
When the basic TPN model is extended to deal with preemption, state-space analysis 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 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 the exponential complexity in the derivation and representation of state classes, the tightest conservative-approximate DBM representation of the state-space of the model is derived using a refinement of the approximation proposed in [10], for which reachability has been proven to be not decidable. This implies that there is no guarantee that enumeration of a bounded PTPN model always terminates (thought we always experienced termination).

The approximation may introduce a number of false behaviors, which are not congruent with the semantics of the model. However, 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 and to the interactive exploration of feasible timings [8]. This technique is implemented through a specific module, called Tracelyzer (not shown in Figure 1).


Formed: A graphical interface for editing temporal logic formulae. We have developed this interactive tool to ease the composition of clauses in temporal logic, avoiding the complexity of its syntax. We adopted a visual formalism which transposes the formal textual syntax into intuitive visual representation. This reduces usage effort and error frequency, increasing the usability of formal methods, and improving their verification power through a potentially richer user involvement.


Model Checker: Supports correctness verification with respect to a time-linear variant of Real Time Temporal Logic (RTTL) [2]. The Model checker works on the reachability graph generated by the TPN Analyzer. An original aspect of this model checker is the capability to find not only a counterexample, but all the symbolic traces (we call them witnesses) which satisfy a given formula. Each state class is labelled with the set of witnesses which, starting from the class, satisfy the formula. In fact, we are not only interested in knowing wether or not a given property is verified, but also in giving a quantitative measure of how the system satisfies the property. For instance, when referring to a schedulability problem, we may be interested in knowing whether or not a given deadline is met, as well as determining which is the worst completion time within the deadline itself.

In the case of preemptive models, since the enumeration analysis may introduce false behaviors and approximate time profiles, the identified traces can be cleaned up by Tracelyzer.


Trace Viewer: Supports the visualization of the traces extracted by the model checker. The dense variety of timings that can be applied to a trace results from the combination of a number of non-deterministic variables, representing the dwelling times in the classes along the trace itself [8]. Each of these variables ranges within a minimum and a maximum value and may depend on each other. The Trace Viewer module enables interactive exploration of these dependencies for any trace identified in the state class graph through the model checker.

Figure 2 shows a screenshot which includes most of Oris components at work. The content of the various windows is explained in the sequel.

3 Petri Net Model Creation, Analysis and Validation

In this section we introduce the reader to the usage of Oris, by guiding him/her in a tour starting with the creation of a TPN model, going through simulation and analysis, and terminating with model validation. To this end, we develop a case study of real-time system which includes complexity factors that deny its treatment through analytical techniques.

3.1 System specification and Model Editing

We consider a system comprised of four processes P_1, P_2, P_3 and P_4, internally structured as sequences of computation steps, each characterized by a minimum and maximum computation time and pre-allocation on two processors $cpu1$ and $cpu2$. The initial specification is as follows:

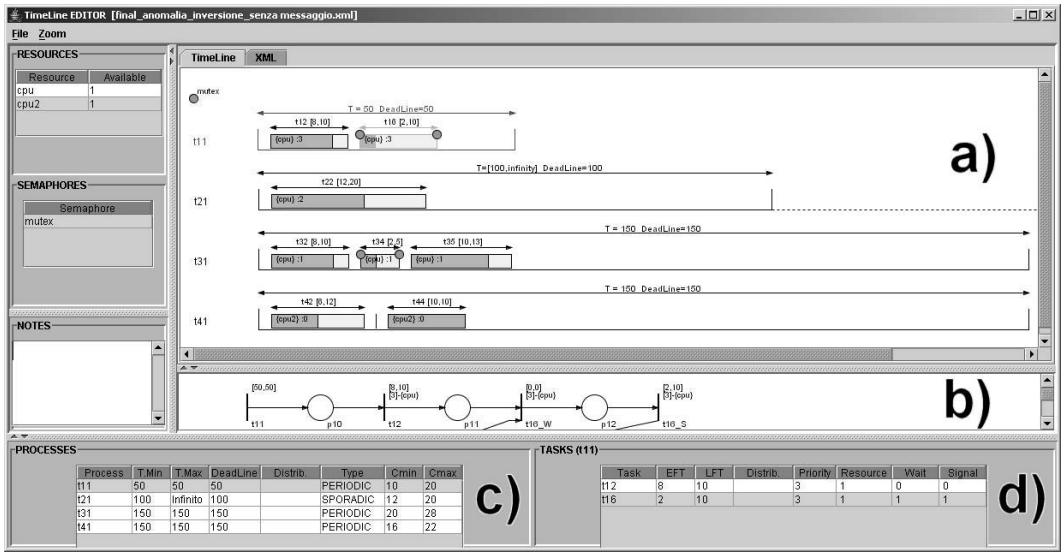

Figure 3. Screenshot of the Timelines Editor representing the tasking set of the case study.

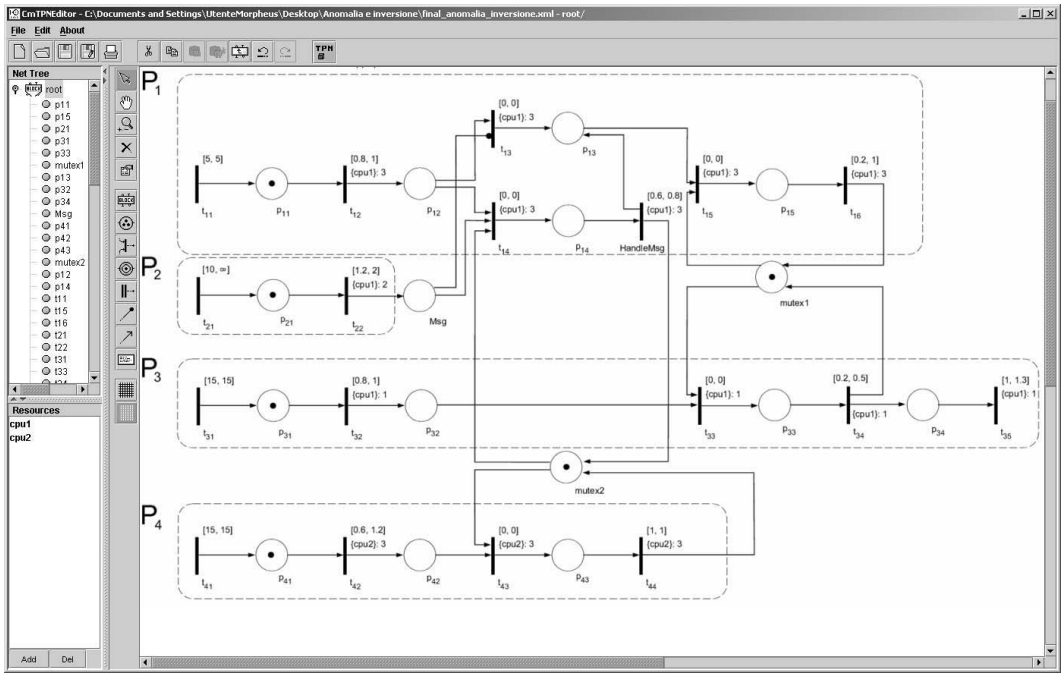

Figure 4. The PTPN model reworked through the TPN Editor.

- P_1 is periodic with period of 5 time units, it runs at priority level 3, and is composed of two sequential steps; the second one requires semaphore $mutex1$ to enter a critical section. P_1 runs on $cpu1$.

- P_2 is sporadic, with a minimum inter-arrival time of 10 time units, it runs at priority level 2. P_2 runs on $cpu1$.

- P_3 is periodic with period of 15 time units, it runs at priority level 1, and it is composed of three sequential steps, the second one requires control over semaphore $mutex1$. P_3 runs on $cpu1$.

- P_4 is a periodic process with period of 15 time units running at priority level 3. It is composed of two sequential steps. P_4 runs on $cpu2$.

All steps, except the second part of process P_4, have nondeterministic computational times. With regard to priorities, the greater the number the higher the priority.

Usually, tasking sets are described through timeline schemas, i.e. a visual formalism that represents aspects such as: (i) the kind of task-releases (sporadic/recurring); (ii) computational times of each task in terms of best and worst completion time; (iii) priorities; (iv) semaphore synchronization [9]. In Oris this is done through the Timelines Editor. For example, in Figure 3 window a), process P_1 is divided into 2 sequential steps with non-deterministic computation times. In the graphic view, each step is represented by a segment whose length corresponds to the duration of the step. This is divided into two parts: the first (heavy grey) represents the minimum computation time; the second (light grey) accounts for the difference between the maximum and the minimum completion time. Semaphore acquisition and release are schematized by two circles embracing the step.

The user can specify the tasking either by directly manipulating the timeline representation in window a) or by providing tasks parameters in a textual manner in windows c) and d) of Figure 3. Whatever the case, the system automatically generates on the fly the corresponding PTPN model in window b).

As mentioned above the user can also directly build a PTPN model using the TPN Editor, or use the TPN Editor to improve a PTPN model translated from a timeline specification. For instance, referring to our example, we started from the model of Figure 3, corresponding to the initial specification, and then we extended it so that P_1 is given the ability to optionally perform an additional step, according to reception of a message from process P_2. The concept of conditional execution is not supported by timeline schematization, therefore the desired extension has been applied using the TPN editor on the PTPN model generated by the Timelines Editor. Furthermore we have also imposed that the additional step acquires semaphore $mutex2$ to exclusively access a critical section with respect to process P_4.

The resulting final PTPN model is shown in Figure 4: process P_1 is represented by transitions t_{11} through t_{16}. Transition t_{11} releases tasks periodically; t_{12} accounts for the first computation step; transition t_{14} and $HandleMsg$ account for the computation step performed in the critical section ruled by $mutex1$. This path is followed when a message from process P_2 is pending (i.e. a token in place Msg). Transition t_{13} is taken to skip the above path if no message is pending. The remaining step is modeled through transitions t_{15} and t_{16}; t_{15} acquires semaphore $mutex1$, t_{16} accounts for the nondeterministic computation and for the release of the semaphore.

The other processes are modeled in a similar manner. Note that transition t_{21} has a nondeterministic timing so as to account for a sporadic release with minimum interarrival time of 10 time units.

3.2 Simulation and Analysis

Once the model has been built, it can be animated through the TPN Animator, the module which moves tokens by playing the so called "token game". Transitions are fired based on the model semantics [8]. Figure 5 is a screenshot of the animator. Window a) represents the dynamic evolution of the net over time. Appropriate colors have been chosen to identify conditions such as *transition enabled*, *transition firing*, and so on. Window b) reports the lists of firable, enabled and suspended transitions. Window c) lists all the events that have occurred and related times. The animator is helpful for a first understanding of model behavior; a further help comes from conventional batch simulation, which produces statistic data that can be analyzed off-line.

However, simulation does not guarantee correctness. This can be achieved only through the exhaustive analysis of the state space of the model.

3.3 Validation of the model

Referring to our tasking set, consider, for instance, the following formula:

$$\Diamond(p_{13} \neq 0 \wedge (p_{21} \neq 0 \wedge p_{33} \neq 0))$$

stating that a state will eventually be reached in which the marking has a non null number of tokens in all three places p_{13}, p_{21}, and p_{21}. When this condition is true, the system incurs in a *priority inversion* [9]: the high-priority process P_1 is blocked on semaphore $mutex_1$ but, the low-priority process P_3, which holds the semaphore (a token in p_{33}), is suspended by the mid-priority process P_2, which is computing (a token in p_{21}). This is a particularly simple formula,

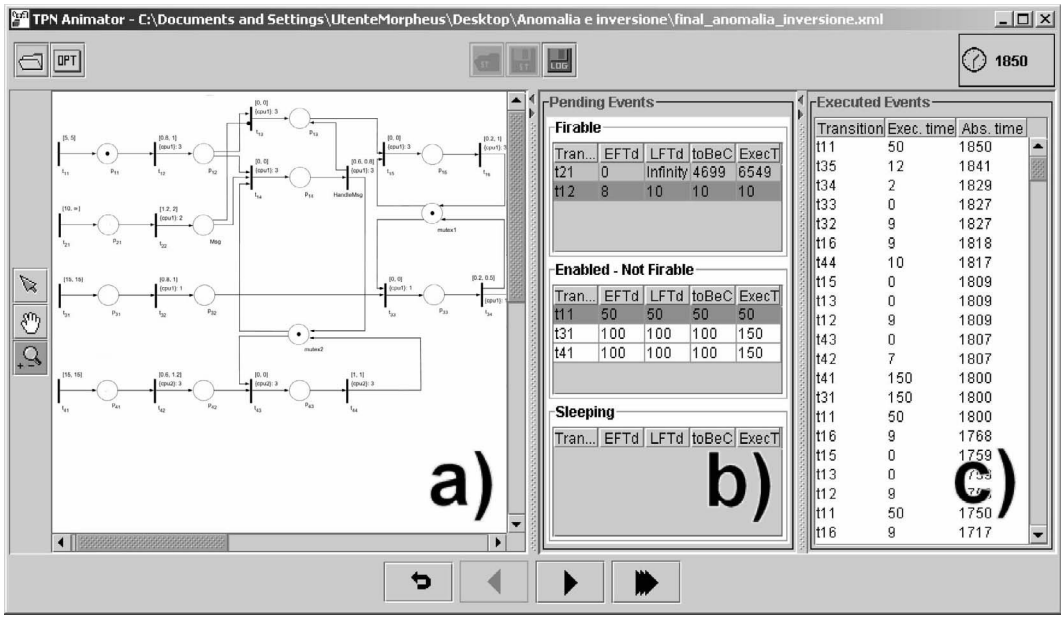

Figure 5. The TPN Animator running on the PTPN model of the example.

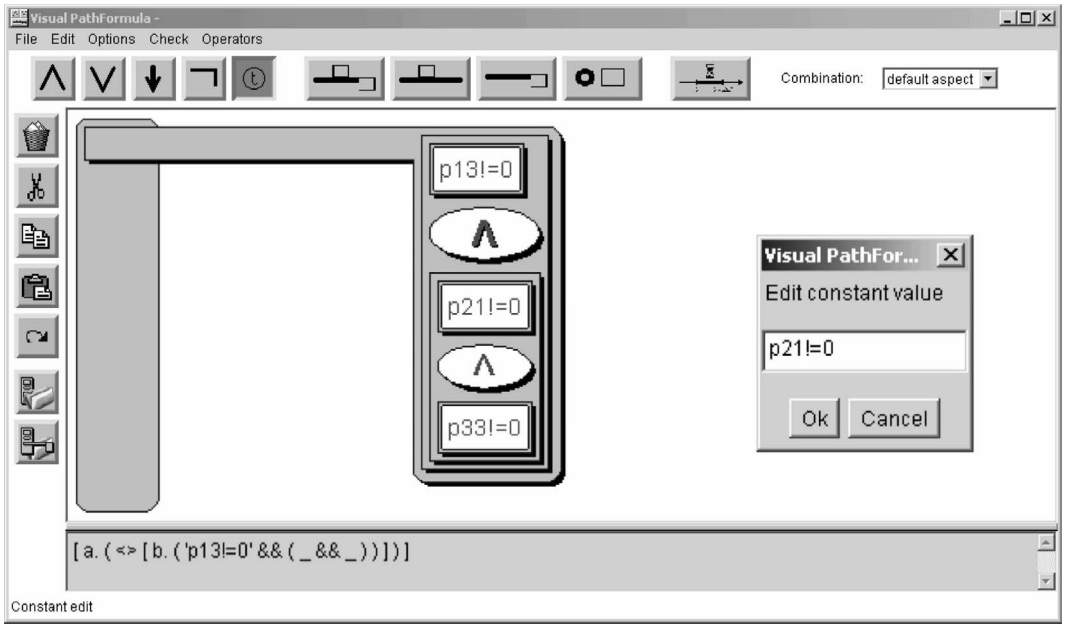

Figure 6. The Formed module at work: the user is graphically building the formula $\Diamond(p_{13} \neq 0 \wedge (p_{21} \neq 0 \wedge p_{33} \neq 0))$**.**

but, in general, temporal logic may lead to very complex clauses.

To simplify the composition of temporal logic formulae, Oris provides the graphic editor Formed. For instance, the above formula is constructed in a graphical manner as in Figure 6. Note that a toolbar of the editor provides a set of logical and temporal operators to ease expression composition.

By using Formed, the designer can establish any number of formal conditions. These are then passed to subsequent validation phases. The Model Checker takes these formulae to label the nodes of the timed class state-space with the set of witnesses which satisfy them. Since witnesses are in the form of symbolic traces, Oris was provided with a Trace Viewer to selectively display them and to explore the interdependencies among variables representing dwelling times in the classes visited along the traces.

The part above the horizontal time axis, in window a) of Figure 7 and Figure 8, visualizes the range of variability of each clock as a timeline. The range of variability can be repeatedly restricted through the introduction of a *lock* which forces any clock to take a predefined value within its acceptable range: thus reducing the range of variability of the profile due to dependencies among clocks. To make this effect evident, the range of variability is drawn with different textures, so as to distinguish the maximum range of variability (dashed line) from the range which is feasible under the locks that have been added (solid line).

The lower part of the main window (part b) of Figure 7 and Figure 8, visualizes a feasible trace timing which satisfies all the locks imposed by the user. For each transition in the trace, the diagram shows a timeline which makes evident the periods in which the clock has been progressing (solid line) or suspended (dashed line). The end of the line indicates whether the transition has fired (arrow termination) or has been disabled (rounded termination).

In general, even after the introduction of a number of locks, the trace may still admit a dense infinity of timings. In this case, non-determinism is reduced automatically according to different possible heuristics: (i) *local latest* heuristic constructs the timing by always selecting the maximum dwelling time in each visited class; (ii) *global latest* selects a timing which maximizes the time of execution of the last event in the trace. *Local earliest* and *global earliest* work in a similar manner.

The deterministic timing selected for the trace induces a partition of the temporal axis, which is marked by vertical lines at transition firings. Regions between two subsequent vertical lines correspond to the classes visited by the trace.

4 Putting it all together

Enumeration of the state-space of the model shown in Figure 4 produces a class graph with 5247 classes. To carry out schedulability analysis, the class graph has been inspected through the Model Checker, by selecting all the traces starting with the release of a process and terminating with its completion. This has led to the identification of 3879, 1777, 17718, and 4072 traces for processes P_1, P_2, P_3 and P_4, respectively. Timeliness analysis through Tracelyzer has shown that all traces of processes P_1, P_2, and P_4 are actually feasible. Whereas, 1304 traces for process P_3 are not feasible under any timing. Timeliness analysis also reports an exact worst case response time equal to 8.1, 5, 12.4, and 2.8, for processes P_1, P_2, P_3, and P_4, respectively. This indicates that the high-priority process P_1 may miss its deadline.

Inspection of the behaviors which attain the worst response time for process P_1 makes evident the occurrence of *priority inversion* in the interval [70,90]: in fact while the high-priority process P_1 is blocked on the semaphore $mutex_1$, the low-priority process P_3, holding the semaphore, is suspended by the mid-priority process P_2 (see Figure 7). To overcome the problem, the priority of the computation of process P_3 in the critical section (i.e. transition t_{34}) is boosted to that of process P_1, according to a priority ceiling emulation protocol.

State-space analysis of the model modified according to this protocol produces a class graph with 5026 classes with 3707, 1738, 18024, and 4080 traces for processes P_1, P_2, P_3 and P_4, respectively. Timeliness analysis identifies 1124 false behaviors for process P_3 and derives tight worst case response times equal to 4.3, 5.3, 12.4, and 2.8 time units for processes P_1, P_2, P_3 and P_4, respectively.

Finally, let us explain how Oris can be used to detect subtle dependencies among process timings. Consider processes P_1 and P_4. We can show that early completion of process P_4 (running on $cpu2$) delays the starting time of the third step of process P_1 (running on $cpu1$). To this end, through the Trace Viewer module, we identify the conditions yielding the worst case response time for process P_1, by forcing its completion event (the firing of t_{16}) at its latest possible time (i.e. by locking the clock of transition t_{16} to be equal to 71, see Figure 8). Under this conditioning, completion of the first step of P_4 (transition t_{42}) is restricted to take a single deterministic value (i.e. the clock of transition t_{42} is restricted to fire at time 10, see Figure 8, while its initial range of variability was [6,12], as reported in Figure 4). In other words, an early completion can change the relative starting time of computations on different processors giving rise to a well-known anomaly in the theory of resource reclaiming for real time scheduling [4] on multiprocessor systems.

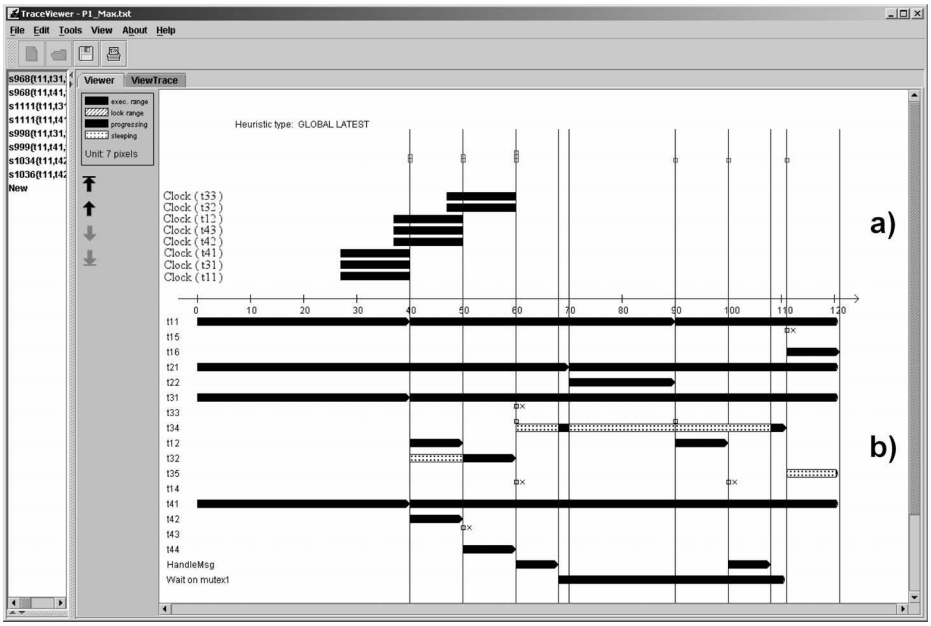

Figure 7. A trace incurring in a priority inversion among processes P_1, P_2 and P_3. In the interval [70,90] the high-priority process P_1 is blocked on the semaphore $mutex_1$ (the fictitious transition Wait-on-mutex1 progressing); the low-priority process P_3 holding the semaphore (transition t_{34} enabled) is suspended by the mid-priority process P_2 (transition t_{22} progressing).

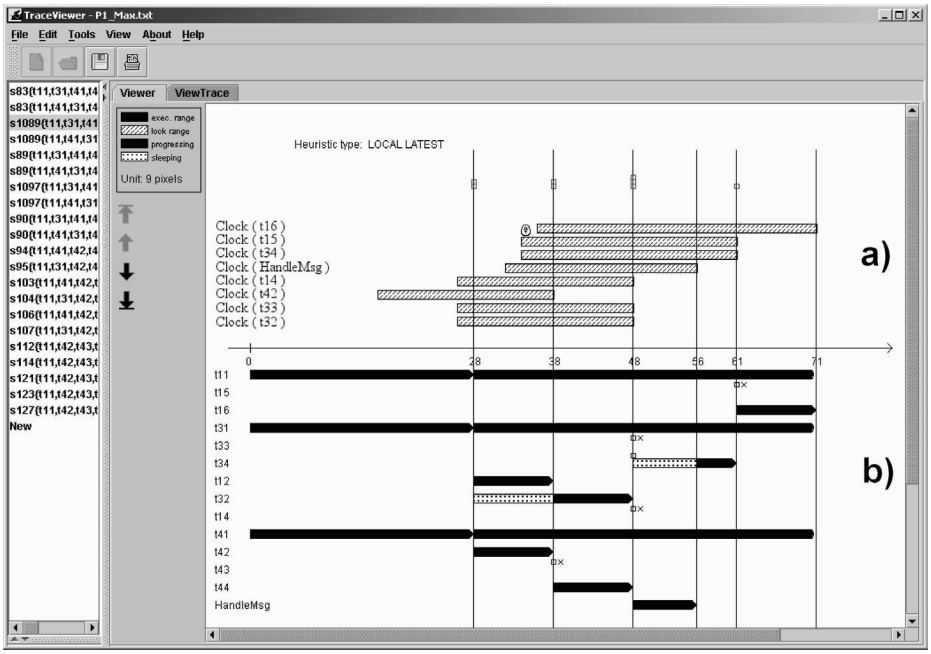

Figure 8. Visualization of the conditions which enable the worst case completion time for process P_1.

It must be remarked that the worst case response time of P_1, i.e 71 time units, can be observed only when the first step of P_4 takes exactly 10 time units. It cannot be observed if the computation time of the first step of P_4 is replaced either with the worst case or through the best case completion time. In other words, characterizing the task set only through worst computation times is not always sufficient to detect worst case timing of the overall system.

5 Conclusions

We have described Oris, a tool for building, simulating, analyzing and validating complex real time systems. The core component of Oris is the analysis engine which performs state-space enumeration based on the theory of Preemptive Time Petri Nets.

The semantics of PTPNs gives Oris a great expressive power, which permits the treatment of practical schedulability problems. As a result, Oris can be used to model complex tasking sets which include different release policies such as recurring, sporadic and one shot. It permits modelling of inter-task dependencies due to the timing of releases, to mutual exclusion on shared resources and dataflow precedence relations. It also permits modelling of internal sequencing of tasks and nondeterministic computation times. This applies to both the case of single and multiple processors systems.

The analysis engine has been implemented in C++, while almost all the remaining modules have been implemented in Java. Oris can be obtained by contacting the authors. Being a research tool, the environment undergoes continuous development. We are currently extending its capabilities to deal with reactive real-time systems, such as robotic workcells, which, in addition to sequencing and timing constraints, also include spatial and kinematics aspects.

References

[1] R.Alur, D.L.Dill, "Automata for Modeling Real-Time Systems," *17th ICALP*, 1990.

[2] R. Alur, T. A. Henzinger, "Logics and Models of Real Time: A Survey," *LNCS*, Vol. 600, pages. 74-106, 1992.

[3] T.Amnell, E.Fersman, L.Mokrushin, P.Pettersson, W.Yi, "Times - A Tool for Modelling and Implementation of Embedded Systems," *LNCS*, Vol.2280, pages 460-464, 2002.

[4] B.Andersson, J.Jonsson, "Preemptive multiprocessor scheduling anomalies," *Proceedings of IPDPS 2002*, pages: 12-19, 2002

[5] J.Bengtsson, K.G.Larsen, F.Larsson, P.Pettersson, W.Yi, "UPPAAL: a Tool-Suite for Automatic Verification of Real-Time Systems," in R.Alur, T.A.Henzinger, E.D.Sontag, editors, *Hybrid Systems III*, *LNCS* Vol.1066, pages 232-243, Springer-Verlag, 1995.

[6] B.Berthomieu, M.Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans.on Soft.Eng.*, Vol.17, No.3, pages 259-273, March 1991.

[7] B.Berthomieu, and M.Menasche, "An Enumerative Approach for Analyzing Time Petri Nets," *IFIP Congress Series*, Elsevier Science Publ. Comp, Vol. 9, pages 41-46, Sept. 1983.

[8] G. Bucci, A. Fedeli, L. Sassoli, E. Vicario, "Timed State Space Analysis of Real Time Preemptive Systems," *IEEE Trans.on Soft.Eng.*, Vol.30, No.2, pages 97-111, February 2004.

[9] G. Buttazzo, "Hard Real-Time Computing Systems," *Norwell, MA Kluwer, 1997*.

[10] F.Cassez, K.G.Larsen, "The Impressive Power of Stopwatches," *LNCS*, Vol.1877, pp.138-152, August 2000.

[11] C.Daws, A.Olivero, S.Tripakis, S.Yovine, "The tool KRONOS," in R.Alur, T.A.Henzinger, E.D.Sontag, editors, *Hybrid Systems III*, *LNCS* 1066, pages 208-219, Springer-Verlag, 1996.

[12] D.Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. of the International Workshop on Automatic Verification Methods for Finite State Systems*, *LNCS*, Vol. 407, pages: 197-212, 1989.

[13] C.L.Liu, J.W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, Vol.20, No.1, 1973.

[14] P.Merlin, D.J.Farber, "Recoverability of Communication Protocols," *IEEE Trans.on Communications*, Vol.24, No.9, Sept. 1976.

[15] L.Sha, R.Rajkumar, S.S.Sathaye, "Generalized Rate Monotonic Scheduling Theory: a Framework for Developing Real Time Systems," *IEEE Proceedings*, Vol.82, No.1, 1994.

[16] E.Vicario, "Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets," *IEEE Trans.on Soft.Eng.*, Vol.27, No.8, pages 728-748, August 2001.